|  |
| --- |
| Omáda Programming |
| Team 15 - Java Card  Version 1.5  Brasoveanu Andrei-Alexandru  *Delvin Varghese*  *Dominic Lee* |
|  |

**Version History**

|  |  |  |
| --- | --- | --- |
| Date | Name | Reason for Changes |
| 15/04/2013 | Delvin Varghese | First version of document. |
| 02/05/2013 | Dominic Lee | Differences between Javacard and Java |
| 05/05/2013 | Delvin Varghese  Dominic Lee | Definition, References etc |

Applet

# Definition

This document explains the client-side architecture and implementation of the Fortune software. Its scope is limited to the Java Card language specification, the industry specifications, code review of the client implementation and key terminology needed to understand the concepts of this applet.

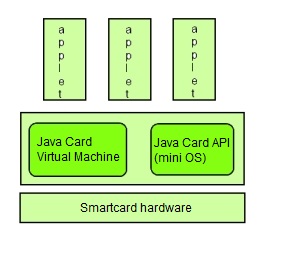
**Applet:** The technical definition[1] of the term is:

*“an applet is any small application that performs one specific task that runs within the scope of a larger program”*

In this document, **applet** refers to the collective piece of code running on the SIM card of a GSM mobile phone. This represents the client software in this client-server relationship.

The project consists of this applet, a server and a communication protocol which includes elements from the mobile telephone network which enable the SIM to send and receive messages to/from the server.

## Applet Architecture



Coding

# Overview

The client side has been written entirely in Javacard.

According to the Oracle[2] website:

*Java Card technology provides a secure environment for applications that run on smart cards and other devices with very limited memory and processing capabilities.*

Java Card allows Java-based applications to be run on smart cards. Also it has a very important feature that applets written for one smart card can be scaled to another very easily. Multiple applications can be deployed on a single card, and new ones can be added to it even after it has been issued to the end user. Due to this it is used extensively in GSM\* mobile phones.

\* *GSM is a standard used to describe a set of protocols used by 2G mobile networks. It is the most widely used in the world. Other standards used today include 3G and more recently, 4G.*

These are some of the reasons why our client uses JavaCard for their development.

# More Info

Java Card defines a Java Card Runtime Environment (**JCRE**) and provides classes and methods to help developers create applets. Applets run within the JCRE. The JCRE and APIs are modelled after the smart card specification ISO 7816.

Java Card aims to emulate the portability features of Java (Oracle famously employed the slogan “Write Once, run anywhere” to describe Java).

It does so by using the Java Card Virtual Machine and the JCRE which enables it to run abstractly regardless of differences between different cards. It also supports cryptography, firewall and data encapsulation for security requirements.

# Differences between Java Card and Java

Java is a class based object orientated programming language its major quality is that once the code has been interpreted into byte code it can be ran on any Java virtual machine, regardless of architecture. Java-Card is the smallest branch of Java designed for making applets specifically aimed at embedded devices. All the language constructs that exist in java are in Java-Card and behave in exactly the same way.

Even though this is the case Java-Card does not support types *char, double, float, long* and arrays of more than one dimension, and also the *int* type is not supported by most smart cards themselves. Because of these limitations the ways in which problems are solved subtly differ.

Java-Card has been described as a

superset of a subset of java

This is because as mentioned above, may features typical of Java are removed. Due to hardware constraints, other powerful features like ability to create threads, Java's Garbage Collector etc are also not available.

On the other hand, it has some extras like

1. communication via APDU's or RMI.
2. persistent & transient data in EEPROM & RAM
3. Transaction mechanism.

The Java-Card applet is limited in the above way to save on size as this is a valuable resource in a smart card. A smart card may hold many applets and as the *Java Security Manager* class is not supported in Java Card, it has its own security measures. One of which is the *Java Card firewall* which separates the applets form each other, it also includes an option to allow an applet to make a variable available to other applets.

# Coding Explanations

*This section provides details of the methods used in the Application code.*

public FortuneApplet(byte[] bArray, short bOffset, short parametersLength)

This is the applet constructor. It sets up the applet for use. First, it adds support to the Applet for formatted SMS-PP messages, since this is how the applet will send and receive fortune cookies. It also sets up the buffers to be used by the OS since the memory space available for execution is limited.

This also creates a byte array containing an identifier so messages sent from this applet can be identified on the server.

Also, it creates the menu entry on the SIM menu list, so a menu option called “Fortune” is available to the user to press.

public static void install(byte[] bArray, short bOffset, byte bLength)

This method is called by the Java Card Virtual Machine (JCVM) on the SIM OS initially on applet run.

public void process(APDU apdu)

Another method called by the JCVM. This processes an incoming Application Data Unit (APDU).

private void processBuffer(byte[] apduBuffer)

Used for the simulated testing environment on PC only. This method will not be used on actual Applet when deployed on device.

private void displayFortuneMenu()

For additional functionality which can be done once Fortune message is displayed. This can be added later by the team or by the client.

public void processToolkit(byte event)

This is the logic called when an event is fired, for example when a message is received or a menu selection is made.

private void processIncomingEnvelope()

Called by the logic in processToolkit(), this processes an incoming message into an appropriate format so it can be displayed to user.

private byte sendSms(byte[] msisdn, byte[] payload, byte dcs)

The method which enables the applet to send a SMS. The *msisdn* is the unique identifier of where to send the message. *Payload* represents the message to send. *Dcs* is the default character set.

private short pack(byte[] src, short offsetSrc, byte[] dst, short offsetDst, short length)

Converts an 8 bit GSM alphabet message sent from server into a 7 bit ASCII format. The converted message is displayed by the device. *Src* is the initial message source and *dst* is the packed message target.

References

1. <http://www.answers.com/topic/applet>
2. <http://www.oracle.com/technetwork/java/javame/javacard/overview/getstarted/index.html>