The Android mobile platform

by

Benjamin Speckmann

A Review Paper Submitted to the

Eastern Michigan University

Department of Computer Science

In Partial Fulfillment of the Requirements for the

Master of Science

in Computer Science

Approved at Ypsilanti, Michigan on April 16th , 2008

Professor Matthew Evett

Professor Krish Narayanan

Professor Elsa Valeroso Poh

ABSTRACT

The thesis is a review paper that gives an introduction to the new mobile platform Android as well as a comparative evaluation with regard to other mobile operating systems.

The key topic of this thesis is the categorization of Android.

Therefore it first gives a historical introduction to cell phones and mobile operating systems.

Then it describes the main features of Android for a better understanding of this platform.

In the following theoretical part Android will be compared to the mobile operating systems Symbian OS and Windows Mobile. Features and criteria defined in this part will be considered and included in the comparison of these systems.

The practical part contains a comparison of the Software Development Kits (SDK) from Android and Symbian OS. In this context a simple application implementation on both systems is realized to support this comparison.

Finally an outlook and a conclusion complete this elaboration.

Table of content

List of Figures

List of Listings

Chapter 1 : Introduction

The Personal Computer and the Internet have found revolutionary ways to connect people, to entertain them and let them exchange information. But none of these is able to reach each person anywhere and anytime like the cell phone does.

Based on the company “The Mobile World” in 2007 [MW2007] the global mobile phone usage had exceeded 3.25 billion at the end of 2007 which is equivalent to around half of the worlds population. This shows what a size is behind the brand “cell phone”. Ten years ago nobody would think about a development like this. That this development is going on is shown by a further survey according to the Lemelson-MIT invention index study in the beginning of 2004 [LE2004]. Nearly one in three adults say the cell phone is the invention they most hate but cannot live without. This clearly indicates how the cell phone affects the life of people and how important it is has become in todays society. Considering the results of such surveys, everyone has to ask himself, how it will be possible to take advantage of such a trend.

Google has found perhaps the adequate answer to this question, as they come out with the new open and comprehensive platform for mobile devices called Android. It includes an operating system, middleware, user-interface and applications. It is manufacturer spanning and able to run on every cell phone. Unlike on the market for cell phones, where many manufacturers compete, there are only two main competitors in the domain of cell phone operating systems which are Symbian with “Symbian OS” and Microsoft with “Windows Mobile”. Android must successfully compete with them if Google wants to exist on the mobile market.

Section 1.1 : Motivation

The internet capable cell phone is the future!

– Virpi Roto

This sentence released in a presentation on a technology media event from Nokia called “The Way we live next” attracts interest [NO2007]. In the past it was never clear that the cell phone will have such an important status in today’s society. But the development of the cell phone proves this statement right. Last year Apple’s Iphone came on the market. It includes, besides the normal functionalities, applications like a web browser which allows you to see web pages the way they were designed to be seen. That makes the cell phone more similar to a PC. So not the cell phone itself seems to be the most important thing. It is the operating system and the applications on it which can make the difference in the future. Japan is another example for a successful future of the cell phone. In Japan more people are connected to the internet via their mobile phone than there are PCs with online connections [MW2006].

This massive potential market is now being discovered by one of the biggest internet companies of the world, Google. Google’s approeach is to develop an operating system which can run on every mobile device and not for the mobile device itself shows what their strategy is: Reach as much people as possible.

Section 1.2 : General problem statement

Android makes it easier for consumers to get and use new content and applications on their handsets.

– Andy Rubin

This is a brave statement but meets exactly the problems I would like to address here. The main problem in my opinion is that most cell phone users don't know much about their operating system and its potential. The most common applications like telephone, SMS and in the meantime camera functionalities are widely used. But a cell phone in today's society is not only a tool for telephoning and writing SMS. It is a personal item which provides entertainment and information. It is important to keep in mind that there are different types of cell phone users. I would like to consider three different user groups:

- The normal user who uses only the basic applications provided by the cell phone.
- The advanced user who uses a large part of the provided applications.
- The expert user who tries to get deeper into the cell phone environment, develops applications and uses the total band of functionalities provided by the cell phone.

Each type of user has different needs and expectations. Several questions arise from this: Does the new Android mobile platform fit the needs of the different user groups? Do users really need this new mobile platform or are other existing platforms good enough? To get a significant answer, we will consider Android in comparison to the existing and widely used mobile platforms Symbian OS and Windows Mobile.

Section 1.3 : Goal of the Thesis

This thesis introduces cell phones and operating systems for mobile devices, in general, and evaluates the new mobile platform Android in comparison to the existing and commonly used mobile platforms Symbian OS and Windows Mobile. The intended audience of this thesis are the user groups identified above, including new cell phone users, but also advanced users, who are interested in new information, as well as developers who are engaged in application implementation on cell phones.

In detail, the thesis

- provides an historical overview and introduction to cell phones and operating systems for mobile devices,
- introduces and evaluates the new open mobile platform Android in comparison to the existing and commonly used platforms Symbian OS and Windows Mobile,
- uses the implementation of a simple application on both the Android and Symbian OS to illustrate strength and weakness of the two platforms

and can be used by the reader as an introduction to as well as a support for operating systems for mobile devices. Why Symbian OS is used in the practical part of this Thesis to compare it to Android is explained in Chapter 4.

Section 1.4 : Structure

The paper is divided in five chapters. In detail I will proceed as follows:

- Chapter 1 provides a historical overview of operating systems for mobile devices which builds the foundation for the usage of the Android mobile platform. This introduction is not connected to any product or brand and is therefore usable for everyone.
- Chapter 2 introduces the mobile platform Android in detail.
- Chapter 3 compares Android to the already existing and mostly used platforms “Symbian OS” and “Windows Mobile”. This deepens the understanding and shows the main advantages and disadvantages. For a detailed evaluation criteria are needed which characterize each product. These are portability, reliability, connectivity, product diversity, open system, kernel size, standards, security and special features.
- Chapter 4 discusses and implements the application “TeaTime” on the mobile platform “Android” as well as “Symbian OS”. TeaTime is a “countdown timer application” which allows users to set a countdown depending on the tea they like to drink. The implementation of this application on two different platforms deepens the understanding of programming and also demonstrates the advantages and disadvantages Android has in comparison to other platforms (chapter 3).
- Chapter 5 gives an outlook and a resume which evaluates the main advantages and disadvantages of the Android mobile platform.

Section 1.5 : Historical development of cell phones and operating systems

Before I explain the historical development of cell phones and operating systems it is good to know which needs of its users the cell phone meets. The following needs represent a survey of the most important needs for users in my opinion. They shall not be considered to be complete. Moreover these needs are not required by every user. This is rather dependent on the user's profile.

- A cell phone has to be small. It has to fit everywhere and it has to be possible to carry it at any place where we are.
- A cell phone has to be cheap. It must be possible to get a cell phone also if not much money is available.
- A cell phone has to have functionalities like playing music, making photos or accessing the internet. In addition to telephony and SMS there must be further applications.
- A cell phone has to be comfortable in handling and needs a graphical user interface which is easy to use. We don't want to spend much time with searching functionalities. It has to be clear and well arranged.
- A cell phone has to be modern. This is related to the design as well as handling.

In my opinion it is possible to filter user needs out if you have a look at the historical cell phone development. The changes which are made through the whole evolution shows exactly what user need.

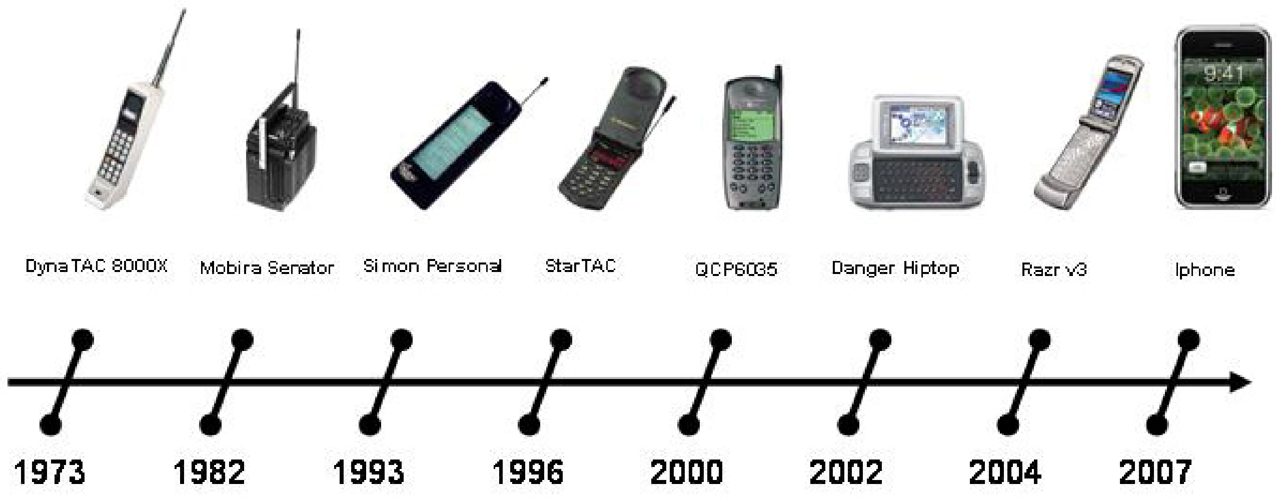


Figure 1: Historical development of cell phones

The first worldwide mobile network was introduced by the USA in 1946 and could only be used nationally at this time, mostly for military purposes. Not until the end of the 1950's was this technique replaced by the Analog network (A-network). Then, in 1973, Motorola presented a prototype of the world's first cellular telephone. It was about one foot long, weight almost 2 pounds and cost $3995. This cell phone which became commercially available in 1983 provided one hour of talk time and could store 30 phone numbers. In only one year 300.000 people, worldwide, were owners, considering price, this was remarkable market growth.

In 1982 the Finnish handset maker Nokia introduced its first Mobile phone, "Mobira Senator". This device looked very much like a portable radio and weight 21 pounds. The first cell phone with PDA features was introduced in 1993 by Bell South/IBM. It included phone and pager functionalities, calculator and calendar applications as well as fax and e-mail capability. The weight was about 18 pounds and it sold for $900. Motorola's "StarTac", in 1996, merged fashion and functionality. Its weight was about 3.1 pound's which is lighter than some of today's cell phones. Kyocera introduces its QCP6035 mobile phone in year 2000. It was the first widely available Palm OS based phone. In 2002 the Danger Hiptop, later known as the T-Mobile Sidekick, was introduced. It was one of the first mobile devices to include a Web browser, reliable e-mail access and instant messaging. With the RAZRv3 Motorola again came back and started a trend towards ultra-thin, stylish phones. It was the first mobile device which many people from high schooler to businessmen wanted to have, primarily, because of its style and because it was fashionably. It is still one of the most popular mobile phones today. The last very impressing innovation was presented by Apple with the release of the iPhone in 2007, a beautifully designed cell phone that includes an innovative touch screen navigation interface [BM2007]. The following table summarizes the most important data of these cell phones:

	Motorola DynaTac 8000x	**Nokia Mobira Senator**	**BellSout/IBM Simon Personal**	**Motorola StarTAC**	**Kyocera QCP6035**	**T-Mobile Danger HipTop**	**Motorola Razr v3**	**Apple Iphone**
Size (cm)	1x0.3	?	?	9.4x5.1	14.2x6.4	11.6x6.5	9.8x5.5	11.5x6.1
Weight (pound)	1	21	18	3.1	0,5	0,4	0.218	0.3
Price ($)	3995	?	900	900	400	?	?	400
Specifics	First cell phone	Úse in cars	PDA functions	First fashion phone	PalmOS based	Web browser	Thin and stylish	Web browser, touch display

Listing 1: Most important data of some cell phones

In a nutshell changes of the hardware related mostly to improvements in weight, price and look. The operating system was improved by e.g. games, calendar applications, email applications and other functionalities to use the internet. This development underlines clearly what a cell phone needs today. It must be as small as possible, it has to be “stylish” and it has to cover a wide range of functions especially internet connection which is used by many people in the world every day.

Chapter 2 : Main features of Android

There should be nothing that users can access on their desktop that they can't access on their cell phone.

– Andy Rubin

This statement [EM2007] given by Andy Rubin, Google's director of mobile platforms, reflects exactly the goal of the Android mobile stack (a stack inculdes a mobile operating system, middle ware and applications). Android is intended to revolutionize the mobile market by bringing the internet to the cell phone and allowing its use in the same way as on the PC.

This second chapter shows the main features of the Android mobile platform and compares it with the commonly used mobile platforms Symbian OS and Windows Mobile.

Section 2.1 : What is Android?

The term "Android" has its origin in the Greek word *andr-*, meaning "*man* or *male*" and the suffix -*eides*, used to mean "*alike* or *of the species*". This together means as much as "being human".

Andorid is a software stack for mobile devices which means a reference to a set of system programs or a set of application programs that form a complete system. This software platform provides a foundation for applications just like a real working platform.

The software stack is divided in four different layers, which include 5 different groups:

- *The application layer*

  The Android software platform will come with a set of basic applications like browser, e-mail client, SMS program, maps, calendar, contacts and many more. All these applications are written using the Java programming language. It should be mentioned that applications can be run simultaneously, it is possible to hear music and read an e-mail at the same time. This layer will mostly be used by commonly cell phone users.

- *The application framework*

  An application framework is a software framework that is used to implement a standard structure of an application for a specific operating system. With the help of managers, content providers and other services programmers it can reassemble functions used by other existing applications.

- *The libraries*

  The available libraries are all written in C/C++. They will be called through a Java interface. These includes the Surface Manager (for compositing windows), 2D and 3D graphics, Media Codecs like MPEG-4 and MP3, the SQL database SQLite and the web browser engine WebKit.

- *The runtime*

  The Android runtime consists of two components. First a set of core libraries which provides most of the functionality available in the core libraries of the Java programming language. Second the virtual machine Dalvik which operates like a translator between the application side and the operating system. Every application which runs on Android is written in Java. As the operating system is not able to understand this programming language directly, the Java programs will be received and translated by the virtual machine Dalvik. The translated code can then be executed by the operating system. A very important notice is that applications will be encapsulated in Dalvik. For every program an own virtual machine is available even if some programs are running parallel. The advantage is that the different programs do not affect each other, so a program error for example can lead to a crash of the program but not of the whole system.

- *The kernel*

  The Linux Kernel will be used by Android for its device drivers, memory management, process management and networking.

The following diagram shows the major components of the Android operating system listed above: [GO2008-2]

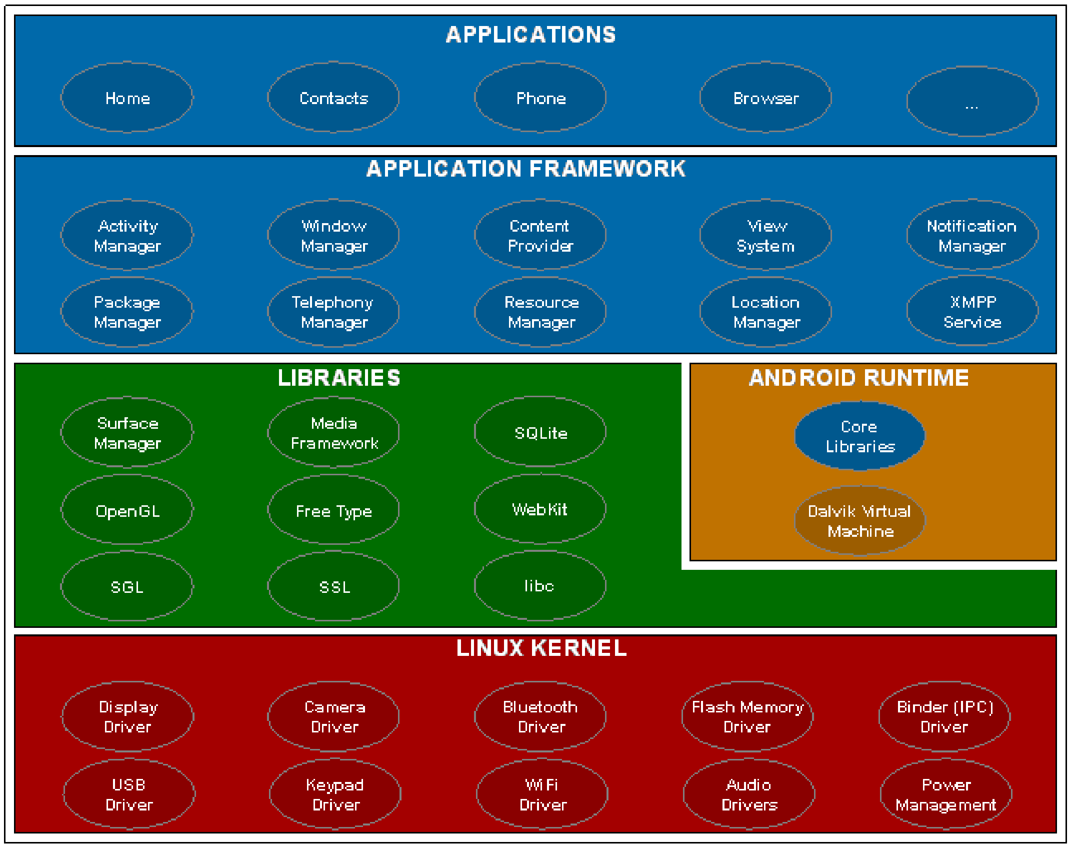


Figure 2: Major components of the Android operating system

Section 2.2 : Important features integrated in Android

Android offers many features cover many areas such as application development, internet, media and connectivity. Some of the most important ones are presented in the following list [GO2008-2].

