A

Seminar report

On

Java Card

Submitted in partial fulfillment of the requirement for the award of degree
Of MCA

SUBMITTED TO:

www.studymafia.org

SUBMITTED BY:

www.studymafia.org

Preface

I have made this report file on the topic **Java Card,** I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to ……..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

CONTENTS

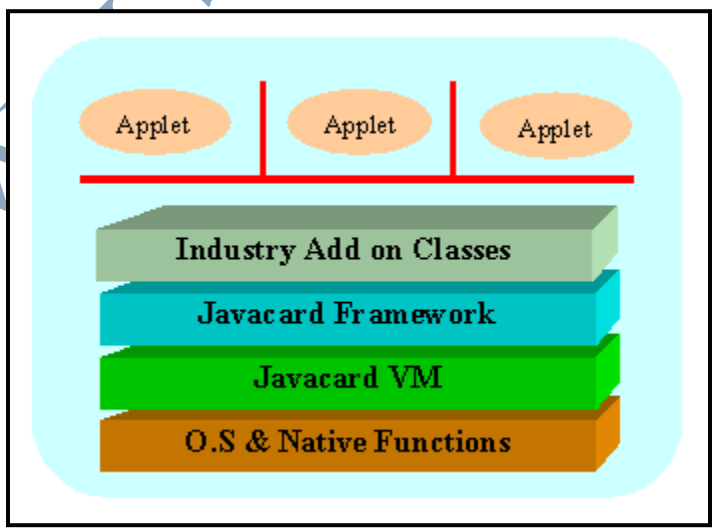
INTRODUCTION

Java Card is a smart card that is capable of running programs written in Java. For this a new Java platform, Sun's JavaSoft division has made available the Java Card 2.0 API specification, and several licensees are now implementing this API on smart cards. In order to program Java Cards that are 2.0-compliant, developers need to understand what a Java Card is architecturally, what its core classes are, and how to develop applications for the card. This article gets inside a Java Card, providing you, the developer, with technical guidance on the system architecture, application programming interface, and runtime environment of the Java platform in a smart card.

WHAT IS A JAVA CARD?

A Java Card is a smart card that is capable of running Java programs. The Java Card 2.0 specification contains detailed information for building the Java Card virtual machine and application programming interface (API) in smart cards. The minimum system requirement is 16 kilobytes of read-only memory (ROM), 8 kilobytes of EEPROM, and 256 bytes of random access memory (RAM). A Java Card is shown below.

System architecture on the Java Card



As shown in the figure, the Java Card VM is built on top of a specific integrated circuit (IC) and native operating system implementation. The JVM layer hides the manufacturer's proprietary technology with a common language and system interface. The Java Card framework defines a set of Application Programming Interface (API) classes for developing Java Card applications and for providing system services to those applications. A specific industry or business can supply add-on libraries to provide a service or to refine the security and system model. Java Card applications are called *applets*. Multiple applets can reside on one card. Each applet is identified uniquely by its *AID* (application identifier), as defined in ISO 7816, part 5.

An important point to keep in mind is what smart cards *are not*: They are not personal computers. They have limited memory resources and computing power. Users should not think of Java Card 2.0 as simply a stripped-down version of the JDK.

The lifetime of a Java Card

The Java Card lifetime starts when the native OS, Java Card VM, API classes libraries and optionally, applets are burned into ROM. This process of writing the permanent components into the non-mutable memory of a chip for carrying out incoming commands is called masking.

Before it lands in your wallet, a Java Card needs to go through initialization and personalization. Initialization refers to loading general data into a card's non-volatile memory. This data is identical across a large number of cards and is not specific to an individual; an example might be the issuer or manufacture's name.

The next step, personalization, involves assigning a card to a person. It can occur through physical personalization or through electronic personalization. Physical personalization refers to embossing or laser engraving your name and card

number on the plastic surface of a card. Electronic personalization refers to loading personal data into a card's non-volatile memory, for example, your personal key, name, and pin number. Initialization and Personalization vary from vendor to vendor and issuer to issuer. In both, EEPROM (a type of non-volatile memory) is often used for storing data.

At this point, the Java Card is ready for use. You can get a Java Card from an issuer or buy it from a retailer. Cards sold by a retailer are general-purpose, in which case personalization is often omitted. Now you can insert your Java Card into a reader and send APDU commands to the applets residing on the card or download more applets or data onto the card. A Java Card remains active until it is expired or blocked due to an unrecoverable error.

