Developing Object-Based Applications in Java

Rationale

- Java is an object-oriented language that enables you to create real-world applications. The code reusability feature of Java enables the software developers to upgrade the existing applications without re-rewriting the entire code of the application. The concept of working with files and I/O streams enables the software developers to store and retrieve the information from a flat or a text file. Packages enable the reusability of classes and methods across various applications.

Objectives

- In this session, you will learn to:
  - Identify characteristics of the Java programming language
  - Declare class members
  - Use arrays

Java Programming Language

- Java is an OOP language that was designed to meet the need for a platform-independent language.
- Java is used to create applications that can run on a single computer as well as a distributed network.
- Java is used to develop stand-alone and Internet-based applications.
- With the increasing use of the Internet, Java has become a widely used programming language.
- The Java software works everywhere, from the smallest devices, such as microwave ovens and remote controls to supercomputers.
- The Java programs work on any type of compatible device that supports Java.

Need for Java

- The primary motive behind developing Java was the need for a portable and platform-independent language that could be used to produce code that would run on a variety of systems.
- A few types of Java applications are:
  - Applications that use CUI
  - Applications that use GUI
  - Applets
  - Servlets
  - Packages
- Java is a platform-independent language that enables you to compile an application on one platform and execute it on any platform.

Evolution for Java

- In 1991, a team of software developers at Sun Microsystems, USA, was designing a language for consumer electronic devices.
- The development team headed by James Gosling wanted to design a portable language by which programs could run on computers with different platforms.
- The team considered C++ as the model language for designing the new language.
- The team deprecated various ambiguous features of C++ from this new language.
- Initially, this developed language was called Oak, but was later renamed to Java.

Evolution for Java (Contd.)

- The following table describes the evolution of Java.

Year	*Precedence*
1990	*Sun Microsystems developed software to manipulate electronic devices.*
1991	*A new language named Oak was introduced using the most popular object-oriented language C++.*
1993	*World Wide Web (WWW) appeared on the Internet that transformed the text-based Internet into graphical Internet.*
1994	*The Sun Microsystems team developed a Web browser called HotJava to locate and run applet programs on the Internet.*
1995	*Oak was renamed as Java.*
1996	*Java was established as an object-oriented programming language.*

Evolution for Java (Contd.)

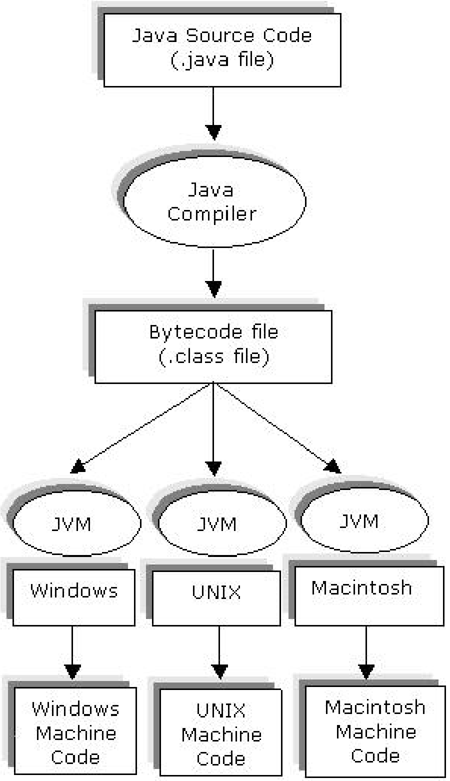
- Java exhibits the following characteristics:
  - Simple
  - Object-oriented
  - Compiled and interpreted
  - Portable
  - Distributed
  - Secure

Evolution for Java (Contd.)

- Simple:
  - A Java programmer does not need to know the internal functioning of Java, such as how memory is allocated to data.
- Object-oriented:
  - Java supports the object-oriented approach to develop programs.
- Compiled and interpreted:
  - The Java programs are first compiled and then interpreted. After the program is compiled, it is converted to a bytecode. The Java Virtual Machine (JVM) then interprets this bytecode into the computer code and runs it.
- Portable:
  - Refers to the ability of a program to run on any platform without changing the source code of a program.

Evolution for Java (Contd.)

- The following figure shows how the Java bytecode and the JVM together make Java programs portable on different platforms.



Evolution for Java (Contd.)

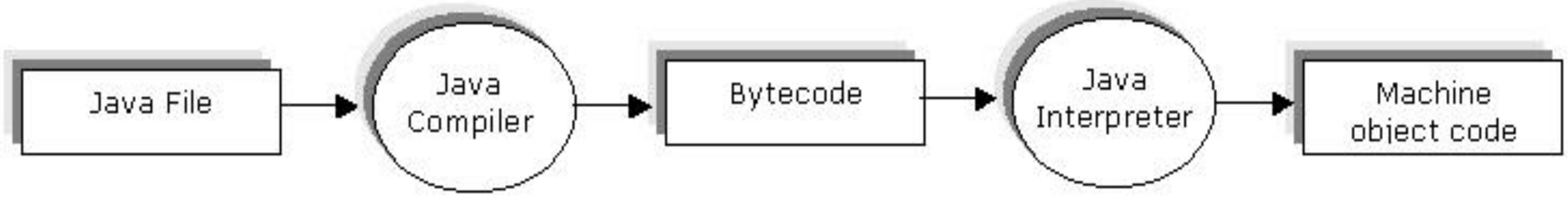
- Distributed:
  - Java is designed for the distributed environment of the Internet because it supports the various Internet protocols, such as Transmission Control Protocols and Internet Protocol (TCP/IP).
- Secure:
  - Java has built-in security features that verify that the programs do not perform any destructive task, such as accessing the files on a remote system.