- **Application framework** enabling reuse and replacement of components
- **Dalvik virtual machine** optimized for mobile devices
- **Integrated browser** based on the open source WebKit engine
- **Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- **SQLite** for structured data storage
- **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM Telephony** (hardware dependent)
- **Bluetooth**, EDGE, 3G, and WiFi (hardware dependent)
- **Camera**, GPS, compass, and accelerometer (hardware dependent)
- **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

Chapter 3 : Android vs. Symbian OS vs. Windows Mobile

The third chapter deals with the comparison between Android and the operating systems Symbian OS and Windows Mobile. Before comparing these operating systems, some basic terminology must be established. This includes the following questions: [VA2004]

- What is an operating system?

  An operating system is an organizational unit within a computer system. It is the interface between applications and hardware. The primary function is the administration of the available operating resources.

- What is a mobile system?

  A mobile system is a computer system which isn't linked to a certain place. It is possible to move it or carry it around like e.g. a cell phone, a handheld or a special computer system in a car. Although there are many similarities between a stationary and a mobile operating system, there are also clear distinctions concerning mobility. An example for an application where a normal operating system is not able to be used is ABS control in a car. An operating system like Windows XP which is not stable enough to guarantee the running of the ABS system over a long time, can not be used. This example points out which attributes are important for a mobile system of any device: The system must be stable and fail-proof.

Section 3.1 : Classification of operating systems for mobile devices

Mobile devices have changed their profile dramatically in the last years. The advanced mobile phones of today integrate fully-featured personal digital assistant (PDA) capabilities with those of a traditional mobile phone. This chapter examines the critical factors for operating systems in this market which differentiate them from each other.

3.1.1 Characteristics of the mobile phone market

The classification of operating systems has to consider the market in which they are used. The market for advanced mobile devices is hard to compare to other markets like the PC market where also operating systems are used. User needs and requirements are different.

Symbian, manufacturer of the mobile operating system SymbianOS, believes that the mobile phone market has five key characteristics that make it unique, and result in the need for a specifically designed operating system [SY2003-1]:

1. *Mobility:* mobile phones are both small and mobile
2. *Universal:* mobile phones are ubiquitous – they target a mass market of consumer, enterprise and professional users
3. *Connection:* mobile phones are occasionally connected – they can be used when connected to the wireless phone network, locally to other devices, or on their own

4. *Innovation:* manufacturers need to differentiate their products in order to innovate and compete in a fast-evolving market.
5. *Open:* the platform has to be open to enable independent technology and software vendors to develop third-party applications, technologies and services

These five characteristics of the mobile phone market underline the difference to other markets where operating systems are used. To be succesful in this market, a product must address these characteristics without limiting functionalities.

3.1.2 Classification criteria

Try to classify operating systems for the purpose of comparison. Technical aspects of these systems have to be considered, also user needs are very important. Because user needs differ the identification of an ideal operating system is not possible. Only a classification or an optimal solution relating to a certain group of individuals is possible.

In the following we will have a look at classification criteria which are important to compare operating systems. For the purpose of clarity I will divide the criteria in several different groups. Looking at the mobile phone market characteristics from chapter 3.1.1 we can deduce the following criteria which are important for a successful operating system.

3.1.2.1 Main criteria

- *Portability*

  Portability is the characteristic of being transportable from one location to another. As to operating system for advanced mobile devices it means the possibility to use the operating system on every cell phone, no matter which brand or type.

- *Reliability*

  Reliability is the ability of a system to perform its required functions under stated conditions for a specific period of time.

- *Connectivity*

  Connectivity is the unbiased transport of packets between two end points. As to operating systems for advanced mobile devices it means the possibility to connect supported by the operating system like wireless, bluetooth or infrared.

- *Product diversity*

  Product diversity is the difference, characteristic or feature which makes the product special. As to operating systems for advanced mobile devices it constitutes the key factor of the system which makes the product unique. The marketing strategy of the producer plays also a major role with regard to this criterion.

- *Open System*

  An Open System is a collection of interacting software, hardware and components with well-defined, publicly available interfaces maintained by a consensus process. As to operating systems for advanced mobile devices it means the free usage and expandability of the system which allows to change it in every possible way.

The above are the main criteria I will address. They are most important for a classification of an operating system in my opinion. They will be explained and discussed in detail. But before that I will address another group of criteria. I would like to expand the main group with a few criteria which I think are also important for a classification of an operating system.

- *Kernel size*

  The kernel of the operating system is the central component which is responsible for memory management, process and task management, and disk management. The size of the kernel is very important to operating systems for advanced mobile devices as it is loaded first and then remains in the main memory of the operating system. This influences the capacity.

- *Standards*

  A standard is a commonly approved or accepted definition or format. An operating systems for advanced mobile devices needs standards concerning programming language, connectivity, data exchange and networking. This is also an important factor for an *open* system.

- *Security*

  Security is the attribute of a system to be safe against attacks or other interference. As to operating systems for advanced mobile devices it means the features of the operating system in order to make it safe in any respect.

- *Special features*

  Special features of operating systems are features which make the difference between them.

These additional criteria combined with the criteria deduced from the characteristics listed in subsection 3.1.1 are adequate in my opinion to build a foundation for a detailed and significant classification of operating systems for advanced mobile phones which can be used by everyone.

3.1.2.2 Further criteria

There are many other criteria which can be consulted to classify an operating system. This criteria are user-dependent, which means that every user assesses this criteria in a different way.

Because of that I will address some other groups of criteria that can be consulted for the classification of the operating systems. These criteria will not be explained in detail because they are either self-explanatory or user dependent. A table a the end uf this subsction displays the criteria. The further criteria are:

Basic criteria

- Public domain or private domain.
- Manufacturer-specific or manufacturer independent.
- User driven or manufacturer driven.
- Functionality is disclosed and well documented or functionality is closed and badly documented.
- Configuration possibility is given or configuration possibility is less given.
- Market alignment

Technical criteria

- Powermanagement

- Multitasking
- Configuration possibility
- Footprint

Usability criteria

- Acts on the assumption of experienced user or acts on the assumption of unexperienced user
- Good usability or bad usability
- Applications
- Driver configuration

User Interface criteria

- Division between operating system and user interface
- Change of user interfaces

Section 3.2 : Comparison of Android, Symbian OS and Windows Mobile

The operating systems Android, Symbian OS and Windows Mobile will be compared using the criteria groups of section 3.1. Each system will be proved numerically. At the end of every evaluation I will award one point for the first place, which is the operating system that fulfills the requirements at most, half a point for the second place and zero points for the last place. The last chapter will be a conclusion where all points of all criteria will be summed up. The number of points will show which operating system is “the best“ in comparison to the others with regard to the main criteria.

3.2.1 Classification based on main criteria

3.2.1.1 Portability

Portability is a very important assessment criterion. Symbian OS has many references in this area and is used on many cell phones and smart phones today. Because of the standadized architecture and the openeness to software from other manufacturers a wide field of operations is available.

Windows Mobile also can run on different platforms with different features. Unfortunately Windows Mobile also has several applications that are specific to certain hardware platforms and therefore are not portable.

The Android Mobile platform is a Linux based system and has the big advantage that this operating system can be used on many different platforms. The open access will help to collect a lot of experience which will make it easier in the future to access other sections. The fact that Android is based on the standardized programming language Java, which is also used for application development, underlines the importance of portability for this platform.

The fact that Symbian mostly runs on Nokia cell phones and that it is not Java based lets it fall behind Android. Also Windows Mobile doesn't reach Android in terms of portability because some applications are hardware platform dependent.

As a result Android gets one point, Symbian OS gets half a point and Windows Mobile zero points.

Total so far:	Symbian OS = 0.5	Windows Mobile = 0	Android = 1

3.2.1.2 Reliability

Reliability is very much dependent on user experience. An operating system can be tested extensively, but without having experience of several years in "the real world" it is very hard to give a good estimate.

The article "Can we make operating systems reliable and secure?" from Andrew S. Tannenbaum [AT2006] deals with the same question. Is it possible to say systems are reliable or not? This article points out two characteristics that make current operating systems unreliable:

- Operating systems are huge
- Operating systems have very poor fault isolation

All operating systems consist of around one million lines of code. Another study which is also taken from the above article deals with the amount of bugs to be found in executable code in average

[AT2006]. Following this study the Linux kernel has probably something like 15,000 bugs and the Windows kernel more than double.

The large size of current operating systems and the big amount of bugs being in every operating system show that it is not possible to understand the whole system as well as to say that the system is totally reliable.

Because of many years of user experience and the amount cell phones working with each of the systems it is possible to say that both, Symbian OS and Windows Mobile, are reliable enough for all kinds of users and applications which are available at the moment. It doesn't mean that both systems run perfectly well but problems with the systems will not result in major difficulties.

Because Android is not available on cell phones at the moment it is not possible to say if the system will be reliable or not. But the Linux kernel, used by Android, has existed for more than a decade and has proven that it is stable and fail-proof. Therefore it is useful for mobile applications. Nowdays it is often used on WebServers or similar applications which require a high degree of reliability. So I think it is possible to say that Android will not rank behind Symbian OS and Windows Mobile regarding reliability.

Android is not available yet, so it is very hard to compare these systems with regard to reliability. Because Symbian OS and Windows Mobile control the biggest part of the market and Android is Linux based I will give every operating system one point. All operating systems are so far developed that the reliability will not differ considerably.

Total so far:	Symbian OS = 1.5	Windows Mobile = 1	Android = 2

3.2.1.3 Connectivity

There are many ways to connect a cell phone to other devices, such as personal computers, the internet or other cell phones. Although we have the possibility to connect our cell phone via cable with the other devices, the mobility of a cell phone generally make a wireless connection preferable. Therefore we only deal with wireless connection in this section. This can be wide area, like connecting to the internet, or personal area which includes infrared and bluetooth links. The operating system has to feature applications that are designed to support all the requirements as well as multi-tasking and the most important communication protocols. It also has to provide a rich set of APIs, which are source code interfaces to support requests for services to be made on it, to ensure that applications can benefit fully from current connectivity possibilities and be easily adapted to take advantage of new protocols as they are implemented.

Symbian OS features GSM telephony, Bluetooth, Infrared and WI-FI. The Symbian OS APIs enables a development that targets all of these features and categories [SY2007].

Windows Mobile features GSM-telephony, Bluetooth, Infrared Communications and WI-FI. The API supports the many features available on the Windows Mobile platform [WI2008-2].

Android features GSM telephony, Bluetooth, EDGE (technique to increase the data rate in GSM-mobile network), 3D (third generation mobile standards), and WI-FI. All developers have the same access to the framework APIs used by the core applications [GO2008-2].

Also with regard to this criterion it can be said that the three operating systems act on the same level in most of the cases. All of them support the common and mainly used connectivity standards. Therefore I will give each operating system one point.

Total so far:	Symbian OS = 2.5	Windows Mobile = 2	Android = 3

3.2.1.4 Product Diversity

Product differentiation is not just a design matter of the operating system. Today a provider of a product has to make sure that it is possible to innovate and develop new product lines. All three providers of operating system which are Symbian, Microsoft and Google have contact to phone manufacturers who are active participants in software development and help to extend the operating system. This helps to develop new functionalities and applications very fast and enhance the whole system.

The most important feature concerning product diversity is to make the relevant product open to the market for development which guarantees product diversity. This is done by all vendors and gives them one point each for product diversity.

Total so far:	Symbian OS = 3.5	Windows Mobile = 3	Android = 4

3.2.1.5 Open platform

Android was built from the ground up with the explicit goal to be the first open, complete, and free platform created specifically for mobile devices.

– Open Handset Allicance

This vision of the Android Mobile Platform is located on the web site of the Open Handset Allicance [OHA2008]. What does “open platform” really mean when the OHA says that Android is the first one?

Symbian [SY2003-2] and Microsoft [WI2008-1], describe their platforms as open systems differently:

> *Symbian OS is the world-leading open operating system that powers the most popular and advanced smartphones from the world’s leading handset manufacturers.*
>
> – Microsoft

or

> *The Windows Mobile platform is an open platform that supports needs beyond mobile messaging.*
>
> – Symbian

A clear definition for the term “open platform” is needed before it can be decided which operating system fulfill this criterion. As there is no common definition available I will list criteria which are important in my opinion to characterize and define an open platform. This definition is not neccessarily complete. Nevertheless, it serves as a general definition for the thesis.

An open platform:

- allows developers to implement additional functionality to the system. That includes access to every API and other source code.
- allows developers to re-implement and replace functionality or the whole operating system. It helps to make an individualized, interactive system and content.
- needs standards to guarantee high quality.
- should have no costs for using the platform, develop applications for the platform or publish own developed applications.
- uses a programming language with an open standard like Java.
- offers multifaceted ways for communication and connectivity.
- is usable on all mobile devices.

Summarizing these criteria yields the following definition of an open platform:

> *An "open mobile platform" is a software stack, including an operating system, middleware and key applications, which can be used on every mobile device. It allows users to develop additional software and change or replace functionality without limitations. The most common standards for communication and connectivity are used. All these functionalities have to be free of charge.*

The only operating system which really fits to these criteria is the Android mobile platform. All operating systems achieve several standards for communication and offer a software development kit (SDK) that allows to build applications for it. But only Android is based on a free available operating system which is based on a Linux Kernel. Another fact is that publishing your own developed applications is free which is not the case for Symbian OS and Windows Mobile. Those systems require payment for code signing, or need a special program for code developing for example. This is the reason why Android gets one point and the other operating systems half a point.

Total so far:	Symbian OS = 4	Windows Mobile = 3.5	Android = 5

3.2.1.6 Kernel Size

An often used assessment factor for comparing the kernel size is the "Memory footprint" which is the amount of memory used by the operating system. For a significant classification we need to find the operating system with the lowest "Memory Footprint" which in turn maximizes the performance of the operating system. Symbian OS has about 200 kbyte minimal memory requirement. The Windows Mobile platform is built on top of Windows CE which requires for a typical installation about 300 kbyte minimal memory. The Android operating system which is a Linux kernel will need about 250 kbyte. All the data above apply to an installation with the basic and minimal fucntionalities possible [KD2006]. As a result Symbian OS needs less memory than Android which needs less memory than Windows Mobile. So Symbian gets one point, Android gets half a point and Windows Mobile zero points.

Total so far:	Symbian OS = 5	Windows Mobile = 3.5	Android = 5.5

3.2.1.7 Standards

Standards in general make the platform more open and attractive for developers. If standards exist it is easier for everyone and especially for developers, to get to know the new system. Therefore we need

- a standard for internationalization like Unicode,
- a standardized programming language like Java,
- a standardized network protocol like TCP/IP,
- a standard for email exchange like POP3, IMAP4 or SMTP,
- a standard for sending textmessages and mutimedia messages like SMS and MMS,
- a standard for data communication between devices like Bluetooth, Infrared or OBEX,
- a standard which helps to make internet content visible for slow data rates like WAP,
- and a standard for data synchronization like SyncML.