Lifetime of a Java Card virtual machine

Unlike the Java virtual machine (JVM) in a PC or workstation, the Java Card virtual machine runs forever. Most of the information stored on the card must be preserved even when the power is removed -- that is, when the card is removed from the reader. The Java Card VM creates objects in EEPROM to hold the persistent information. The execution lifetime of the Java Card VM is the lifetime of the card. When the power is not provided, the VM runs in an infinite clock cycle.

The lifetime of Java Card applets and objects

An applet's life starts when it is properly installed and registered with the system's registry table and ends when it is removed from the table. The space of a removed applet may or may not be reused, however, depending on whether garbage collection is implemented on the card. An applet on a card is in an inactive stage until it is explicitly selected by the terminal.

Objects are created in the persistent memory (for example, EEPROM). They could be lost or garbage-collected if other persistent objects do not reference them. However, it's a thousand times slower to write to EEPROM than to RAM. Some objects are accessed frequently, and the contents of their fields need not be persistent. The Java Card supports transient (temporary) objects in RAM. Once an object has been declared as transient, its contents can not be moved back to the persistent memory.

Java Card 2.0 language subset

Java Card programs are, of course, written in Java. They are compiled using common Java compilers. Due to limited memory resources and computing power, not all the language features defined in the Java Language Specification are supported on the Java Card. Specifically, the Java Card does not support:

- Dynamic class loading
- Security manager
- Threads and synchronization
- Object cloning
- Finalization
- Large primitive data types (float, double, long, and char)

It's no surprise that keywords that support those features are also omitted from the language. VM implementers may decide to support 32-bit integer type or native methods for post-issuance applets if they are working on a more advanced smart card with more memory. Post-issuance applets are those applets that are installed on a Java Card after the card is issued to a card holder.

The Java Card 2.0 framework

Smart cards have been in the market for 20 years, and most of them are generally compatible with ISO 7816 Parts 1-7 and/or EMV. We've already looked at ISO 7816. What's EMV? The EMV standard, defined by Europay, MasterCard, and Visa, is based on the ISO 7816 series of standards with additional proprietary features to meet the specific needs of the financial industry. The Java Card Framework is designed to easily support smart card systems and applications. It hides the details of the smart card infrastructure and provides Java Card application developers with a relatively easy and straightforward programming interface.

Java Card security

Java applets are subject to Java security restrictions, however, the security model of Java Card systems differs from standard Java in many ways. The Security Manager class is not supported on Java Card. Language security policies are implemented by the virtual machine. Java applets create objects that store and manipulate data. An object is owned by the applet that creates it. Even though an applet may have the reference to an object, it cannot invoke the object's methods, unless it owns the object or the object is explicitly shared. An applet can share any of its objects with a particular applet or with all applets.

An applet is an independent entity within a Java Card. Its selection, execution, and functionality are not affected by other applets residing on the same card.

How things work together inside a Java Card

Inside a Java Card, JCRE (Java Card Runtime Environment) refers to the Java Card virtual machine and the classes in the Java Card Framework. Each applet within a Java Card is associated with unique AID assigned by JCRE. After an applet is correctly loaded into the card's persistent memory and linked with the Java Card

Framework and other libraries on the card, JCRE calls the applet's install method as the last step in the applet installation process. A public static method, `install`, must be implemented by an applet class to create an instance of the applet and register it with JCRE. Because memory is limited, it's good programming practice, at this point, to create and initialize the objects the applet will need during its lifetime.

An applet on the card remains inactive until it is explicitly selected. The terminal sends a "SELECT APDU" command to JCRE. JCRE suspends the currently selected applet and invokes the applet's `deselect` method to perform any necessary cleanup. JCRE then marks the applet whose AID is specified in the "SELECT APDU" command as the currently selected applet and calls the newly selected applet's `select` method. The select method prepares the applet to accept APDU commands. JCRE dispatches the subsequent APDU commands to the currently selected applet until it receives the next "SELECT APDU" command.

DEVELOPING A JAVA CARD APPLET

After you write a Java Card applet, you're ready to prepare it for execution in a Smart Card that implements the Java Card runtime environment. Preparing a Java Card applet for execution involves a number of steps, such as converting it to a runtime format and testing it in various simulated environments.