Java Architecture

- Various components of Java Architecture are:
  - Java programming language and class file
  - JVM
  - Java Application Programming Interface (API)
- Java programming language and class file:
  - Java programs are saved with an extension, .java.
  - A .java file is compiled to generate the .class file, which contains the bytecode.
  - The JVM converts the bytecode contained in the .class file to machine object code.
  - The JVM needs to be implemented for each platform running on a different operating system.

Java Architecture (Contd.)

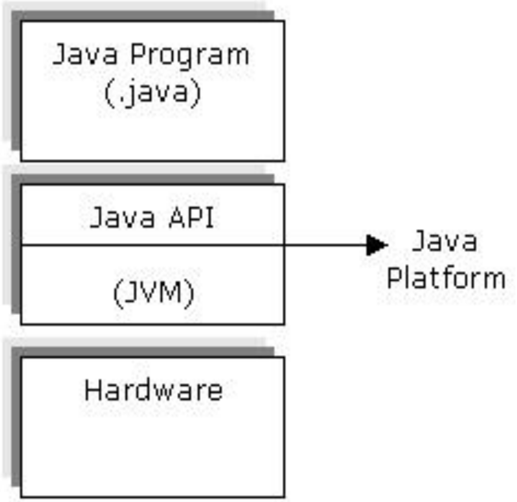
- The following figure shows the relationship among various components of the Java programming environment.



- JVM:
  - The JVM forms the base for the Java platform and is convenient to use on various hardware-based platforms.
  - The major components of the JVM are:
    - Class loader
    - Execution engine
    - Just In Time (JIT) compiler

Java Architecture (Contd.)

- Java Application Programming Interface (API):
  - The Java API is a collection of software components that provide capabilities, such as GUI.
  - The related classes and interfaces of the Java API are grouped into packages.
  - The following figure shows how the Java API and the JVM forms the platform for the Java programs on top of the hardware.

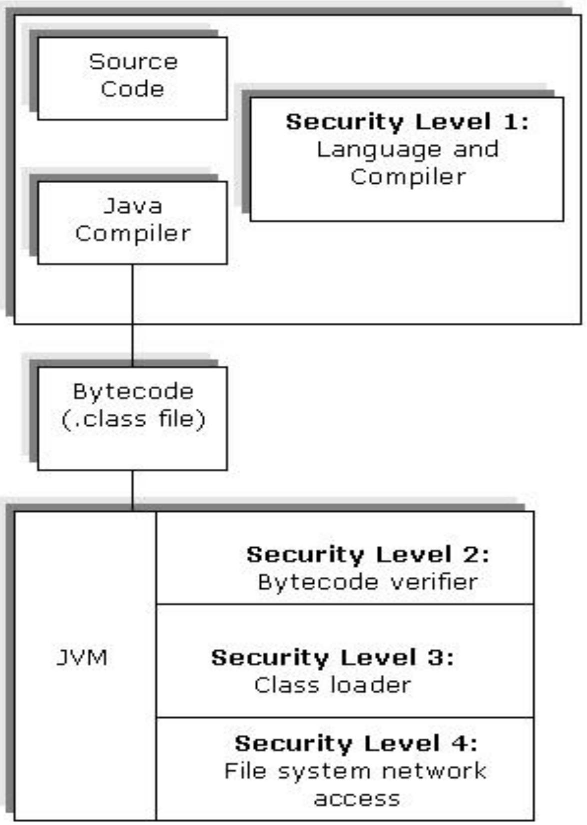


Java Architecture Security

- The Java architecture consists of the following security features that make Java a secure programming language:
  - Compiler level security
  - Bytecode verifier
  - Class loader
  - Sandbox model

Java Architecture Security (Contd.)

- The following figure shows the various levels of security implemented on Java programs.



Source Code

Security Level 1: Language and Compiler

Java Compiler

Bytecode (.class file)

JVM

Security Level 2: Bytecode verifier

Security Level 3: Class loader

Security Level 4: File system network access

Java Architecture Security (Contd.)

- Compiler level security:
  - Java prevents errors that arise due to improper memory usage, reducing the compile-time errors.
  - Typecasting in Java ensures that there is no data loss in the result or output of a Java code.
- Bytecode verifier:
  - It ensures that the bytecode does not violate access restrictions, such as read/write operations, and verifies that the bytecode does not forge pointers.
  - The bytecode is verified in two phases:
    - In the first phase, the verifier checks for the structure of the .class file.
    - The second level phase occurs when the bytecode is run. The bytecode verifier checks the validity of classes, variables, and methods used in a program.