Most of the major standards are supported by all three of the operating systems. Every operating system uses the most common standards concerning networking, e-mails, messaging and communication, but only Android is based on the standardized programming language Java. This is also the only programming language used to develop applications. The advantage of Java is that its programs can run on any platform without having to be rewrited. This is also a positive aspect of portability. As a result Android gets one point, Symbian OS and Windows Mobile each half a point.

Total so far:	Symbian OS = 5.5	Windows Mobile = 4	Android = 6.5

3.2.1.8 Security

Symbian OS offers a new platform security architecture which provides a platform with the ability to defend itself against malicious or badly implemented programs. The architecture consists of two high level components: "Certificate management" and "Cryptography". These two modules form the basis for a number of high level components which include Certificate management control panel item, Software installation and Secure communications [SY2002]. Windows Mobile also has its own Security Model. It contains a combination of security policies, roles and certificates to address configuration, remote access and application execution. These features control access to a device. If a user or an application, for example, is allowed to access, security policies control the boundaries for the actions, access and functionalities [MS2007]. Android is a multiprocess system, where each application (and parts of the system) runs as its own process. Most security between applications and the system is enforced at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications. Additional finer-grained security features are provided through a "permission" mechanism that enforces restrictions on the specific operations that a particular process can form [GO2008-1].

Every platform has its own security model that covers the most important actions concerning software installation, secure communication etc. This makes them equal and results in one point for each operating system.

Total so far:	Symbian OS = 6.5	Windows Mobile = 5	Android = 7.5

3.2.1.9 Special Features

This section deals with features or applications which are designed to make the system unique. The Android mobile platform has significant advantages in this case. The new integrated browser based on the open source WebKit engine allows to access web pages through the internet the same way as through the PC. Also the virtual machine Dalvik optimized for mobile devices, is a feature which enables every application runs in its own process. This is very important for stability and reliability issues.

Windows Mobile has, due to its outstanding position in the computer market, the advantage that the synchronization between the PC and the cell phone is very easy. Symbian OS however has no special features which must be mentioned in my opinion. The communication with the Internet and the Personal Computer will play a major role in the future with regard to mobile platforms. Therefore Android gets one point, Windows Mobila half a point and Symbian OS zero points.

Total so far:	Symbian OS = 6.5	Windows Mobile = 5.5	Android = 8.5

3.2.2 Classification based on further criteria

This section contains a list with further criteria which help to classify the three mobile operating platforms. I divided the criteria in 4 different groups which are “Basic criteria”, “Technical criteria”, “Usability criteria” and “User interface criteria”. The user can utilize these to personalize his choice of an operating system for his mobile device.

3.2.2.1 Basic criteria

	Android	**Symbian OS**	**Windows Mobile**
Public domain or private domain.	Public domain	private domain	private domain
Manufacturer-specific or manufacturer independent.	manufacturer independent	manufacturer specific	manufacturer specific
User driven or manufacturer driven.	user driven	manufacturer driven	manufacturer driven
Functionality is disclosed and well documented or functionality is closed and bad documented.	Functionality is disclosed and well documented	Functionality is disclosed and well documented	Functionality is disclosed and well documented
Configuration possibility is given or configuration possibility is less given.	Configuration possibility is given	Configuration possibility is less given	Configuration possibility is less given
Market alignment	universal	smartphone market	universal

Listing 2: Basic criteria of operating systems

3.2.2.2 Technical criteria

	Android	Symbian OS	Windows Mobile
Powermanagement	Yes	Yes	Yes
Multitasking	Yes	Yes	Yes
Configuration possibility	high	low	middle
Footprint (min. memory requirements)	approx. 250 kbyte	approx 200 kbyte	approx. 300 kbyte

Listing 3: Technical criteria of operating systems

3.2.2.3 Usability criteria

	Android	Symbian OS	Windows Mobile
Acts on the assumption of ex-/unexperienced user.	Acts on the assumption of unexperienced user	Acts on the assumption of unexperienced user	Acts on the assumption of unexperienced user
Good or bad usability	good	good	good
Applications	high	middle	middle
Driver configuration	good	good	good

Listing 4: Usability criteria of operating systems

3.2.2.4 User Interface criteria

	Android	**Symbian OS**	**Windows Mobile**
Division between OS and user interface	Yes	Yes	Yes
Change of user interfaces	possible	hardly possible	hardly possible

Listing 5: User interface criteria of operating systems

Section 3.3 : Conclusion

In section 3.2 the Android platform, Symbian OS and Windows Mobile were compared with regard to the main criteria for a mobile operating system. Every criterion was explained in detail and applied to the three operating systems. At the end of every criterion classification points were given from one to zero. The best operating system relating to the criterion got one point, the second got half a point and the third zero points. The total sum of points added for each operating system will show which of them is the "best" with regard to the main criteria. The following table shows the results in detail:

	Android	**Symbian OS**	**Windows Mobile**
Portability	1	0.5	0
Reliability	1	1	1
Connectivity	1	1	1
Product diversity	1	1	1
Open system	1	0.5	0.5
Kernel size	0.5	1	0
Standards	1	0.5	0.5
Security	1	1	1
Special features	1	0	0.5
Result	**8.5**	**6.5**	**5.5**

Listing 6: Result table of operating system comparison

With regard to the main criteria the new Android mobile platform gets most points. It is the only truly “open system” which makes the major difference in my opinion. Also the special feature like the Web browser and the virtual machine Dalvik play a major role in this comparison. Standards in connectivity, portability and security are more or less achieved by every operating system in the same way.

For a meaningful evaluation with regard to the “Further criteria” I will go back to the main user groups defined in section 1.2.:

- The normal user who uses only the basic applications provided by the cell phone.
- The advanced user who uses a large part of the provided applications.
- The expert user who tries to get deeper into the cell phone environment, develops applications and uses the total band of functionalities provided by the cell phone.

These groups will now serve as example groups. They represent a large number of users. It should be said that all operating systems have a very high standard and can be used by every user group. No company develops an operating system for cell phones which is usable only by a special group of users. A successful product must meet the needs of all user groups. The following analysis regarding the user groups has to be seen as an indication respectively a personal evaluation based on true data and experience which kind of users would be more satisfied with what kind of mobile operating system.

The **normal user** needs an operating system with which he is familiar. Options to configurate the system as well as develop applications for it are not necessary. Important is a good documentation, easy and familiar usability/handling and just the basic features like telephone, SMS or camera usage. All these features are supported by all three operating systems.

Nevertheless I would recommend the operating system Windows Mobile for the normal user. The most important point for this decision is the user's familiarity with the system. Windows sets standards with its operating systems for personal computers in usability and handling. Operating systems like Windows XP run on the majority of personal computers, therefore many users are close to this system. The look and feel, the structure and the handling of Windows Mobile is similar to the PC based operating system. That makes the user feel comfortable and familiar and enables an easy access to it.

The **advanced user** needs an operating system that offers not only the basic features like SMS, telephone or camera usage, he needs functionalities which allow the advanced usage of the system as defined above. Advanced usage includes functionalities like data exchange, synchronization and usage of applications like calendar or e-mail synchronization programs. Also design and usability features are very important for this user as well as technical criteria like multitasking. This allows e.g. to hear music and check e-mails at the same time. For this user it is very hard to find an adequate solution. Every operating system seems to be appropriate. Depending on the operating system he uses for his personal computer (which will most of the time be Windows) I would recommend Windows Mobile or Symbian OS. Both, especially Windows Mobile, offer fully developed tools to synchronize or update the cell phone operating system with the applications on the computer. Calendar or e-mail programs like Outlook will be updated very easily through a standardized program which is already on the cell phone. The problem Android has at this time respectively why I don't recommend it to that user group is the following: It does not yet completely fit to the requirements of the advanced user. As it is a very new system without practical experience it needs more efforts and time for a further development like for example PC synchronisation. When this development is achieved there will be no question for me to recommend this system also to the groups of normal and advanced users.

The **expert user** needs an operating system that offers not only the basic features and tools. In addition he wants to discover the operating system, find solutions for common problems like data exchange and tries new ways. He also attaches much importance to design and wants to use his cell phone like a computer. Also personalization of the operating system which includes changes with regard to the operating system as well as own developed applications are very important for this user. A very good documentation is also not required, because he likes to try things out. Technical criteria like memory requirements, multitasking and power management are of particular importance.

The right choice for this user is only the mobile platform Android. This operating system is the only truly open system which allows major changes inside the operating system and offers common basic standards for programming. Also the new Web browser which allows the usage like a normal PC based browser is important.

It is again important to realize that all operating systems can be used by everybody because all operating systems meet the requirements of a modern mobile operating system. The Android will be more challenging for the user compared to the other two systems while there is little experience with it at the moment. Therefore it provides for new features like the web browser. This fits perfectly to the present general trend and could therefore attract many users.

The above findings suggest three conclusions:

- there is not (yet) the ideal mobile operating system, but their usefulness differ in relation to user requirements. So users should be aware of their requirements and the functionalities they really want and then choose the adequate operating system.
- Android can threaten the market dominance of the other two systems when it achieves more practical experience and the system is further developed. Then it will have the potential to become attractive not only to the expert user, but also to the normal and the advanced user. This could make the mobile operating systems Windows Mobile as well as Symbian OS superfluous.
- We have seen that the requirements of the user groups as definded above are much different. Producers of cell phones could be well advised to differentiate their cell phones according to user groups in order to meet their specific requirements even better.

Chapter 4 : Tea Time - Application implementation with Android- and Symbian OS SDK

The fourth chapter of this elaboration deals with a comparison of the two Software Development Kits (SDK) from Android and Symbian OS platform “S60”. A simple application will form the basis for this comparison.

At the beginning we will have a look at the installation process, the system requirements for an installation and the usage of both SDKs. The installation process includes the SDK itself as well as the Integrated Development Environment (IDE) or other tools which are required to develop applications. Next, a list of developer features related to these SDKs will be presented. Based on own experiences through the application implementation the list will be put to the proof and expanded if necessary. A closer look at the APIs of both platforms which are mainly used for the implementation of the TeaTime application will give a deeper understanding where the differences of both APIs are. At the end a conclusion will complete this chapter.

Section 4.1 : Why compare Symbian OS to Android?

For a valuable analysis of an SDK it is not only important to have a theoretical look at its features, you also need practical experience with them. For that I will compare two products which serve the same functionalities and operate on the same user group and market.

The mostly installed operating system is Symbian OS. It runs on almost every Nokia cell phone which is the world's leading mobile phone supplier and on many others like Motorola or Samsung. This high market penetration makes Symbian OS a very good candidate for a comparison with Android. Another very important reason for Symbian OS as a comparison is the possibility to develop mobile applications in Java. Primarily C++ is used to develop Symbian applications but it is also possible to use Java 2 Micro Edition (J2ME) which is a Java based environment for applications running on mobile devices. These applications are then portable across many devices which is the same with Android applications. The use of a standardized programming language like Java is another major criterion for the comparison of Android with Symbian OS.

Section 4.2 : Installation process

4.2.1 System requirements

None of the SDKs have special or exceptional hardware requirements. All requirements will be fulfilled by almost every computer these days, even though there are some differences. The Android SDK is able to run with a lower processor and less RAM requirements than the Symbian OS SDK. There are also major differences regarding software requirements. Both SDKs require a Java Runtime Environment version 5 or higher but Symbian OS needs an additional Active Perl installation. Another advantage of the Android SDK is that it can be installed on all common operating systems like Windows, Mac or Linux whereas Symbian OS SDK has to be installed on a Windows machine. This is a disadvantage in an open market as it is required today. The IDEs recommended by both companies are the common standards with Eclipse, NetBeans or Apache Ant. This is not decisive for a comparison. The following table shows the minimum system requirements needed to develop Android- respectively Symbian OS-applications briefly:

	Android	**Symbian OS**
Hardware Requirements (minimum)	200 megahertz online processor	1 GHz processor (minimum IDS and OS requirement of processor is 500 MHz)
	32 MB RAM and 32 MB Flash	512 MB RAM (minimum IDE and OS requirement is 128 MB of RAM)
Software Requirements (minimum)	JDK 5 or JDK 6 (JRE alone is not sufficient)	Java Runtime Version 1.5.0 or newer Active Perl Version 5.6.1 or newer
	Windows XP or Vista Mac OSX 10.4.8 or newer (only x86) Linux (tested on Linux Ubuntu Dapper Drake)	Microsoft Windows XP SP2 or Microsoft Windows 2000 SP4.
	Supported Development Environments (IDE): Eclipse 3.2 (Europa) or newer and JDT, WST and ADT plugins for Eclipse. Apache Ant 1.6.5 or newer.	Supported Development Environment (IDE): Eclipse 3.2 or newer and EclipseME version 1.7.7 or newer. NetBeans IDE with Mobility Pack 5.5 or newer.

Listing 7: Minimum system requirements of Android and Symbian OS

4.2.2 Installation

As a detailed installation instruction is included in both SDK packages I will not explain in detail how the installation is done. A description of my experience with the actual installation of both systems will be more helpful for a useful comparison.

The environment to develop applications for Android consists of the Android SDK, the IDE Eclipse and the Java Development Kit (JDK) which has to be preinstalled for the installation of both, Android SDK and Eclipse. The following versions of the tools mentioned above are used:

- *Android SDK:* android-sdk m5-rc15
- *JDK:* jdk1.6.0_01
- *Eclipse:* eclipse 3.3.2 (europa)
- Eclipse plugin: ADT plugin
- Underlying operating system: Windows XP

The installation of all these components worked without a problem and needed about 5 minutes. The Android developer page is clearly designed with direct links to the sources needed. The whole page is much focused on understanding and using Android, what makes it very easy to get in touch with it. Google seems to go on with their concept of making information available in a simple and focused way comparable to their search engine that is also just focused on the search functionality. After installing the SDK, which is done by simply extracting the downloaded ZIP-file in a folder, the path to the SDK has to be set in the path variable at your environment variables. The Eclipse is also installed by just extracting the downloaded ZIP-file to a save place. A very good option in using Eclipse as your IDE is the automatic installation of the Eclipse plugin for using Android with it. You only have to use the automatic update functionality which downloads and installs the plugin from a URL allocated by Google. This URL is the following:

- https://dl-ssl.google.com/android/eclipse/

After installing this plugin you are free to go on with the development of Android applications.

The development Environment for Symbian OS consists of the Symbian SDK, the IDE Eclipse, the Java Runtime Environment (JRE) and Active Perl. Active Perl and the JRE have to be preinstalled for an installation of the Symbian SDK. The following versions of the tools mentioned above are used:

- *Symbian SDK:* S60 3rd edition SDK with Feature Pack 2 (Symbian OS v9.1)
- *JRE:* jre1.6.0_01
- *Eclipse:* eclipse 3.3.2 (europa)
- Eclipse plugin: EclipseME
- *Active Perl:* ActivePerl-5.10.0.1002
- Underlying operating system: Windows XP

Symbian offers three different kinds of SDKs depending on the cell phone. There is the UIQ (User Interface Quartz) used preferably by Sony Ericsson and Motorola, the S60 (Series 60 User Interface) preferably used by Nokia and Nokia Series 80 also preferably used by Nokia. The S60 platform is the leading smartphone platform in the world and is therefore picked as the comparative platform to Android.

The installation of all these components worked not as fast as the installation of the environment for Android. It took about 45 minutes to install everything. The Symbian SDK- and the Active Perl installation, which are both executable files, took 90% of the time. The executable files create a Wizard which guides you through the installation process. This is a nice feature for absolute beginners but not essential. Another disadvantage in comparison to Google's installation process is the way information and sources are presented and made available. None of the software is directly available on the Symbian website. All sources are linked to their manufacturer or user (for example Nokia). Both websites, Symbian as well as Nokia, are overloaded with information nobody needs for application development and installation of the SDK. This makes the finding process of sources much harder and costs much more time. The installation of Eclipse is the same as with Android and the automatic update functionality for the Eclipse plugin is also given. The URL where Eclipse downloads the plugin is the following:

- http://www.eclipseme.org/updates

The installation of Active Perl is very straight forward but is another additional expense in comparison to Android. The installation documentation which is available for both SDKs is helpful and should be used for the installation.