Using the Java Card Development Kit

Use the Java Card 2.1.2 Development Kit to develop a Java Card applet. You can use the Java Card 2.1.2 Development Kit to develop an applet for masking. Masking means embedding the applet into the read-only memory of a smart card when the card is manufactured. Alternatively, you can use the Java Card 2.1.2 Development Kit to develop an applet for installation onto a smart card after the card is manufactured.

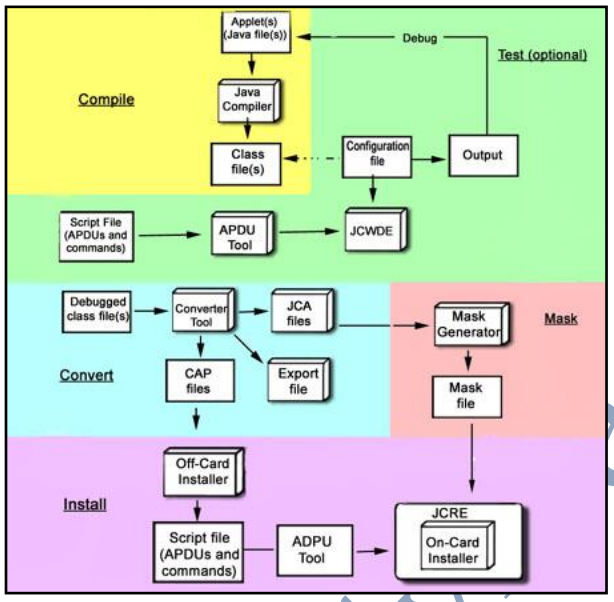
The Java Card 2.1.2 Development Kit provides components and tools that you need to develop applets for masking or installation. This includes:

- Java Card Framework classes that are essential for developing Java Card applets.
- A Java Card Workstation Development Environment (JCWDE) that simulates the Java Card runtime environment on a Java[tm] virtual machine.
- An APDUTool utility that sends command APDUs to the JCWDE or to a Java Card runtime environment. Command APDUs are the way operational requests are made to a smart card.
- A Converter tool that converts a Java Card applet into a format required for masking or for installation.
- Off-card verification tools that check the integrity of files produced by the Converter.

- A mask generator that generates a mask file for incorporation into a mask in a Java Card runtime environment.
- An off-card installer for installing a Java Card applet onto a smart card.
- Using these classes and tools, you develop a Java Card applet on your workstation or PC. Specifically, you:
- Compile the applet.
- Optionally, test the applet in the JCWDE, and debug the applet.
- Convert the applet. If you develop an applet that will be masked, you convert the applet class and all the classes in its package to a Java Card Assembly (JCA) file. If you develop an applet for installation, you convert the applet and all the classes in its package to a Converted Applet (CAP) file, and possibly an export file. An export file is used to convert another package if that package imports classes from this package.

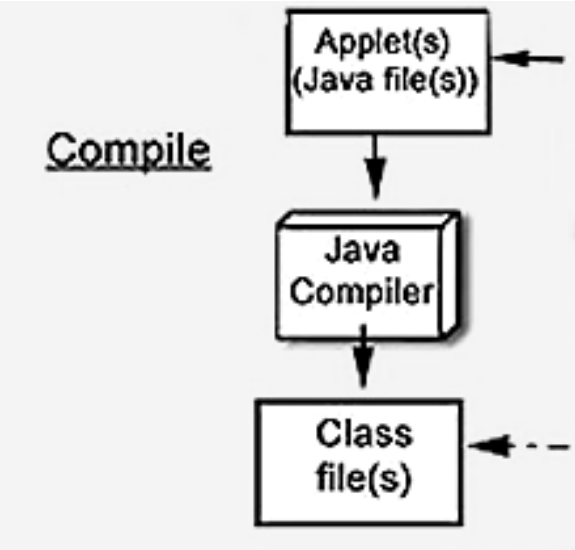
The next step depends on whether you develop an applet for masking or for installation. For masking, you run the mask generator to produce a mask file. For installation, you run the off-card installer; this produces a script file that contains command APDUs -- you then use the file as input to the APDUTool. The APDUTool works in conjunction with the installer on the smart card to download the CAP file and instantiate the Java Card applet in the CAP file.