Java Architecture Security (Contd.)

- Class loader:
  - The class loader determines how and when an applet will use classes in a running Java environment.
  - In a Java environment, there can be many class loaders and each class loader can create its own run-time environment.
  - The class loader loads all the applets and their references.
- Sandbox model:
  - The sandbox model is implemented in the Java applets container, such as Web browsers.
  - The sandbox model determines the limitations of Java applets that they can only access the resources of the host computer and cannot access the files on the local computer.

Garbage Collection in JVM

- Garbage collection is the process that is used to free the memory of the objects that are no longer in use.
- When a program stops referencing an object, it is not required any more and can be deleted.
- The space that is used by the object is released for use by another object.
- The garbage collection feature implies that the new objects are created and all the unreferenced objects are deallocated from the memory.

Garbage Collection in JVM (Contd.)

- The different approaches used for detecting garbage objects are:
  - Reference-counting collectors
  - Tracing collectors
  - Compacting collectors

Using Various Data Types

- The data stored in the memory of the computer can be of many types.
- Java is a strictly typed language, which means that Java gives importance to type checking.
- Expressions and variables in Java can be of different types.
- The various data types in Java are:
  - Primitive data types
  - Reference data types
  - Abstract data types

Using Various Data Types (Contd.)

- Primitive data types:
  - The built-in data types in Java are known as the primitive or the simple data types.
  - There are eight primitive data types in Java, which are further grouped in the following categories:
    - Integer type: Can store whole number values.
    - Floating point type: Can store fractional numbers.
    - Boolean type: Can store only the true and false values.
    - Character type: can store symbols, such as letters and numbers.

Using Various Data Types (Contd.)

- The following table lists the primitive data types with their size and range, grouped in four categories.

Group	Data Type	Size	Range	Default Value
Integer	`byte`	*One byte*	*-2^{7} to 2^{7-1} (signed)*	*0*
	`short`	*Two byte*	*-2^{15} to 2^{15-1}*	*0*
	`int`	*Four byte*	*-2^{31} to 2^{31-1}*	*0*
	`long`	*Eight byte*	*-2^{63} to 2^{63-1}*	*0*
Floating point	`float`	*Four byte*	*3.4^{e-038} to 3.4^{e+038}*	*0.0*
	`double`	*Eight byte*	*1.7^{e-308} to 1.7^{e+308}*	*0.0*
Boolean	`boolean`	*One bit*	*true or false*	*false*
Character	`char`	*Two byte*	*A single character*	*null*

Using Various Data Types (Contd.)

- Reference data types:
  - Contain the reference or an address of the dynamically created objects.
  - Are also known as non-primitive data types.
- The default value of a variable that is of reference data type, is null.
- The examples of reference data types in Java are:
  - Objects
  - Arrays
- Abstract data types:
  - It includes the data types derived from the primitive data types and have more functions than primitive data types.
  - For example, string is an abstract data type that stores letters, digits, and characters such as /, (), :, :, $, and #.

Using Various Data Types (Contd.)

- Keywords available in Java:
  - Keywords are the reserved words for a language, which express the language features.
  - Keywords cannot be used to name variables, constants, or classes.
  - Java is a case-sensitive language and the keywords should be written in lowercase only.
  - The keywords with all or some letters in uppercase can be treated as a variable name but that should be avoided.

Using Various Data Types (Contd.)

- The following table lists the Java keywords.

abstract	boolean	break	byte
case	catch	char	class
const	continue	default	do
double	else	extends	final
finally	float	for	goto
if	implements	import	instanceof
int	interface	long	native
new	package	private	protected
public	return	short	static
strictfp	super	switch	synchronized
this	throw	throws	transient
try	void	volatile	while
enum	assert		

Defining Variables and Literals

- A variable is the name that refers to a memory location where some data value is stored.
- You can assign different values to a variable during program execution.
- Java allocates memory to each variable that you use in your program.
- Each variable that is used in a program must be declared.

Defining Variables and Literals (Contd.)

- Naming conventions for a declaring variable in Java are:
  - The name of a variable needs to be meaningful, short, and without any embedded space or symbol.
  - A variable name must be unique.
  - A variable name must begin with a letter, an underscore (_), or the dollar symbol ($), which can be followed by a sequence of letters or digits (0 to 9), ‘$’, or ‘_’.
  - A variable name should not start with a digit.
  - A variable name should not contain embedded white spaces.
  - A variable name should not consist of a keyword.
  - A variable name in Java is case sensitive.

Defining Variables and Literals (Contd.)

- The various types of variables based on the variable scope in Java are:
  - Class variables
  - Instance variables
  - Local variables
  - Static variables
  - Automatic variables
- The following code snippet shows how to declare a variable:

```
<type> <variablename>;
// Single variable of given type
<type><var1,var2.....variable_n_name>
// Multiple variables of given type
```