Section 4.3 : TeaTime – An application description

For a useful comparison of an application development process the question came up which simple application to develop. This application shall run on both platforms equally and shall use some of the major functionalities offered by both SDKs. Graphical and technical features are not required as they do not contribute to a simple comparison because of the different premises of the SDKs. The requirements which should be met by the application shall be restricted to the following. The application should be able to:

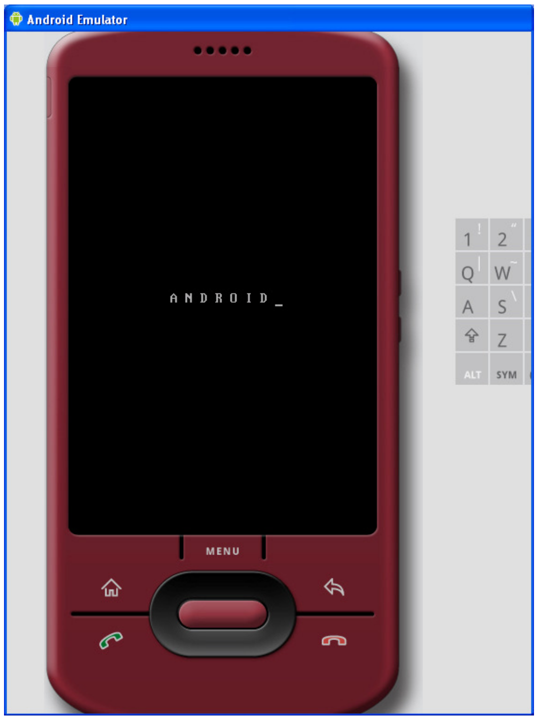
- display text,
- display input fields which can be used to save data put in by the user,
- use the most important functionalities of a persistent storage like adding or deleting content,
- show a functionality like a count-down timer.

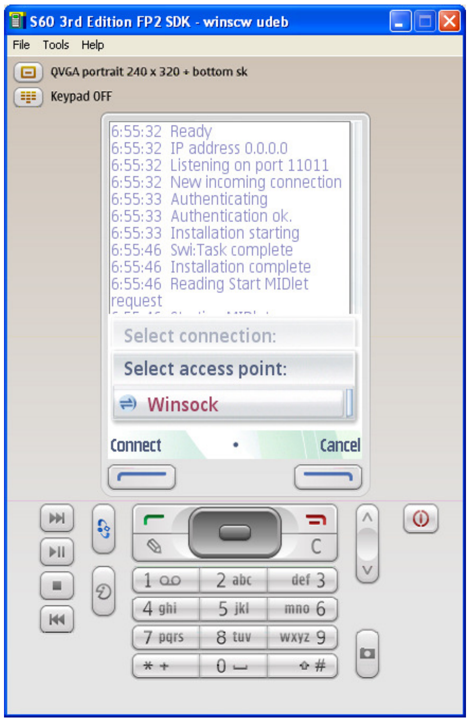
TeaTime is an application which fulfils all the requirements and doesn’t go beyond the scope of this elaboration. It uses the basic functionalities which should be supported by a Software Development Kit and makes it possible to compare both SDKs form Google and Symbian.

The TeaTime application allows to define teas and their brewing time and can be used as count-down timer and reminder.

The following screenshots show how the TeaTime application looks like, realized with Android and Symbian OS:

Before the application starts, the emulator has to be initialized.




S60 3rd Edition FP2 SDK - winscw udeb
File Tools Help
QVGA portrait 240 x 320 + bottom sk
Keypad OFF

```
6:55:32 Ready
6:55:32 IP address 0.0.0.0
6:55:32 Listening on port 11011
6:55:32 New incoming connection
6:55:33 Authenticating
6:55:33 Authentication ok.
6:55:33 Installation starting
6:55:46 Swi:Task complete
6:55:46 Installation complete
6:55:46 Reading Start MIDlet
request
```

Figure 3: Android emulator - initialization

Figure 4: Symbian emulator - initialization

After starting a list of tea sorts will be displayed on display1. The tea data shown come from a persistent storage.

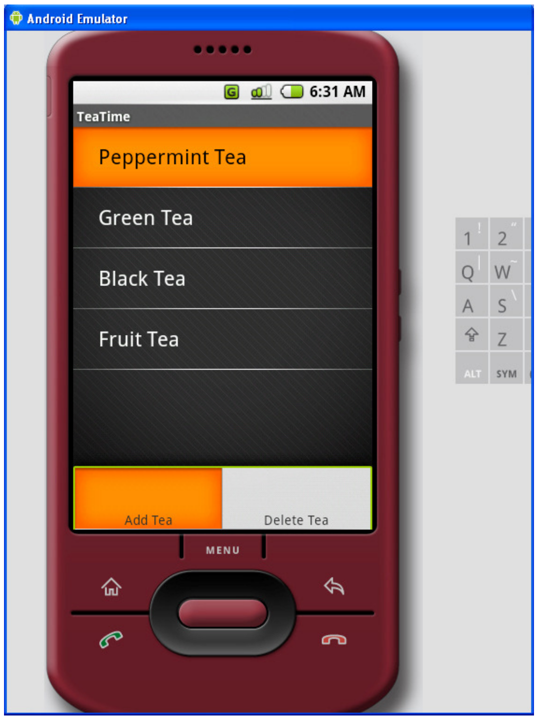


Figure 5: Android emulator – List of teas

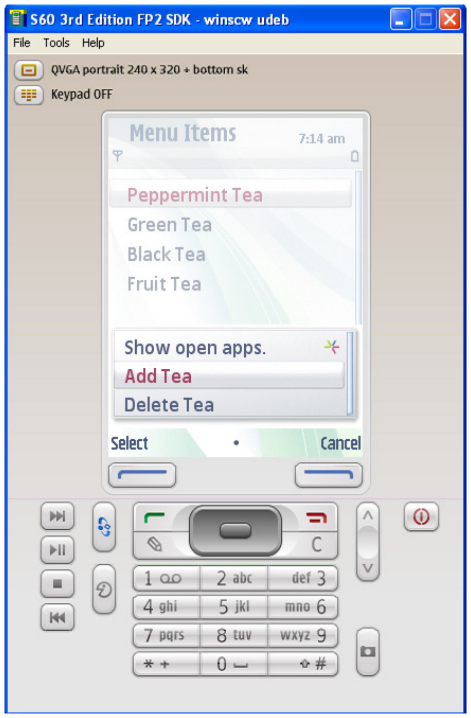
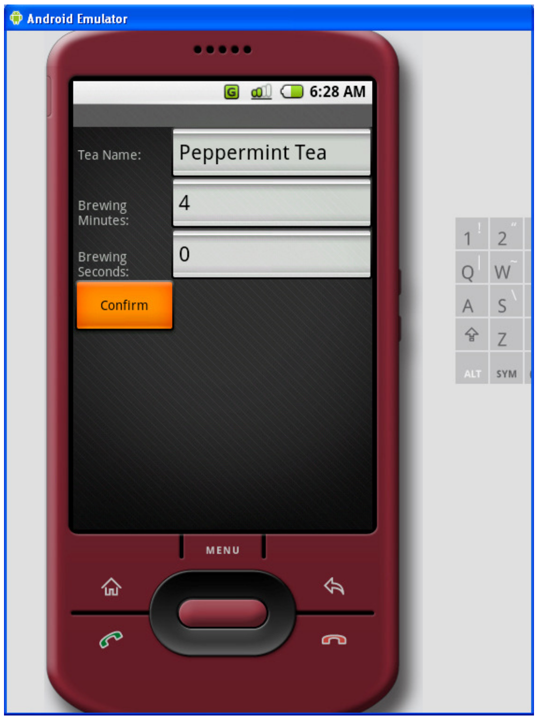


Figure 6: Symbian emulator – List of teas

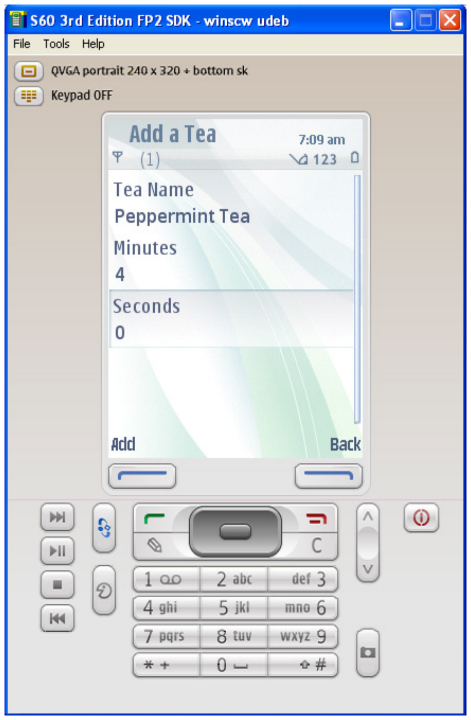
Display1 offers four command opportunities to add a tea, to delete a tea, to enter the count-down timer display for each tea sort and to exit the application. Pressing command “Delete” on display1, the selected tea sort will be deleted out of the persistent storage. Using the button also reloads the screen. The updated list of tea sorts will be displayed then.

On command "Add", display2 appears. Three input fields for defining the name of the tea, the minutes and the seconds for the brewing time are displayed. Display2 offers a button to save the data in a persistent storage. Using this button the data will be saved and you return automatically to display1 with the updated list of teas.



Tea Name: Peppermint Tea
Brewing Minutes: 4
Brewing Seconds: 0

Figure 7: Android emulator – Add a tea


S60 3rd Edition FP2 SDK - winscw udeb
File Tools Help
QVGA portrait 240 x 320 + bottom sk
Keypad OFF
7:09 am
Add
Back

Add a Tea
Tea Name
Peppermint Tea
Minutes
4
Seconds
0

Figure 8: Symbian emulator – Add a tea

Selecting the name of a tea sort on display1 makes you go to display3 which in fact is the count-down application. The minutes and the seconds which are defined for every tea in the persistent storage are displayed on the screen depending on the tea sort you select.

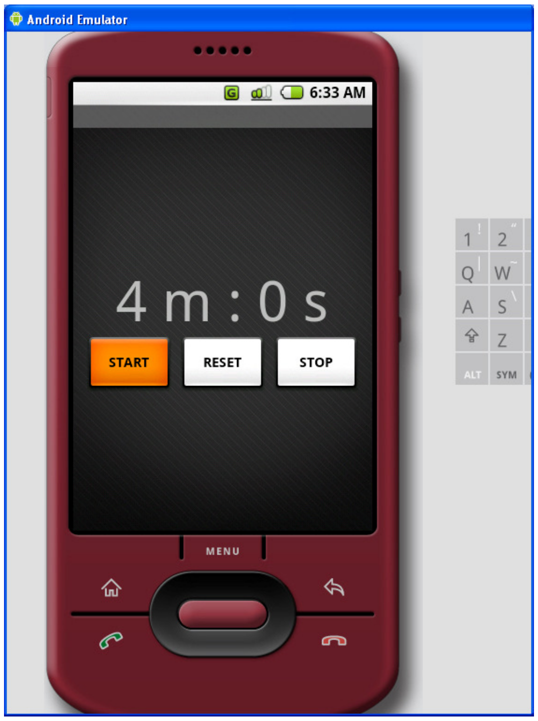


Figure 9: Android emulator – countdown timer

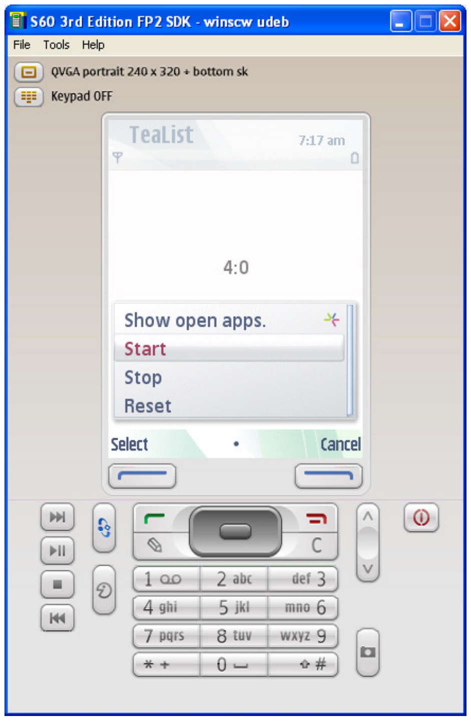


Figure 10: Symbian emulator – countdown timer

Display3 offers four command opportunities to start, stop or reset the count-down timer and to cancel the timer and go back to display1, which is on Android the arrow in the loewr right corner. The time which is shown on this display comes from the minutes and seconds defined in the persistent storage. The minutes and seconds depend on the tea you selected earlier.

In the next part of this practical chapter we will have a look at the two implementations of the application. A comparison table showing developer features between Android SDK and S60 SDK will serve as a support which guides us through this implementation and helps us with the comparison. Some of these features will be questioned critically in order to consider them with the experience got through the process of implementation. Also features which do not come up in the table but become important through the implementation process will be considered.

Section 4.4 : Life-cycle of an Android and a Symbian application

Before we begin to have a closer look at the Software Development Kits of both platforms it is important to know which application model is used by Android and the S60 platform. The application model defines how an application is managed and how management responsibilities are divided between the application and the underlying system. It can also be described by the following questions:

- How is an application started and stopped?
- When can an application give access to system resources?
- How does it discover its initialization parameters?

The Android application model is defined by four building blocks which make up an application. These components are:

- **Activity**: Class that presents the User Interface of an application.
- **Intent Receiver**: Class that handles execution of code in the application in reaction of external events.
- **Service**: Class that runs in the background, not interacting with the user.
- **Content Provider**: Class that implements standard methods to let other applications store or retrieve data handled by the content provider.

All these components have to be listed in a file called AndroidManifest.xml. This XML file contains all components of the application as well as their capabilities and requirements. It is not necessary for a complete application to include all components. The TeaTime application for example uses the Intent Receiver to react on button usage and the Activity to display the user interface.

The Activity is the most common component of an Android application. It describes usually a single screen on the cell phone. Because most applications consist of multiple screens the same amount of Activities has to be created for it. The TeaTime application for example needs a screen to display a list of teas, a screen with input fields to add data to the database and a screen that displays the count-down timer. Each of these screens has to be implemented by an Activity. Moving between these screens is accomplished by starting the Activity. You do that with the method "startSubActivity()". The layout of an Activity can be designed using XML files. Each activity class has an XML layout as a view screen which represents the activity class. The structure of these XML layout files is very simple. It is a tree of tags where each tag is the name of a View class. So you can connect every element you create in your activity to the XML layout file. This model is inspired by the web development model, which allows you to separate the presentation of your application (its User Interface) form the application logic.

Application development on the S60 platform is, as I mentioned before, based on the Java 2 Platform Micro Edition (J2ME). Supported by it is the "MIDlet Model" which is used for the application implementation of TeaTime. A MIDlet is a Java program which has to extend the abstract MIDlet class found in the javax.microedition.midlet package. The main class defines three life-cycle notification methods which have to be overridden by the MIDlet. These methods are the following:

- startApp()
- pauseApp()
- destroyApp()

Another important point in understanding the MIDlet life-cycle is the state in which a MIDlet can be found. The three possible states are the following:

- **paused**: The MIDlet instance has been constructed and is inactive
- **active**: The MIDlet is active
- **destroyed**: The MIDlet has been terminated

The MIDlet's initial state is *paused*. When the system activates the MIDlet the startApp() method will automatically be invoked. It creates and displays the application's user interface. After it returns from startApp(), the MIDlet's state changed from *paused* to *active*. If there is any problem with the initialization of the MIDlet an Exception will be thrown and a shift to the *destroyed* state follows.