The steps in the development process are illustrated in the following figure.



Compiling a Java Card Applet

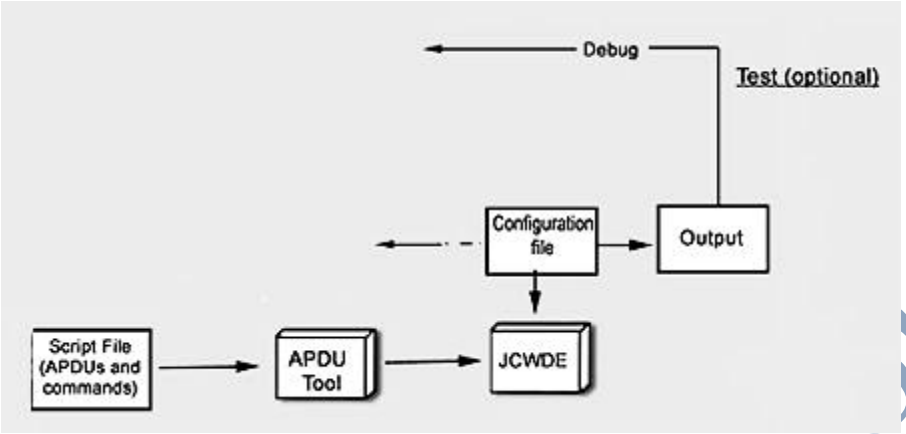
You write Java Card applets in the Java programming language. However because applets are designed to run in the very small memory space of a smart card, they're coded using an appropriate subset of the Java programming language. As you do for a Java application or applet, you compile Java Card applets on your workstation or PC. You can use any Java compiler that supports Java 2 Platform, Standard Edition version 1.2.2, 1.3 (or above), such as the javac compiler in Java 2 SDK version 1.3. Remember to include api21.jar in your class path before you compile.



The javac compiler is invoked with the -g option(javac –g <filename.java>). This tells the compiler to generate debugging information. You need to specify this option in preparation for running the Converter tool. That's because the Converter tool requires information about local variable types within the applet, information that it gets from the LocalVariableTable attribute. The attribute is generated only if the -g option is specified when you compile the applet.

Testing a Java Card Applet in the JCWDE

This step is optional. You can wait to test your applet until a later step in the development process, for instance, when you convert it to a format for masking or for installation. However if you want to do an early test of your applet, you can test it in the JCWDE. This gives you a way of testing a Java Card applet on your workstation or PC, that is, in a Java virtual machine, without having to convert the applet.



To test a Java Card applet in the JCWDE, you:

- Start the JCWDE
- Run the APDUTool Utility
- Debug the Applet

Starting the JCWDE

The JCWDE, which runs on your workstation or PC, simulates the Java Card runtime environment on a Java virtual machine.

It allows you to run your applet as though it was masked in the read-only memory of a smart card. And importantly, it allows you to run the test in your workstation or PC, without having to convert the applet, generate a mask file, or install the applet. To start the JCWDE, issue the `jcdwe` command. (This runs a script file in the Solaris Operating Environment, and a batch file in the Windows NT platform.) The primary input to the command is a configuration file that identifies one or more applets.

The applets identified in the configuration file are masked into the JCWDE, as though the applets were stored in the read-only memory of the smart card runtime

environment. The applets are identified in the configuration file by their Application Identifier (AID). With the applets configured into its mask, the JCWDE is able to direct processing requests in the form of command APDUs to the appropriate applet for processing.

Running the APDUTool Utility

The APDUTool Utility submits command APDUs to a Java Card runtime environment, or to a simulated runtime environment such as the JCWDE. It's used in the process of installing a CAP file into the Java Card runtime environment of a smart card. But it also provides a convenient way for you to submit command APDUs to a Java Card applet masked into the JCWDE, as a way of testing the applet. The primary input to the utility is a script file that contains one or more command APDUs as well as some other commands unique to the APDUTool. When used with the JCWDE, the utility directs each command APDU to the JCWDE, which in turn, sends it to the appropriate applet for processing. Each applet responds with a response APDU. As output, the APDU utility displays the command APDUs and the associated response APDUs.

To start the APDUTool Utility, issue the `apdutool` command, and specify the script file as input. This runs a script file in the Solaris Operating Environment (a different script file than the one that contains the command APDUs), and a batch file in the Windows NT platform.