Section 4.5 : Software Development Kit comparison

4.5.1 Developer features between the SDKs

The following table is an abridgement of a comparison chart [NP2008] which shows important relational features of the SDKs. These features only concentrate on the use of the SDKs. This abridgement helps us to compare both of them. The most important points of this chart have to be questioned and compared with the experience made in the implementation process of TeaTime. These features must not to be seen as complete. For a more valuable comparison we have to take a closer look at the Application Programming Interface (API) and the opportunities this API features. But before we go deeper into the API lets have a look at the following table:

	Android	**S60**
Costs	Free	Free; could be more depending on tools used
Wide-scale app availability	Depends on device availability	Now
Native development	Yes	Yes
Languages supported for native development	Java	C++
Digital certificates	No	Available, required for some phones
Retail support	No, but Android Developer Challenge offers money and	Limited

	publicity	
Platform maturity	Immature	Mature
First-party support	Yes	Yes
Community support	Excellent	Excellent
Application installation method	Unknown; installation on emulator is not reflective of production devices	Direct, PC suite
Emulator available	Yes	Yes
Remote debugging	Yes	Yes
Target device variety	Poor (that will change, though)	Good
Touch screen support	Single touch	No
App availability and variety	Poor (that will change, though)	Excellent
Underlying architecture	Linux	Symbian
Flash availability	No	Yes
Java availability	Yes	Yes

Listing 8: Developer features of Android SDK and Symbian S60 SDK

Some of these features have already been discussed in previous chapters and will therefore only be briefly mentioned.

Both SDKs offer a free development of applications. Just for the usage of high quality tools like Visual Studio, if you develop in C++, extra costs will accrue from using S60 SDK . The support for both systems especially at the time where I developed the TeaTime application was excellent. My experience for example with the major discussion groups/forums was excellent. Fast and valuable answers helped very much. The discussion groups are accessible by the following websites which are also linked on the company's websites:

- Android discussion groups: http://code.google.com/android/groups.html
- Nokia discussion board (for Symbian S60): http://discussion.forum.nokia.com/forum/

For an application installation on the Android platform Google offers, unlike to the information in the table above, the Android Debug Bridge called adb. It is a versatile tool that lets you manage the state of a device or emulator. The Eclipse also offers a plugin called ADT plugin which installs the applications .apk files automatically on the device or emulator. The experience I had with application installation on the emulator was just positive. It ran the first time I tried. Symbian respectively Nokia offers the PC suite which is the official synchronization application of Nokia between a Nokia cell phone and the PC. Beside the installation of the applications .sis files you can also transfer any kind of data with this software. For the installation of the TeaTime application on the emulator there is also a plugin available which allows you to deploy the developed Java MIDlets without problem on the emulator. An advantage for Android is the possibility to use a touch screen for handling the operating systems user interface. An example touch screen application is already implemented on Android's emulator. The touch screen interface for the S60 is officially announced for some time in year 2008. The availability of application to extend the operating system is poor with regard to Android at the moment. This is the consequence of the advantage of experience Nokia or Symbian have on the mobile market. Google started a developer's challenge where they spend about 10 million dollars prize money to heat up the development of free available applications. The missing Flash availability on the Android platform will attract attention for example when you use the web browser. Many web pages have Flash elements which then can not be seen on the Andorid web browser. But since the SDK is out for free development, Macromedia will for sure be thinking about initializing Flash for the Android platform. Symbian is able to show flash but only on the cell phone itself because there is no real web browser available.

To sum it up it can be said, without having a detailed look at the APIs, that a good application development is possible on both systems by comparing the features above. Android offers more interesting possibilities with the web browser or the touch screen for all different kinds of applications. The challenge will show what is exactly possible with this new platform. To get an even better possibility to compare the platforms, we need to have a look at the APIs. This will be done in the following regarding the requirements of the application defined in chapter 4.3.

4.5.2 The APIs

Google's Software Development Kit supports a relatively large subset of the Java Standard Edition 5.0 library. This includes amongst the following standard APIs:

- *java.io* – provides classes for system input and output through data streams
- *java.net* – provides classes for implementing networking applications
- *java.security* – provides classes for the security framework
- *java.util* – provides classes for collections framework, legacy collection classes, event model, date/time facilities and internationalization
- *javax.sound.midi* – provides classes for interfaces and classes for Input and Output, sequencing and synthesis of MIDI data (Musical Instrument Digital Interface)
- *org.apache.http* – provides classes for standard information transfer.

Things like a printing API which are normally given by the Java Standard Edition 5.0 library were left out because they didn't make sense or better APIs were available that are specific for Android like user interfaces. Important APIs developed by Google itself are related to Android specific functionalities. Some APIs in this field are:

- *android.graphics* – provides low level graphic tools like canvases, color filters etc.
- *android.media* – provides Media Player and Recorder
- *android.net* – provides help for network access, beyond the normal java.net API

- *android.opengl* – provides OpenGL utilities for 2D/3D graphics
- *android.telephony* – provides tools to make, receive and monitor phone calls
- *android.database.sqlite* – provides classes to manage its own private database
- *android.view* – provides classes to expose basic user interface classes that handle screen layout
- *android.widget* – provides UI elements to use on the application screen
- *android.content* – provides classes for accessing and publishing data on the device
- *android.app* – provides high level classes encapsulating the overall Android application model

An overview of all classes available by the Android API is directly linked on the Android Developer's page.

The SDK for Symbian OS S60 is provided by Nokia and is based on the Java 2 platform Micro Edition (J2ME). This J2ME platform is extended and enhanced by the Mobile Information Device Profile (MIDP). MIDP provides for the core application functionality required by a mobile application. This includes the user interface, network connectivity, local data storage and application life-cycle management. This whole functionality is packaged as a standardized Java Runtime Environment and a set of Java technology APIs. The MIDP API encompasses the four core

Connected Limited Device Configuration (CLDC) packages (java.lang, java.io, java.util, javax.microedition.io), plus the following three MIDP-specific packages:

- *javax.microedition.lcdui* – provides a set of features for implementation of user interfaces for MIDP.
- *javax.microedition.midlet* – defines MIDP applications and the interactions between the application and the environment in which it runs.
- *javax.microedition.rms* – The MIDP provides a mechanism for MIDlets to persistently store data and later retrieve it.

In addition to these packages, MIDP also adds four new classes to the core CLDC packages:

- *java.util.Timer* – Used to schedule tasks for future execution in a background thread
- *java.util.TimerTask* – Used by java.util.Timer to define tasks for later execution
- *javax.microedition.io.HttpConnection* – Interface that defines everything necessary for HTTP connection
- *java.lang.IllegalStateException* – A RuntimeException that indicates that a method has been invoked at an illegal or inappropriate time.

The CLDC and the MIDP are both standard-based technologies for developing applications that run on mobile devices. There are more APIs available which extend the functionality of the Java ME platform. This APIs will not be considered in this elaboration because they are irrelevant for the development of the TeaTime application. The most important ones will be listed in the following:

Mobile Media API (MMAPI) – provides for audio, video and other time-based multimedia support to resource constraint devices

- *File Connection API* – provides access to file systems residing on mobile devices
- *Bluetooth API* – provides the functionality to communicate with other devices using bluetooth wireless technology and OBEX protocol for Object Exchange
- *Mobile 3D Graphics API* – provides access to a high level library to manage and run3D scenes and objects inside the application
- *Wireless Messaging API (WMA)* – provides platform-independent access to wireless communication resources like SMS and MMS

A comparison of the APIs shows that Google as well as Symbian (respectively Nokia) provide a large variety of functionalities which can be used for appliation implementation. Both provide for basic APIs to display text and simple graphics as well as high level APIs for 3D graphics. Also the important standards for connection and communication like bluetooth or wireless are supported by both sides. Symbian has the advantage of experience and know how which is guaranteed through the J2ME API development that is done by many big companies like Motorola, Sony Ericsson, Nokia or Texas Instruments. Googles API offers on the first look everything you need for comfortable application implementation but one has to keep in mind that it is not tested under real life conditions over years whereas Symbian is.

4.5.3 Practical comparison between the APIs

After we have got an overall view of about the most important APIs provided by the SDKs we will have a closer look at the APIs I used to develop the TeaTime application. I will divide the TeaTime application in different minor applications and then apply the different APIs on the respective application parts. The minor applications are the following:

- Show a list of items
- Achieve an action by pressing a button
- Show normal input fields
- Give access to a persistent storage
- Realize a count-down timer

4.5.3.1 Show a list of items

Android:

For displaying items on the screen you have to use the following APIs:

- andorid.app.Activity
- android.widget.SimpleCursorAdapter
- android.app.ListActivity

The code for the implementation of a list looks like this:

```
// Create a Cursor
      myTeaCursor = myDBHelper.callUpAllTeas();
      startManagingCursor(myTeaCursor);
// Create an array to specify the fields we want to display in the list
      String[] from = new String[]{TeaDB.KEY_NAME};
// and an array of the fields we want to bind those fields to
      int[] to = new int[]{android.R.id.text1};
// Now create a simple cursor adapter and set it to display
      ListAdapter tea = new SimpleCursorAdapter(this,
        android.R.layout.simple_list_item_1, myTeaCursor, from, to);
      setListAdapter(tea);
```

Listing 9: Android source code – Show a lis of items

At the line 2 and 3 a Cursor is defined called “myTeaCursor” which is set to the beginning of a result list defined by a database expression. In line 5 the array “from” which specifies the fields we want to display in the list is created. Because we just want to show the tea name in the database we define “KEY_NAME” which is the tea name field in our database. The second array “to” in line 7 contains the tag name in the corresponding XML file we want to bind the names of array “from” to. The ID “android.R.id.text1” defines the id in our Output XML-file where the list has to be displayed. Because it is just the name which has to be displayed only one ID is needed. The final step is done by defining a SimpleCursorAdapter in line 9 to 10. In this SimpleCursorAdapter the context “this”, the layout “simple_list_item_1” which is a standard Android layout for lists, the cursor “myTeaCursor” and the arrays “from” and “to” which contain the data for the User Interface and the views to which the data has to be bound have to be defined.

Symbian OS:

For a list implementation in Symbian OS S60 API you have to use the following APIs:

- javax.microedition.lcdui.List
- javax.microedition.rms.RecordStore
- javax.microedition.lcdui.Display

The code for the implementation of a list is the following:

```
1   // Get the number of teas saved in the RecordStore
2         teaNumber = rs.getNumRecords();
3   // Create a String array with the size of the teanumber
4         String[] teaList = new String[teaNumber];
5   // Save all names of the teas in the string array
6         teaList = db.callUpAllTeas(teaNumber, rs);
7   // Define a display
8         display = Display.getDisplay(this);
9   // Define a list “menu” and save all names from the string array in it
10        menu = new List("Menu Items", Choice.IMPLICIT);
11        for (int i = 0; i < teaNumber; i++) {
12              menu.append(teaList[i], null);
13        }
14  // Display the list “menu” on the screen
15        display.setCurrent(menu);
```

Listing 10: Symbian Source code – Show a list of items

First a string array has to be created which contains the list with tea names in line 4. For that the number of teas which is in our persistent storage is needed. We get this number when we execute the method “getNumRecords()” on our RecordStore “rs”. This is done in line 2. What a RecordStore exactly is will be explained in the minor application “Add or Delete items in the persistent storage”. In this tea list all the tea names available in our RecordStore will be saved to the String array in line

6. The following line 8 defines a Display object which is unique to this MIDlet. This object is able to display a List object on the screen called “menu” (see line 15) which is created in line 10. To create a List object “menu” we just have to define it and append the names from our String to it done in line 11 and 12.

Conclusion:

Displaying a list with both APIs is very simple. An advantage in my opinion is that Android separates the view and the logic by using XML files which makes the code more manageable. The usage of the SimpleCursorAdapter API in Android enables displaying the list without using any loops which is very comfortable and fast. How such a XML file might look can be seen in the minor application “Show normal input fields” in this section or the applications source code in the appendix.

4.5.3.2 Achieve an action by pressing a button

Android:

For implementing a command button you need the following APIs:

- android.view.menu
- android.view.menu.Item

For this implementation the following code is needed which defines a delete button in the Android Menu and reacts when the Menu button is selected:

```
// Create the Menu button “Delete”
public boolean onCreateOptionsMenu(Menu menu) {
      menu.add(0, DELETE_ID, R.string.menu_delete);
      return true;
}
// Activity Handler for the menu button
public boolean onMenuItemSelected(int featureId, Item item) {
      switch(item.getId()) {
      case DELETE_ID:
            delete();
            return true;
      }
      return super.onMenuItemSelected(featureId, item);
}
```

Listing 11: Android source code – Achieve an action by pressing a button

Both methods are available through the Android API and will be executed automatically by starting the Activity. The OnCreatOptionsMenu() method (line 2) creates a simple menu button by just calling the menu.add() method (line 3). A group (0), an ID and a title, which is a link to a resource file, have to be defined for this method. The onMenuItemSelected() method (line 7) will be called automatically by selecting any button on the screen. Comparing the IDs of the buttons inside this method (line 8 and 9) allows us to assign any action we want to the button. In our case we choose the ID "DELETE_ID" and the action "delete()" (line 10) which deletes a tea out of the database.

Symbian OS:

For implementing a command button you have to use the following APIs:

- javax.microedition.lcdui.Command
- javax.microedition.lcdui.CommandListener

The following code is needed to implement the command activity with the Symbian SDK:

```
// Define a Command object
static Command deleteCommand = new Command("Delete Tea",
                                                Command.SCREEN, 2);
// Use the standardized method add a button to the menu and activate
the CommandListener
public void startApp() throws MIDletStateChangeException{
      menu.addCommand(deleteCommand);
8          menu.setCommandListener(this);
9    }
10   // Standardized method will be called by selecting a button
11   public void commandAction(Command c, Displayable d){
12         String label = c.getLabel();
13         if (label.equals("Delete Tea"))
14               delete();
15         }
16   }
```

Listing 12: Symbian source code – Achieve an action by pressing a button

In the variable declaration of the class (line 2) a Command "deleteCommand" is created. A name, a pre-defined command and a priority number has to be defined for this object. In the "startApp()" method (line 6), which is a predefined method of a MIDlet that will always be called when the MIDlet is executed, the command has to be added to the menu (line 7). This menu will then be set to the CommandListener (line 8) which guarantees that the "commandAction()" method (line 11) will be automatically called when the delete command is selected. In the "commandAction()" the label of the button, which is selected, will be read (line 12). Based on this result the delete method is called (line 13 and 14).

Conclusion:

Both APIs offer a standardized way to define buttons and actions to these. They both use an Activity Handler or Listener which reacts automatically if a button is selected. In this case it is not possible to decide which way I would prefer because they are more or less identical.

4.5.3.3 Show normal input fields

Android:

For the implementation of input fields following APIs have to be used:

- android.widget.EditText

The following code is needed to implement the input fields:

```
// Create a EditText object
      myNameText = (EditText) findViewById(R.id.name);
// set the View to the layout defined by the id “tea_add” in the XML
      setContentView(R.layout.tea_add);
// The Output XML File
<LinearLayout android:orientation="horizontal"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">
      <TextView android:layout_width="100sp"
            android:layout_height="wrap_content"
            android:text = "Tea Name:  "
            android:layout_marginLeft="5px" />
      <EditText android:id="@+id/name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"/>
</LinearLayout>
```

Listing 13: Android source code – Show normal input fields

Line 2 shows the definition of the EditText object “myNameText”. A unique ID will be assigned to that object which helps to bind the object to the correct tag in the Output XML File. The method “setContentView()” (line 4) defines the XML file which has to be used as the output file for this Activity. In this example it is the XML file “tea_add.xml”. The XML file needs to define the Layout style (line 6), which is linear, a “TextView” tag for the name of the input field (line 9) and the “EditText” tag for the input field itself (line 13). The “EditText” tag has the same “id” we defined for the EditText object in the Activity. This assigns the object in the Activity class to the input field defined in the XML file.

Symbian OS:

For the implementation of input fields following APIs have to be used:

- javax.microedition.lcdui.Form
- javax.microedition.lcdui.TextField

The following code is needed to implement the input fields:

```
// Create a Form for displaying the content
      Form teaForm = new Form ("Add a Tea");
// Create a textfiel for the input fiels
      TextField name = new TextField ("Tea Name", "", 20,
                                                  TextField.ANY);
```

```
6   // A stadardized method which is automatically called. The textfield
7   appends to the Form object which can be displayed on the screen
8   protected void paint(Graphics g) {
9          teaForm.append(name);
10         display.setCurrent(teaForm);
11  }
```

Listing 14: Symbian source code – Show normal input fields

To display an input field you first have to create a Form object "teaForm" which defines the frame of the screen (line 2). This object "teaForm" can be appended to a TextField (line 9) which is created in line 4 . A name, the size of the textfield and the value type have to be defined for the Textfield. The Method paint() which is a standardized Method for a MIDlet is responsible for showing the input field on the screen (line 8). This will be done by the "setCurrent()" method which appends the form, containing the text field to the display Object (line 10).

Conclusion:

On the first glance the solution for displaying textfields provided by the Android APIs seems to be more complex than the solution given by the Symbian API. But the first impression is wrong. You need just 2 lines of java code to define an input field in the Activity. The many lines defined in the XML file are not as bad as it seems. The tags are very easy to implement and to understand and the separation of the view from the logic is worth it.

Access to a persistent storage

Android:

Using a persistent storage the following APIs have to be used:

- android.database.Cursor
- android.database.sqlite.SQLiteDatabase

Android provides a SQLite database which is a fully relational database management system. You can use standard SQL to handle with it which is very comfortable.

The following code is needed if you want to manage the SQLite database:

```
// Standardized method to create the database
      myDb = mCtx.createDatabase(DATABASE_NAME, DATABASE_VERSION, 0,
                                                                    null);
// Standardized method to open the database
      myDb = mCtx.openDatabase(DATABASE_NAME, null);
// Standardized method to insertn a row to the database
      myDb.insert(DATABASE_TABLE, null, initialValues);
// Standardized method to delete a row out of the database
      myDb.delete(DATABASE_TABLE, "_id" + "=" + id, null) > 0;
// Standardized method to execute a SQL query to the database
      myDb.query(true,  DATABASE_TABLE, new String[] {
```

```
12                              KEY_ROWID, KEY_NAME, KEY_BREWTIME_MINUTES,
13                              KEY_BREWTIME_SECONDS}, KEY_ROWID + "=" + rowId,
14                              null, null, null, null);
15 }
```

Listing 15: Android source code – Access to a persistent storage

The API of Android for database management exposes fully methods to manage a SQLite Database. Writing of standard SQL is not necessary any more because all standards method like create, delete or execute SQL code are implemented by pre-defined methods.

In line 2 a Context in which the Database is created has to be defined. A definition of the name and the version of the database is enough to create it. Open the database also requires the Context object and the database name like it is done in line 5. To insert values into the database a definition of the database name and the content values is needed (line 7). The content values, in our example named by “initialValues”, are an object of type ContentValues. This ContentValues object holds all values of one row which have to be inserted to the database. The values can be added to the ContentValues object with the method add(). The method to delete a complete row out of the database is provided in line 9. The name of the database, the “where Clause” and the “where Argument” to execute this command successfully has to be defined. The last method provided in line 11 to 14 demonstrates a standard “Select command”. It queries the given table and returns a Cursor over the result set. A distinct value (true/false), the table name and a String array consisting of the table, the columns, the selection, the selection arguments, the groupBy argument, the having argument and the orderBy argument can be defined in this method.

Symbian OS:

Using a persistent storage the following APIs have to be used:

- javax.microedition.rms.RecordStore
- javax.microedition.rms.RecordFilter
- javax.microedition.rms.RecordEnumeration
- javax.microedition.rms.RecordComperator

Symbian doesn't provide a fully relational database management system like Android does. Their solution of a persistent storage is called RecordStore. A RecordStore is able to save an array of bytes. All values have to be converted into byte arrays before they can be added to the RecordStore.

The following code is needed if you want to manage a RecordStore:

```
// Open a RecordStore or create it when it doesn't exist
      RecordStore rs = RecordStore.openRecordStore("MyTeaDatabase",
3                                                                          true)
4   // Delete a Record in the RecordStore
5         rs.deleteRecord(recordId);
6   // Get a Record out of the RecordStore
7         b = rs.getRecord(id);
8   // Insert a Record into a RecordStore
9         RecordStore rs;
10        ByteArrayOutputStream baos = new ByteArrayOutputStream();
11        DataOutputStream outputStream = new DataOutputStream(baos);
12        outputStream.writeUTF(teaName);
13        outputStream.writeInt(brewMinutes);
14        outputStream.writeInt(brewSeconds);
15        byte[] b = baos.toByteArray()
16        rs.addRecord(b, 0, b.length);
```

Listing 16: Symbian source code – Access to a persistent storage

The API of the S60 for RecordStore management exposes also standardized methods to manage the RecordStore like insert, update or delete Records in a RecordStore.

To create and open a RecordStore a method is provided shown in line 2. The name of the RecordStore and a boolean value have to be defined for that. The boolean value defines if the RecordStore should be created if it doesn’t exist. Also deleting and reading a record out of the Record Store is very straight forward (see line 5 and 7). The ID of the Record has to be defined in the methods. Much more effort is adding a Record to the Record store. The RecordStore only accepts byte values which requires a compilation of the values you want to store into byte values. You first have to create a ByteArrayOutpuStream "baos" (line 10) and a DataOutputStream "outputstream" (line 11). In this "outputstream" you can put in the values with methods like writeUTF() for text or writeInt() for integer values (line 12 to 14). The ByteArrayOutputStream object then helps to save the data in the byte array by commute them with the method toByteArray() (line 15). After that the byte stream can be saved into the RecordStore with the method "addRecord()" (line 16).

Conclusion:

As you can see both APIs provide a standardized pool of methods for managing their solution of a persistent storage. Each of the mostly used database querys can be done by invoking one method of the Android API. Also the advanced J2ME API used by Symbian offers for the most databases querys standardized methods. But inserting a record in the record store for example causes much more effort. The reason for this is the record store itself which is only able to store byte values. Therefore a byte stream has to be created. The values have to be saved into it and then inserted into the record store. In comparison to the one method solution provided by Google, 5 lines of Java code are needed in the solution used by Symbian.

Section 4.6 : Conclusion

In the fourth chapter we looked at the installation process, the TeaTime application itself, the life-cycle of both Android and Symbian applications and developer features including a detailed consideration of the APIs. This leads us to a better understanding what is needed for a successful application development on both systems. It also helped us to compare both ways and possibilities of the whole application development process.

The installation process of the whole environments was easier than expected. Both systems have a simple way with detailed descriptions available at the company websites even if the installation of the Symbian environment took about 40 minutes longer. This is referred to the different installation ways. The Android installation files are coming in ZIP format. After extracting them at a save place, the path in the environment variables to that sources has to be set. The Symbian installation files are coming in executable format. Wizards are created which guides you through the installation process. This is very comfortable but it takes a long time.

The application life-cycles, which are coined by the Activity in Android and the MIDlet in Symbian OS, are easy to understand. Both classes contain standard methods for starting, pausing and destroying the application. The solution of providing content to the user is in my opinion better implemented by Google’s Android. The source code for the view, which is implemented by the XML standard, is separated from the logic of the application. This brings clarity and better reusability of the code and is oriented to the MVC (Model View Control) pattern which also implements a separation of code. This should be the way for source code has to be developed in the future.

The APIs of both systems seem to fulfil the possibility to use the common standards in graphics, connection, data transfer or communication. The only serious difference between those APIs is the realization of a persistent storage. Android offers a real SQLite database which can be used by API methods or by normal SQL commands. Symbian realized the persistent storage with a solution called RecordStore which allows saving byte streams. The RecordStore is much smaller then a real database but also even slower. If you just need a small amount of data the usage of a RecordStore makes no big difference. If you have an application which is not using a small amount of data only, like an e-mail program which can store a lot of e-mails and contacts, a real database should be required. Nevertheless one also has to keep in mind that the API used by Symbian is much more matured. It is used since years and is enhanced by several big companies like Motorola, Sony Ericsson or Texas Instruments. The API of Android is available since the end of 2007 which is not a long time to get proved. The functionality which I used for the emulator was very good and in comparison to the API of Symbian not worse. There is a difference in running an application on an emulator or a real device.

Nevertheless the application development on the Android platform was convincing. The very simple installation and the very good documented API as well as the perfect example documentation to all kinds of basic problems made it easy to get to know to the system and to use it. The implementation process of Symbian was as good as the implementation process of Android but the longer process and the unclear way of presenting sources, documentation and example code made the application development more difficult and took much more time. The first impression which is in my opinion very important and very formative for the advanced impression was also more positive working with regard to Android. This includes not only the documentation and the examples mentioned before.

Even things like design and usage of the emulator play a role. The look and feel of Android's emulator is modern and contemporary and the application handling (e.g. deleting applications) is done with one shell command.

Looking at the results of chapter four and based on my own experience I would prefer and recommend the application development on Android.

Chapter 5 : Conclusion

The goal of this masterthesis is to describe, characterize and categorize the new mobile operating platform Android. While working on this elaboration the author made positive as well as negative experience, which shall be reviewed in the following conclusion. At the same time it helps the reader to decide which mobile operating system might be convenient for him.

The elaboration is divided in a more theoretical part where technical features are considerd and a more practical part where an implementation of a simle application helps to assess Android.

The systematic assessment of the technical features as well as features like openness of standards comes to the result that Android is superior to its competitive systems. When one adds the category “user needs” to this assessment then Symbian OS as well as Windows Mobile can have advantages for specific user groups. But even then one has to take into account that Andorid is not available on cell phones yet. When the system is further developed it could become a threat to the other two systems and make them superfluous.

The practical chapter demonstrates the comfortable way in which applications can be developed on the Android platform. Also in this part the author comes to the conclusion that Android has advantages in comparison with Symbian OS. The good information organization, the very useful examples, the well organized API and the innovative life-cycle of applications made it very easy to start developing on that system.

As an overall conclusion it can be said, that the Android mobile operating system seems to be a good decision for an operating system on every cell phone. It meets all the requirements an mobile operating system has to meet in the future but practical experience in the real world is indispensable. Consequently one has to keep in mind that Android is not available on a cell phone at the moment which makes it difficult to give a final and realistic conclusion.

In detail the experience made throughout the whole elaboration shows the following advantages and disadvantages of the mobile operating system Android. These apply only in comparison with the operating systems Windows Mobile and Symbian OS which were integrated in this elaboration.

Section 5.1 : Main Advantages

- The mobile platform Android is an open platform.
- The installation of Android is possible on every cell phone.
- The installation of the whole environment to develop Android applications is possible on every operating system.
- Android requires a low footprint of 250 kbyte
- The application model/life-cycle is future oriented with the source code separation of view and logic.
- The emulator of the Android platform has a modern design and is easy to use.
- Application installation on the emulator/device is possible via Android Debug Bridge (adb) or via Eclipse (with ADT plugin)
- Google offers a very good documentation as well as many examples which cover the most basic and important techniques used to get in touch with Android and the application development on it.
- Android supports all established techniques and standards for media, communication and data transfer.
- Android offers a real database which is SQLite.
- Android has an integrated web browser which allows a PC like usage
- Android relies on open operating system Linux version 2.6

- Android uses the standardized and open programming language Java
- The register based virtual machine Dalvik which is optimized for low memory requirements
- Google itself

Section 5.2 : Main Disadvantages

- No experience with Android in the "real world".
- Not many efficient applications available.
- The market leader is not present in the "Open Handset Alliance".
- The openness regarding the source code could be a problem concerning security.

This elaboration will be published through the Eastern Michigan University and can be used as a reference and evaluation of the new mobile operating platform Android.

Section 5.3 : Future Prospects

In the present Master Thesis technical features of mobile operating systems were used to compare the Android platform with other mobile platforms like Symbian OS or Windows Mobile. For a future look not only actual techniques, standards and methods have to be considered. Also trends and user needs should be of interest.

Which is the most important technology that is used at the moment and which will influence the future of normal and wireless communication most? These questions have to be answered if one wants to know where the future of cell phones and operating systems will be.

Nokias CEO Olli-Pekka Kallasvuo made the following statement at the Nokia World 2006 Conference regarding to future of cell phones and therefore mobile operating systems:

> *The Internet has transformed the way we live our lives and communicate with each other, and we expect it to play a key role in the next phase of Nokia's growth.*
>
> – Olli-Pekka Kallasvuo

The message “Internet is the future” which is as true now as it was in the past will most likely be the biggest challenge for cell phones and mobile operating systems in the future. the complete internet will have to take place on the cell phone as it does on the personal computer today. More cell phones than personal computers circulate around the world which shows the enormous potential of the mobile market. A successful operating system, which Andorid wants to be, has to focus on such a trend. It has to meet all possibilities of the internet, even if it is a webbrowser, connectivity standards or special applications.

The important characteristic of being small and being able to go everywhere makes the cell phone interesting for more or less every industry. Based on an article in a German newspaper it should also be possible to use the cell phone as a wallet in the future. Contact-free payment with an implemented chip is another possibility. This example indicates again how important application development for mobile operating systems is. A successful mobile operating system needs to offer the actual and most important applications. The first step with an open platform, where everybody is able to develop applications, is done by Android.

The trend for cell phones and its mobile devices is clear: Integrate as many functioality in a device as possible. Radio and Television, Timer and Clock, Internet and communication, everything is possible with the cell phone and its operating system.

Chapter 6 : Bibliography

[AT2006] Andrew S. Tannenbaum

Can we make operating systems reliable and secure?

[BM2007] Ben Morisson

The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS

2007, John Wiley & Sons

[EM2007] Andy Rubin

Interview with Andy Rubin by CNET NEWS.com

http://www.zdnetasia.com/insight/communications/0,39044835,62034932,00.htm

[GO2008-1] Google

Security and Permissions in Android

http://code.google.com/android/devel/security.html

[GO2008-2]	Google What is Andorid? http://code.google.com/android/what-is-android.html
[KD2006]	Kai-Oliver Detken Mobiler Einsatz – hoffentlich sicher. Stärken und Schwächen von Palm OS & Co.
[MS2007]	Microsoft Windows Mobile Security Model
[MW2006]	Martin Williams InfoWorld More mobile Internet users than wired in Japan.
[NO2007]	Virpi Roto, Principal Scientist, Nokia Research Center Internet User Experience: Emotions and Values

[NP2008]	Nilay Patel
	iphone SDK comparison chart
	http://www.engadget.com/2008/03/06/iphone-sdk-comparison-chart
[OHA2008]	The Open Handset Alliance
	Official website of the Open Handset Alliance
[SY2002]	Symbian
	Symbian OS Security Guide
[SY2003-1]	Symbian
	Symbian White Paper : Why is a different operating system needed?
[SY2003-2]	Symbian
	Symbian White Paper : Symbian in the Enterprise
[SY2007]	Symbian
	Symbian OS Version 9.4 description

[WI2008-1]	Windows Official Windows webpage: What is Windows Mobile? http://www.microsoft.com/windowsmobile/articles/benefits.mspx
[WI2008-2]	Windows Windows Mobile Features http://msdn2.microsoft.com/en-us/library/bb158483.aspx
[VA2004]	Volker Claus, Andreas Schwill Duden. Informatik. Ein Fachlexikon für Studium und Praxis

Chapter 7 : Glossar

API	*Application Programming Interface* Source code interface provided to support requests made by computer programs.
Activity	A simple class of the Android API.
Bug	A Bug is an error in a computer program.
GSM	*Global System for Mobile communications* Communication standard for mobile phones
IDE	*Integrated Development Environment* A software application that provides facilities to computer programmers like Eclipse.
J2ME	*Java 2 Platform, Micro Edition* Specification of a subset of the Java platform which allows developers to use Java and J2ME wireless toolkit to create applications.

MIDlet	*Mobile Information Device toolkit* Java program for embedded devices.
OHA	Open Handset Alliance A business alliance whose goal is to develop open standards for mobile devices.
PDA	*Personal Digital Assistant* A PDA is a handheld comuter comparable to a cell phone.
RAM	*Random Access Memory* RAM is a type of computer data storage.
SDK	*Software Development Kit* A set of development tools that allows a software engineer to create applications for a certain software package.
UI	*User Interface* Is the interface by which people interact with a system for example a computer or software.