The Script File:

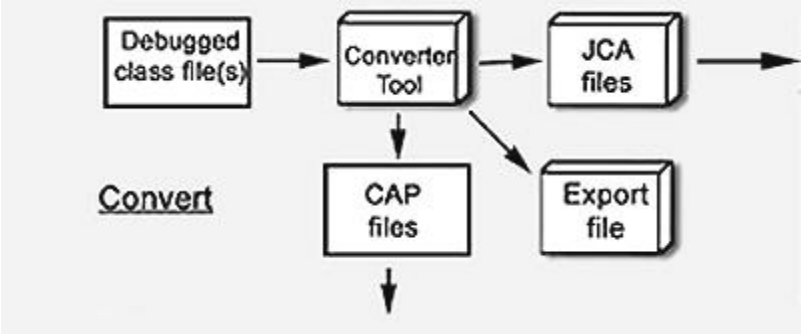
The script file contains the primary input data for the APDUTool utility. The file contains the command APDUs to be processed by the Java Card runtime environment (or JCWDE). It also contains commands that are unique to the APDUTool utility. In addition, the file can include comments.

Debugging a Java Card Applet

You can debug the applet on your workstation or PC just as you do for a Java application. More specifically, you can use the same debugging tools, such as the debugging facilities of an IDE, or the Java debugger tool (jdb) in the Java 2 SDK.

Converting a Java Card Applet

In Java Card technology, you don't directly incorporate a Java Card applet into a mask. Similarly, after a smart card is manufactured, you don't directly download a Java Card applet for installation onto a smart card. Instead, for masking, you convert an applet class and all the classes in its package to a JCA (Java Card Assembly) file. The JCA file and JCA files for any other packages to be included in the mask are then converted into a format compatible with the target runtime environment. It's this converted output for the target runtime environment that is incorporated into the mask.



For installation onto an already-manufactured smart card, you convert an applet class and all the classes in its package to a CAP (converted applet) file. You then download the CAP file to a smart card using the off-card installer in conjunction with the APDUTool utility and the on-card installer.

Both a JCA file and a CAP file are self-descriptive files, that is, they contain information about their contents. In other words, these files contain information about the converted package. In addition, a CAP file is in a compressed format that

is optimized for the limited amount of memory in a smart card. A JCA file is simply a text representation of the contents of a CAP file.

To convert a Java Card applet (whether it's for masking or installation), use the Converter tool that's in the Java Card 2.1.2 Development Kit. You do this by issuing the `converter` command. (This runs a script file in the Solaris Operating Environment, and a batch file in the Windows NT platform.) The tool operates on a Java package. You specify the name, AID, and version of the package; the tool then converts all the classes in the package. What this means is that you don't convert a Java Card applet individually, but instead you convert as a unit the applet and all the other applets in its package.

When you run the Converter tool you specify what output to produce: a CAP file, JCA file, or another file not yet mentioned called an export file. You can request any combination of these files. By default, the converter produces a CAP file and an export file.

What Happens During Conversion?

Of course one thing that happens is that one or more files such as a CAP, EXP, or JCA file is produced. But some other important things happen too. As part of the conversion process, the Converter tool preprocesses the classes. That is, it performs some of the tasks that a Java virtual machine normally does when it loads a Java class in a desktop environment. The idea behind the preprocessing is to keep the Java virtual machine on a smart card as small as possible. One of the preprocessing tasks the Converter tool performs is initializing static variables in the classes, another is resolving symbolic references. In addition, the Converter tool checks whether the Java classes in the package are properly formed, and whether the applets use only the subset of the Java programming language that is supported by the Java Card platform. The Converter tool then directs the result of its preprocessing checks to the standard output stream.