XML	*Extensible Markup Language* is a very simple and flexible text format derived from SGML

Chapter 8 : Appendix A - source code TeaTime application on Android

```
package com.google.android.teatime;

import android.app.ListActivity;
import android.content.Intent;
import android.database.Cursor;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.Menu.Item;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import com.google.android.teatime.TeaDB;

/**
 * TeasList class. Values will be initialized at the beginning and the
 *    method onCreate() automatically called. Creates and handles the  List
 *    screen which is the first screen of the application.
 *
 * @ author Benjamin
 */
public class TeaList extends ListActivity {

      private static final int ACTIVITY_CREATE = 0;
      private TeaDB myDBHelper;
      private Cursor myTeaCursor;
      private static final int ADD_ID = Menu.FIRST;
      private static final int DELETE_ID = Menu.FIRST + 1;

      @Override
      /**
       * Method which will be called if the Activity starts and creates
            the screen.
       */
      public void onCreate(Bundle icicle) {
            super.onCreate(icicle);
            setContentView(R.layout.tea_list);
            myDBHelper = new TeaDB(this);
            myDBHelper.open();
            fillData();
      }

      /**
       * Listener method which reacts when a item in the list is
            selected. The it forwards to the Timer screen.
       */
      protected void onListItemClick(ListView l, View v, int position,
                                                        long id) {
            super.onListItemClick(l, v, position, id);
            Intent i = new Intent(this, TeaTimer.class);
            Bundle bundle = new Bundle();
            bundle.putLong("ID", id);
            i.putExtras(bundle);
            startSubActivity(i, ACTIVITY_CREATE);
      }
      /**
       * Method which destroys the application
       * @param icicle
       */
      public void onDetroy(Bundle icicle) {
            myDBHelper.close();
      }
      /**
       * Method which returns all tea names stored in the database
       */
      private void fillData() {
            // Get all of the rows from the database and create the
            item list
            myTeaCursor = myDBHelper.callUpAllTeas();
            startManagingCursor(myTeaCursor);

            // Create an array to specify the fields we want to display
            in the list (only NAME)
            String[] from = new String[] { TeaDB.KEY_NAME };

            // and an array of the fields we want to bind those fields
            to (in this case just text1)
            int[] to = new int[] { android.R.id.text1 };

            // Now create a simple cursor adapter and set it to display
            ListAdapter tea = new SimpleCursorAdapter(this,
                        android.R.layout.simple_list_item_1,
                                          myTeaCursor, from, to);
            setListAdapter(tea);
      }

      /**
       * Creates the Menu (Add + Delete)
       */
      public boolean onCreateOptionsMenu(Menu menu) {
            super.onCreateOptionsMenu(menu);
            menu.add(0, ADD_ID, R.string.menu_add);
            menu.add(0, DELETE_ID, R.string.menu_delete);
            return true;
      }
      /**
       * Activity Handler for the menu buttons (Add + Delete)
       */
      public boolean onMenuItemSelected(int featureId, Item item) {
            switch (item.getId()) {
            case ADD_ID:
                  createTea();
                  return true;
            case DELETE_ID:
                  myDBHelper.deleteTea(this.getSelectedItemId());
                  fillData();
                  return true;
            }
            return super.onMenuItemSelected(featureId, item);
      }
      /**
       * Forward to the "add tea" activity
       */
      private void createTea() {
            Intent i = new Intent(this, TeaEdit.class);
            startSubActivity(i, ACTIVITY_CREATE);
      }
      /**
       * Delete the whole Database
       *
       * @param name the name of the Database
       */
      public void deleteDB(String name) {
            this.deleteDatabase(name);
      }
```

Listing 17: TeaTime application (Android) – TeaList.java

```
package com.google.android.teatime;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

/**
 * TeasEdit class. Values will be initialized at the beginning and the method
 * onCreate() automatically called. Creates a tea in the database and provides
 * the screen for this
 *
 * @author Benjamin
 */
public class TeaEdit extends Activity {

    private EditText myNameText;
    private EditText myBrewTimeMinutesText;
    private EditText myBrewTimeSecondsText;
    private TeaDB myDBHelper;

    @Override
    /**
       * Method which will be called if the Activity starts and creates the
       * screen. Includes the onClick() Method which is a Listener which reacts
```

```
        * if the add button will be selected.
        */
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.tea_add);

        myNameText = (EditText) findViewById(R.id.name);
        myBrewTimeMinutesText = (EditText) findViewById(R.id.brewTimeMinutes);
        myBrewTimeSecondsText = (EditText) findViewById(R.id.brewTimeSeconds);

        Button confirmButton = (Button) findViewById(R.id.confirm);

        myDBHelper = new TeaDB(this);
        myDBHelper.open();

        confirmButton.setOnClickListener(new View.OnClickListener() {

            public void onClick(View view) {

                myDBHelper.createTea(myNameText.getText().toString(),
                  Integer.valueOf(myBrewTimeMinutesText.getText().toString()),
                  Integer.valueOf(myBrewTimeSecondsText.getText().toString()));
                myDBHelper.close();
                setResult(RESULT_OK, null, null);
                finish();
            }

        });
    }
}
```

Listing 18: TeaTime application (Android) – TeaEdit.java

```
package com.google.android.teatime;

import java.io.IOException;
import java.util.Timer;
import java.util.TimerTask;
import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

/**
 * TeaTimer class. Values will be initialized at the beginning and the method
 * onCreate() automatically called. Creates and handles the Timer screen.
 *
 * @author Benjamin
 */
public class TeaTimer extends Activity {

      private TeaDB myDBHelper;
      Bundle bundle;
      Handler handler;
      TextView tv;
      Timer t;
      long sec = -1;
      long min = -1;
      long seconds = -1;
      long minutes = -1;

      @Override
      /**
       * Method which will be called if the Activity starts and creates the
            screen.
       */
      public void onCreate(Bundle icicle) {
            super.onCreate(icicle);
            setContentView(R.layout.tea_timer);

            Button startButton = (Button) findViewById(R.id.start);
            Button resetButton = (Button) findViewById(R.id.reset);
            Button stopButton = (Button) findViewById(R.id.stop);

            myDBHelper = new TeaDB(this);
            myDBHelper.open();
            bundle = getIntent().getExtras();
            minutes = myDBHelper.getMinutes(bundle.getLong("ID"));
            seconds = myDBHelper.getSeconds(bundle.getLong("ID"));
            min = minutes;
            sec = seconds;

            tv = (TextView) findViewById(R.id.brewTime);
            handler = new Handler();

            tv.setText(min + " m : " + sec + " s");

            startButton.setOnClickListener(new Button.OnClickListener() {
                  public void onClick(View view) {
                        if (min == 0 && sec == 0) {

                              t.cancel();

                        } else
                              runTimer();
                  }
            });
            resetButton.setOnClickListener(new Button.OnClickListener() {
                  public void onClick(View view) {
                        sec = seconds;
                        min = minutes;
                        t.cancel();
                        tv.setText(min + " m : " + sec + " s");
                }
            });
            stopButton.setOnClickListener(new Button.OnClickListener() {
                public void onClick(View view) {
                        t.cancel();
                }
            });
    }

    /**
     * The Countdown Timer function
     */
    private void runTimer() {
            t = new Timer();
            t.schedule(new TimerTask() {

                public void run() {
                        if (sec == 0) {
                            min -= 1;
                            sec = 60;
                        }
                        sec -= 1;
                        handler.post(new Runnable() {
                            public void run() {
                                tv.setText(min + " m : " + sec + " s");
                                if (sec == 0 && min > 0) {
                                    min -= 1;
                                    sec = 60;
                                } else if (sec == 0 && min == 0) {
                                    cancel();
                                    playSound();
                                }
                            }
                        });
                }
            }, 1000, 1000);
    }

    /**
     * Method which reads a mediafile and plays
     */
    private void playSound() {

            MediaPlayer mp = new MediaPlayer();
            try {
                mp.setDataSource("/system/media/audio/ringtones/Ready.mp3");
            } catch (IOException e) {
                Log.e("ioexception", "", e);
            }
            try {
                mp.prepare();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            mp.start();
    }
```

```
}
```

Listing 19: TeaTime application (Android) – TeaTimer.java

```
package com.google.android.teatime;

import java.io.FileNotFoundException;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;

/**
 * TeaDB class. Values will be initialized at the beginning and the constructor
 * automatically called. Provides all important methods to handle the database.
 *
 * @ author Benjamin
 */
public class TeaDB {
      public static final String KEY_NAME = "name";
      public static final String KEY_BREWTIME_MINUTES = "brewTimeMinutes";
      public static final String KEY_BREWTIME_SECONDS = "brewTimeSeconds";
      public static final String KEY_ROWID = "_id";
      // Database creation sql statement
      private static final String DATABASE_CREATE ="CREATE TABLE IF NOT EXISTS "
                  + "TeaTable"
                  + " (_id integer primary key autoincrement, name text not
                        null, brewTimeMinutes long not null, brewTimeSeconds
                        long not null);";

      private static final String DATABASE_NAME = "TeaTimeDatabase";
      private static final String DATABASE_TABLE = "TeaTable";
      private static final int DATABASE_VERSION = 2;

      private SQLiteDatabase myDb = null;
      private final Context mCtx;

      /**
       * Constructor - takes the context to allow the database to be
       * opened/created
       *
       * @param ctx the Context within which to work
       */
      public TeaDB(Context ctx) {
            this.mCtx = ctx;
      }

      /**
       * Open the teaTable database. If it cannot be opened, try to create a new
       * instance of the database. If it cannot be created, throw an exception
       * to signal the failure
       *
       * @return this (self reference, allowing this to be chained in an
       *         initialization call)
       * @throws SQLException if the database could be neither opened or created
       */
      public TeaDB open() throws SQLException {
            try {
                  myDb = mCtx.openDatabase(DATABASE_NAME, null);
            } catch (FileNotFoundException e) {

                  try {
                        myDb = mCtx.createDatabase(DATABASE_NAME,
                                                            DATABASE_VERSION, 0,
                              null);
                        myDb.execSQL(DATABASE_CREATE);
                  } catch (FileNotFoundException ex1) {
                        throw new SQLException("Could not create database");
                  }
            }
            return this;
      }
      public void close() {
            myDb.close();
      }

      /**
       * Create a new Tea using the name and brewTime provided. If the Tea is
       * successfully created return the new rowId for that Tea, otherwise
       * return a -1 to indicate failure.
       *
       * @param name the name of the Tea
       * @param brewTime the brewTime of the Tea
       * @return rowId or -1 if failed
       */
      public long createTea(String name, int brewTimeMinutes, int
                                                            brewTimeSeconds) {
            ContentValues initialValues = new ContentValues();
            initialValues.put(KEY_NAME, name);
            initialValues.put(KEY_BREWTIME_MINUTES, brewTimeMinutes);
            initialValues.put(KEY_BREWTIME_SECONDS, brewTimeSeconds);
            return myDb.insert(DATABASE_TABLE, null, initialValues);
      }

      /**
       * Delete the Tea with the given rowId
       *
       * @param rowId id of Tea to delete
       * @return true if deleted, false otherwise
       */
      public boolean deleteTea(long id) {
            return myDb.delete(DATABASE_TABLE, "_id" + "=" + id, null) > 0;
      }

      /**
```

```
 * Return a Cursor over the list of all Teas in the database
 *
 * @return Cursor over all Teas
 */
public Cursor callUpAllTeas() {
      return myDb.query(DATABASE_TABLE, new String[]{ KEY_ROWID, KEY_NAME,
                  KEY_BREWTIME_MINUTES, KEY_BREWTIME_SECONDS }, null,
                  null, null, null, null);
}

/**
 * Return a Cursor positioned at the Tea that matches the given rowId
 *
 * @param rowId id of Tea to retrieve
 * @return Cursor positioned to matching Tea, if found
 * @throws SQLException if Tea could not be found/retrieved
 */
public Cursor catchUpTea(long rowId) throws SQLException {
      Cursor result = myDb.query(true, DATABASE_TABLE,
                  new String[]{ KEY_ROWID, KEY_NAME, KEY_BREWTIME_MINUTES,
                              KEY_BREWTIME_SECONDS }, KEY_ROWID + "=" +
                              rowId, null, null, null, null);
      if ((result.count() == 0) || !result.first()) {
            throw new SQLException("No Tea matching ID: " + rowId);
      }
      return result;
}

/**
 * Update the Tea using the details provided. The Tea to be updated is
 * specified using the rowId, and it is altered to use the Name and
 * BrewTime values passed in
 *
 * @param rowId id of note to update
 * @param name value to set Tea name to
 * @param brewTime value to set Tea brewTime to
 * @return true if the Tea was successfully updated, false otherwise
 */
public boolean updateTea(long rowId, String name, long brewTimeMinutes,
            long brewTimeSeconds) {
      ContentValues args = new ContentValues();
      args.put(KEY_NAME, name);
      args.put(KEY_BREWTIME_MINUTES, brewTimeMinutes);
      args.put(KEY_BREWTIME_MINUTES, brewTimeSeconds);
      return myDb.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId,
                                                      null) > 0;
}

/**
 * Get the minutes of the tea.
 *
 * @param rowId id of tea to get the minutes of it
 * @return minutes of the tea sort in regard to the tea ID
 */
public long getMinutes(long rowId) {
      long minutes = 0;
      Cursor result = myDb.query(true, DATABASE_TABLE,
                        new String[]{ KEY_ROWID, KEY_NAME, KEY_BREWTIME_MINUTES,
                                KEY_BREWTIME_SECONDS }, KEY_ROWID + "=" +
                                rowId, null, null, null, null);
                if ((result.count() == 0) || !result.first()) {
                        throw new SQLException("No Tea matching ID: " + rowId);
                } else {
                        result.first();
                        minutes = result.getLong(2);
                }

                return minutes;
        }

        /**
         * Get the seconds of the tea.
         *
         * @param rowId id of tea to get the minutes of it
         * @return minutes of the tea sort in regard to the tea ID
         */
        public long getSeconds(long rowId) {
                long minutes = 0;
                Cursor result = myDb.query(true, DATABASE_TABLE,
                        new String[]{ KEY_ROWID, KEY_NAME, KEY_BREWTIME_MINUTES,
                                KEY_BREWTIME_SECONDS }, KEY_ROWID + "=" +
                                rowId, null, null, null, null);
                if ((result.count() == 0) || !result.first()) {
                        throw new SQLException("No Tea matching ID: " + rowId);
                } else {
                        result.first();
                        minutes = result.getLong(3);
                }
                return minutes;
        }

}
```

Listing 20: TeaTime application (Android) – TeaDB.java

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/background1"
        android:orientation="vertical" android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        xmlns:android="http://schemas.android.com/apk/res/android">

        <ListView android:id="@android:id/list"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content" android:layout_weight="1">
        </ListView>

</LinearLayout>
```

Listing 21: TeaTime application (Android) – tea_list.xml

```xml
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical" android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <LinearLayout android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">

            <TextView android:layout_width="100sp"
               android:layout_height="wrap_content" android:text="Tea Name:  "
               android:layout_marginLeft="5px" />
            <EditText android:id="@+id/name"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content" android:layout_weight="1" />
      </LinearLayout>
      <LinearLayout android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">

            <TextView android:layout_width="100sp"
                  android:layout_height="wrap_content" android:text="Brewing
                                                                Minutes: "
                  android:layout_marginLeft="5px" />
            <EditText android:id="@+id/brewTimeMinutes"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:layout_weight="1"
                                                                          />
      </LinearLayout>
      <LinearLayout android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">

            <TextView android:layout_width="100sp"
                  android:layout_height="wrap_content" android:text="Brewing
                                                                Seconds: "
                  android:layout_marginLeft="5px" />
            <EditText android:id="@+id/brewTimeSeconds"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:layout_weight="1"
                                                                          />
      </LinearLayout>
      <Button android:id="@+id/confirm" android:text="Confirm"
            android:layout_width="110sp" android:layout_height="wrap_content" />
</LinearLayout>
```

Listing 22: TeaTime application (Android) – tea_add.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical" android:layout_width="fill_parent"
      android:layout_height="fill_parent" android:gravity="center">
      <TextView android:id="@+id/brewTime"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:textSize="60sp" />


      <LinearLayout
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="horizontal" android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal|bottom">
            <Button android:id="@+id/start" android:layout_width="90sp"
                  android:layout_height="wrap_content" android:text="START"
                  android:textSize="15sp" android:textStyle="bold" />
            <Button android:id="@+id/reset" android:layout_width="90sp"
                  android:layout_height="wrap_content"
                  android:layout_marginLeft="10px"
                  android:text="RESET" android:textSize="15sp"
                  android:textStyle="bold" />
            <Button android:id="@+id/stop" android:layout_width="90sp"
                  android:layout_height="wrap_content"
                  android:layout_marginLeft="10px"
                  android:text="STOP" android:textSize="15sp"
                  android:textStyle="bold" />
      </LinearLayout>

</LinearLayout>
```