Verifying the Integrity of CAP and Export Files

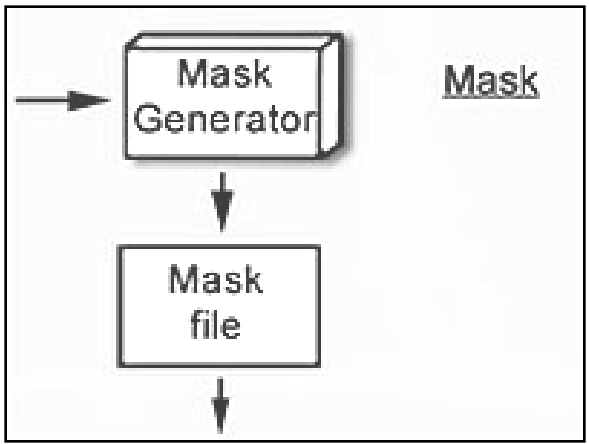
The Java Card 2.1.2 Development Kit provides a number of tools that you can use to verify the integrity of the CAP and export files. Verifying integrity means ensuring that the CAP and export files conform to the Java Card 2.1.1 specifications. For example, one aspect of verifying integrity is checking that the files do not attempt to compromise the integrity of the Java Card virtual machine implementation and other applets.

There are three integrity verification tools that you can use:

- VerifyCap: Use this tool to verify the integrity of a CAP file within the context of the export files it imports. The tool also verifies the integrity of the export file (if any) for this package that can be imported during the conversion of another package.
- VerifyExp: Use this tool to verify the integrity of a single export file.
- VerifyRev: Use this tool to verify the binary compatibility of two export files, for example, where each file is for a different version of a package.

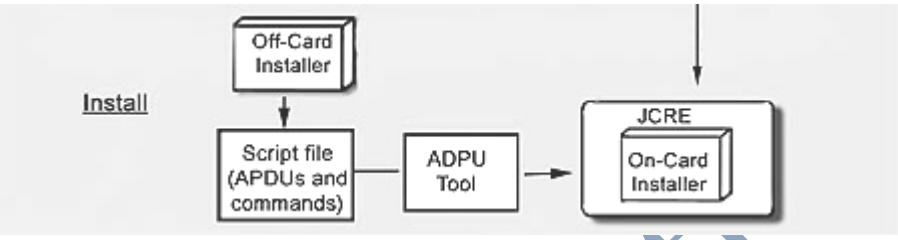
Generating a Mask File

Use the mask generator provided with the Java Card 2.1.2 Development Kit to generate a mask file for one or more Java Card applets. The mask file can then be incorporated into a mask for a specific Java Card runtime environment. You specify as input to the mask generator the JCA file for the package that contains the applets, as well as JCA files for any other packages to be included in the mask file, such as JCA files for any needed Java Card API packages.



Installing a CAP File

As mentioned earlier, you don't install a Java Card applet onto a smart card, instead you install its CAP file. The Java Card 2.1.2 Development Kit includes an off-card installer utility that prepares a CAP file for installation. The "off-card" designation differentiates the installer provided with the Java Card 2.1.2 Development Kit from the "on-card" installer resident in a smart card.



The off-card installer produces a script file that contains command APDUs that identify the beginning and end of the CAP file, its components, and component data. The script file is used as input to the APDUTool Utility. This is the same APDUTool utility and script file that are described in Running the APDUTool Utility. The APDTool utility works in conjunction with the on-card installer to download the CAP file, so you also need to add an APDU command to the script file to start the on-card installer. You can also add command APDUs to direct the on-card installer to create, that is, instantiate, the applets defined in the CAP file.

After you tailor the script file, you run the APDUTool utility, specifying the script file as input. The APDUTool starts the on-card installer, which downloads the CAP file. If requested in the script file, the on-card installer creates the applets that are defined in the CAP file, so that the applets are available in the Java Card runtime environment.

Running the Off-Card Installer

To run the off-card installer, issue the *scriptgen* command. (This runs a script file in the Solaris Operating Environment, and a batch file in the Windows NT

platform.) Specify the CAP file path as input to the command. By default, the off-card installer directs output to the standard output stream, however the command does provide a flag for specifying an output script file name.

CONCLUSION

Java Card can be used in all fields where the smart card is now being used. Java Card can be used as an ID card which contains personal information, as a medical card which stores medical information, as a credit/debit bank card, as an electronic purse etc. Multi-Application Java Cards, that is, more than one application in a single card is also available.

The Java Card adds a new platform to the world of Java. Widespread adoption and deployment of the Java Card will require marketing promotion, more applications and tools development, and time. At the same time, the number of Java Cards in existence could easily extend into the millions within the next few years. Which means you may soon be storing your personal information and downloading applications using a little card you carry around in your wallet or purse.

REFERENCES

- www.google.com
- www.wikipedia.com
- www.studymafia.org