Listing 23: TeaTime application (Android) – tea_timer.xml

Chapter 9 : Appendix B - source code TeaTime application on Symbian OS

```
package com.symbian.os.teatime;

import java.io.IOException;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.rms.InvalidRecordIDException;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreFullException;
import javax.microedition.rms.RecordStoreNotFoundException;
import javax.microedition.rms.RecordStoreNotOpenException;



public class TeaList extends MIDlet implements CommandListener {

      Display display = null;
      List menu = null;
      TeaDB db = null;
      int teaNumber=0;
      RecordStore rs = null;


      // command
      static final Command exitCommand = new Command("EXIT", Command.CANCEL, 0);
      static final Command addCommand = new Command("Add Tea",Command.SCREEN,1);
      static final Command deleteCommand = new Command("Delete Tea",
                                                     Command.SCREEN, 2);
      String currentMenu = null;

      // constructor.
      public TeaList() throws RecordStoreFullException,
                        RecordStoreNotFoundException, RecordStoreException {
            rs = RecordStore.openRecordStore("TeaTableDatabase", true);
      }

      /**
       * Start the MIDlet by creating a list of items and associating the exit
       * command with it.
       */
```

```
public void startApp() throws MIDletStateChangeException{
      db = new TeaDB();
      try {
            teaNumber = rs.getNumRecords();
            String[] teaList = new String[teaNumber];
            teaList = db.callUpAllTeas(teaNumber, rs);
            display = Display.getDisplay(this);
            menu = new List("Menu Items", Choice.IMPLICIT);

            for (int i = 0; i < teaNumber; i++) {
                  menu.append(teaList[i], null);
            }

            menu.addCommand(exitCommand);
            menu.addCommand(addCommand);
            menu.addCommand(deleteCommand);
            menu.setCommandListener(this);

            display.setCurrent(menu);

      } catch (RecordStoreNotOpenException e) {
            e.printStackTrace();
      } catch (IOException e) {
            e.printStackTrace();
      } catch (NullPointerException e) {
            e.printStackTrace();
      } catch (RecordStoreException e) {
            e.printStackTrace();
      }
}

public void pauseApp() {
      display = null;
      menu = null;
}

public void destroyApp(boolean unconditional) {
      notifyDestroyed();
}

/**
 * a generic method that will be called when selected any of the items on
 * the list.
 */
public void callStopWatch() {
      System.out.println();
      display = Display.getDisplay(this);
  try {
      display.setCurrent(new TeaTimerCanvas(this,
            db.getMinutes(db.getRecordID(rs,
            this.menu.getString(this.menu.getSelectedIndex())), rs),
            db.getSeconds(db.getRecordID(rs,
            this.menu.getString(this.menu.getSelectedIndex())), rs)));
      } catch (InvalidRecordIDException e) {
            e.printStackTrace();
      } catch (RecordStoreException e) {
            e.printStackTrace();
```

```
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
		/**
		 * Handle events.
		 */
		public void commandAction(Command c, Displayable d){
			String label = c.getLabel();
			if (label.equals("Exit")) {
				//destroyApp(true);
			} else if (label.equals("Add Tea")) {
				display = Display.getDisplay(this);
				display.setCurrent(new TeaAdd(this, rs));
			} else if (label.equals("Delete Tea")){
				System.out.println("Button Delete Tea");
					try {
						db.deleteTea(db.getRecordID(rs,
					this.menu.getString(this.menu.getSelectedIndex())), rs);
					} catch (RecordStoreNotFoundException e) {
						e.printStackTrace();
					} catch (RecordStoreException e) {
						e.printStackTrace();
					} catch (IOException e) {
						e.printStackTrace();
					}
					try {
						startApp();
					} catch (MIDletStateChangeException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}
			} else {
				callStopWatch();
			}
		}

		public void deleteRS() throws RecordStoreNotFoundException,
										RecordStoreException{
			RecordStore.deleteRecordStore("TeaTableDatabase");
		}
}
```

Listing 24: TeaTime application (Symbian) – TeaList.java

```
package com.symbian.os.teatime;
```

```
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.rms.RecordStore;


public final class TeaAdd extends Canvas implements CommandListener {

      Form teaForm = new Form ("Add a Tea");
      TextField name = new TextField ("Tea Name", "", 20, TextField.ANY);
      TextField minutes = new TextField ("Minutes", "", 1, TextField.NUMERIC);
      TextField seconds = new TextField ("Seconds", "", 2, TextField.NUMERIC);
      public TeaList localTest;
      public Display display;
      private TeaDB db;
      RecordStore localRS = null;

      public static final Command exitCommand = new Command("Back",
                                                     Command.BACK, 0);
      public static final Command addCommand = new Command("Add",
                                                     Command.SCREEN, 2);


      public TeaAdd(TeaList test, RecordStore rs) {
            localRS = rs;
            localTest = test;
            display = Display.getDisplay(test);
            teaForm.addCommand(exitCommand);
            teaForm.addCommand(addCommand);
            teaForm.setCommandListener(this);
      }

      protected void paint(Graphics g) {
            teaForm.append(name);
            teaForm.append(minutes);
            teaForm.append(seconds);
            display.setCurrent(teaForm);
      }

      public void commandAction(Command command, Displayable displayable) {
            String label = command.getLabel();
            if (label.equals("Back")) {
                  try {
                        localTest.startApp();
                  } catch (MIDletStateChangeException e) {
                        e.printStackTrace();
                  }
            } else if (label.equals("Add")) {
                  db = new TeaDB();
                  db.createTea(name.getString(),
                  Integer.valueOf(minutes.getString()).intValue(),
                  Integer.valueOf(seconds.getString()).intValue(), localRS);
                  try {
                        localTest.startApp();
                  } catch (MIDletStateChangeException e) {
                        e.printStackTrace();
                  }
            }
      }
}
```

Listing 25: TeaTime application (Symbian) – TeaAdd.java

```
package com.symbian.os.teatime;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDletStateChangeException;


public final class TeaTimerCanvas extends Canvas implements Runnable,
CommandListener {

      public Display display;
      //public TeaTimer localTimer;
      public TeaList localTimer;
      public int canvasWidth;
      public int canvasHeight;

      public int minutes = 0;
      public int seconds = 0;
      public int saveMinutes = 0;
      public int saveSeconds = 0;
      boolean test = false;

      public long a_long_fld;
      public long b_long_fld;
      public boolean stopWatchStatus;
      public static final Command backCommand = new Command("Back",
                                                   Command.STOP, 0);
      public static final Command startCommand = new Command("Start",
                                                   Command.SCREEN, 1);
      public static final Command stopCommand = new Command("Stop",
                                                   Command.SCREEN, 1);
      public static final Command resetCommand = new Command ("Reset",
                                                   Command.SCREEN, 1);

      public TeaTimerCanvas(TeaList timer, int min, int sec){
            stopWatchStatus = false;
            localTimer = timer;
            display = Display.getDisplay(timer);
            saveMinutes = min;
            saveSeconds = sec;
            // return the Width and Height of the Canvas,respectively
            canvasWidth = getWidth();
            canvasHeight = getHeight();
```

```
		addCommand(backCommand);
		addCommand(startCommand);
		addCommand(stopCommand);
		addCommand(resetCommand);
		setCommandListener(this);
		minutes = saveMinutes;
		seconds = saveSeconds;
		b_long_fld = System.currentTimeMillis();
	}

	public final void paint(Graphics g) {
		// String stopWatchString =  Integer.toString(minutes) + ":" +
		// Integer.toString(seconds+saveSeconds);
		String stopWatchString =  Integer.toString(minutes) + ":" +
								Integer.toString(seconds);
		g.setColor(255, 250, 250);
		g.fillRect(0, 0, canvasWidth, canvasHeight);
		g.setColor(0, 0, 0);
		g.drawString(stopWatchString, canvasWidth/2, canvasHeight/3, 17);
		display.callSerially(this);
	}


	long prevMin = System.currentTimeMillis();
	long prevSec = System.currentTimeMillis();
	public final void run() {
		if (stopWatchStatus == true){
			if( seconds < 0 )
			{
				if( minutes >= 0 ){
					seconds = 59;
					minutes--;
					prevMin = System.currentTimeMillis();
				}

			}
			else if( System.currentTimeMillis() - prevSec > 1000 )
			{
				seconds--;
				prevSec = System.currentTimeMillis();
			}
			repaint();
		}
	}



	public final void commandAction(Command command, Displayable displayable)
	{
		if (command == startCommand) {
		} else if (command == stopCommand){
				stopWatchStatus = false;
		} else if (command == backCommand){
			try {
				localTimer.startApp();
			} catch (MIDletStateChangeException e) {
				e.printStackTrace();
```

```
                    }
            } else if (command == resetCommand){
                    stopWatchStatus = false;
                    minutes = saveMinutes;
                    seconds = saveSeconds;
            }
        }
}
```

Listing 26: TeaTime application (Symbian) – TeaTimerCanvas.java

```
package com.symbian.os.teatime;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;

import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.rms.InvalidRecordIDException;
import javax.microedition.rms.RecordComparator;
import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordFilter;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreFullException;
import javax.microedition.rms.RecordStoreNotOpenException;

public class TeaDB {

        //RecordStore db = null;
        //RecordStore tempDB = null;


        /**
       * Constructor - Open the teaTable database. If it cannot be opened, try to
       * create a new instance of
       * the database. If it cannot be created, throw an exception to signal the
       * failure
       * @return this (self reference, allowing this to be chained in an
       * initialization call)
       * @throws SQLException if the database could be neither opened or created
       */
        public TeaDB() {
        }


        /**
```

```
* Create a new Tea using the name and brewTime provided. If the Tea is
* successfully created
* return the new rowId for that Tea, otherwise return a -1 to indicate
* failure.
* @param name the name of the Tea
* @param brewTime the brewTime of the Tea
* @return rowId or -1 if failed
*/
 public RecordStore createTea(String teaName, int brewMinutes, int
       brewSeconds, RecordStore rs){
       ByteArrayOutputStream baos = new ByteArrayOutputStream();
       DataOutputStream outputStream = new DataOutputStream(baos);
       try {
             outputStream.writeUTF(teaName);
             outputStream.writeInt(brewMinutes);
             outputStream.writeInt(brewSeconds);
       } catch (IOException e) {
             e.printStackTrace();
       }
       byte[] b = baos.toByteArray();
       try {
             rs.addRecord(b, 0, b.length);
       } catch (RecordStoreNotOpenException e) {
             e.printStackTrace();
       } catch (RecordStoreFullException e) {
             e.printStackTrace();
       } catch (RecordStoreException e) {
             e.printStackTrace();
       }
             System.out.println("See ya!");
 return rs;
 }

 /**
* Delete the Tea with the given rowId
* @param rowId id of Tea to delete
* @return true if deleted, false otherwise
*/
 public void deleteTea(int recordId, RecordStore rs){
       try {
             rs.deleteRecord(recordId);
             rs.enumerateRecords((RecordFilter)null, (RecordComparator)
                                                     null, true);
       } catch (RecordStoreNotOpenException e) {
             e.printStackTrace();
       } catch (InvalidRecordIDException e) {
             e.printStackTrace();
       } catch (RecordStoreException e) {
             e.printStackTrace();
       }
 }

 /**
* Return a Cursor over the list of all Teas in the database
* @return Cursor over all Teas
  * @throws RecordStoreException
  * @throws RecordStoreNotOpenException
  * @throws RecordStoreException
  * @throws RecordStoreNotOpenException
  * @throws IOException
*/
 public String[] callUpAllTeas(int teaNumber, RecordStore rs) throws
       RecordStoreNotOpenException, RecordStoreException, IOException,
       NullPointerException{

       String[] teaArray = new String[rs.enumerateRecords(null, null,
                                                   true).numRecords()];
       RecordEnumeration re = rs.enumerateRecords((RecordFilter)null,
                                        (RecordComparator) null, true);
       int i = 0;
       while (re.hasNextElement()){
             byte[] nextRec = re.nextRecord();
             ByteArrayInputStream bais = new ByteArrayInputStream(nextRec);
             DataInputStream dis = new DataInputStream(bais);
             teaArray[i]= dis.readUTF();
             i++;
       }
       return teaArray;
 }


 /**
* Return a Cursor positioned at the Tea that matches the given rowId
* @param rowId id of Tea to retrieve
* @return Cursor positioned to matching Tea, if found
  * @throws RecordStoreNotOpenException
  * @throws IOException
* @throws SQLException if Tea could not be found/retrieved
*/
 public String catchUpTea(int id, RecordStore rs) throws
                               RecordStoreNotOpenException, IOException{
       byte[] b = null;
       try {
             b = rs.getRecord(id);
       } catch (InvalidRecordIDException e) {
             e.printStackTrace();
       } catch (RecordStoreException e) {
             e.printStackTrace();
       }
       ByteArrayInputStream bais = new ByteArrayInputStream(b);
       DataInputStream dis = new DataInputStream(bais);
       String name = dis.readUTF();
       return name;
 }

 /**
* Get the minutes of the tea.
* @param rowId id of tea to get the minutes of it
* @return minutes of the tea sort in regard to the tea ID
  * @throws RecordStoreNotOpenException, IOException
*/
 public int getMinutes(int id, RecordStore rs){
       String name = null;
       int minutes = 0;
```

```
        int seconds = 0;
        byte[] b = null;
        try {
              b = rs.getRecord(id);
        } catch (InvalidRecordIDException e) {
              e.printStackTrace();
        } catch (RecordStoreException e) {
              e.printStackTrace();
        }
        ByteArrayInputStream bais = new ByteArrayInputStream(b);
        DataInputStream dis = new DataInputStream(bais);
        try {
              name = dis.readUTF();
              minutes = dis.readInt();
              seconds = dis.readInt();
        } catch (IOException e) {
              e.printStackTrace();
        }
        return minutes;
  }

  /**
 * Get the seconds of the tea.
 * @param rowId id of tea to get the minutes of it
 * @return minutes of the tea sort in regard to the tea ID
 */
  public int getSeconds(int id, RecordStore rs){
        String name = null;
        int minutes = 0;
        int seconds = 0;
        byte[] b = null;
        try {
              b = rs.getRecord(id);
        } catch (InvalidRecordIDException e) {
              e.printStackTrace();
        } catch (RecordStoreException e) {
              e.printStackTrace();
        }
        ByteArrayInputStream bais = new ByteArrayInputStream(b);
        DataInputStream dis = new DataInputStream(bais);
        try {
              name = dis.readUTF();
              minutes = dis.readInt();
              seconds = dis.readInt();
        } catch (IOException e) {
              e.printStackTrace();
        }
        return seconds;
  }

  protected void destroyApp(RecordStore rs) throws
                                          MIDletStateChangeException {
        try {
              rs.closeRecordStore();
        } catch (RecordStoreNotOpenException e) {
              e.printStackTrace();
        } catch (RecordStoreException e) {
                e.printStackTrace();
            }
      }


      public int getRecordID(RecordStore rs, String name1) throws
                  InvalidRecordIDException, RecordStoreException, IOException{
            int recordID = 1;
            boolean found = false;
            RecordEnumeration re = rs.enumerateRecords((RecordFilter)null,
                                                (RecordComparator) null, false);
            re.keepUpdated(true);

            while (re.hasNextElement() && found == false){
                  byte[] nextRec = re.nextRecord();
                  ByteArrayInputStream bais = new ByteArrayInputStream(nextRec);
                  DataInputStream dis = new DataInputStream(bais);
                  String name2 = dis.readUTF();
                  if (name1.equalsIgnoreCase(name2)){
                        found = true;
                        if (re.hasNextElement() == true){
                              re.nextRecord();
                              recordID = re.previousRecordId();
                        } else if (re.hasPreviousElement() == true){
                              re.previousRecord();
                              recordID = re.nextRecordId();
                        }
                  }
            }
            return recordID;
      }


      protected void pauseApp() {
            // not neccessary

      }

      protected void startApp() throws MIDletStateChangeException {
            // notneccessary

      }

      public void deleteRS(String name) throws RecordStoreNotOpenException,
                                                      RecordStoreException{
            RecordStore.deleteRecordStore(name);
      }
}
```

Listing 27: TeaTime application (Symbian) – TeaDB.java