Institute for Advanced Computing and Software Development, Akurdi.

IACSD

Concepts of Programming Notes

Dr. D.Y. Patil Educational Complex Sector 29,Near Akurdi Railway Station ,Nigdi Pradhikarn,Akurdi,Pune-44

About Java Features , development environment

Features of Java

Object Oriented

Everything in Java is coded using OO principles. This facilitates code modularization, reusability, testability, and performance.

Interpreted/Portable

Java source is compiled into platform-independent bytecode, which is then interpreted (compiled into native-code) at runtime. Java code is "Write Once, Run Everywhere"

Simple

Java has a familiar syntax, automatic memory management, exception handling, single inheritance, standardized documentation, and a very rich set of libraries .

Secure/Robust

Due to its support for strong type checking, exception handling, and memory management, Java is immune to buffer- overruns, leaked memory, illegal data access. Additionally, Java comes with a Security Manager too.

Scalable

Java is scalable both in terms of performance/throughput, and as a development environment. A single user can play a Java game on a mobile phone, or millions of users can shop though a Java-based e-commerce enterprise application.

High-performance/Multi-threaded

With its Just-in-Time compiler, Java can achieve (or exceed) performance of native applications. Java supports multi-threaded development out-of-the-box.

Dynamic

Java can load application components at run-time even if it knows nothing about them. Each class has a run-time representation.

Distributed

Java comes with support for networking, as well as for invoking methods on remote (distributed) objects through RMI.

About JVM,JRE,JDK

Java Development Kit [JDK] is the core component of Java Environment and provides all the tools, executable and binaries required to compile, debug and execute a Java Program. JDK is a platform specific software and that’s why we have separate installers for Windows, Mac and Unix systems.

Java Virtual Machine[JVM] is the heart of java programming language. When we run a program, JVM is responsible to converting Byte code to the machine specific code. JVM is also platform dependent and provides core java functions like memory management, garbage collection, security etc.

Java Runtime Environment [JRE] is the implementation of JVM, it provides platform to execute java programs. JRE consists of JVM and java binaries and other class libraries to execute any program successfully. To execute any java program, JRE is required.

JVM architecture with journey of java program from source code to execution stage

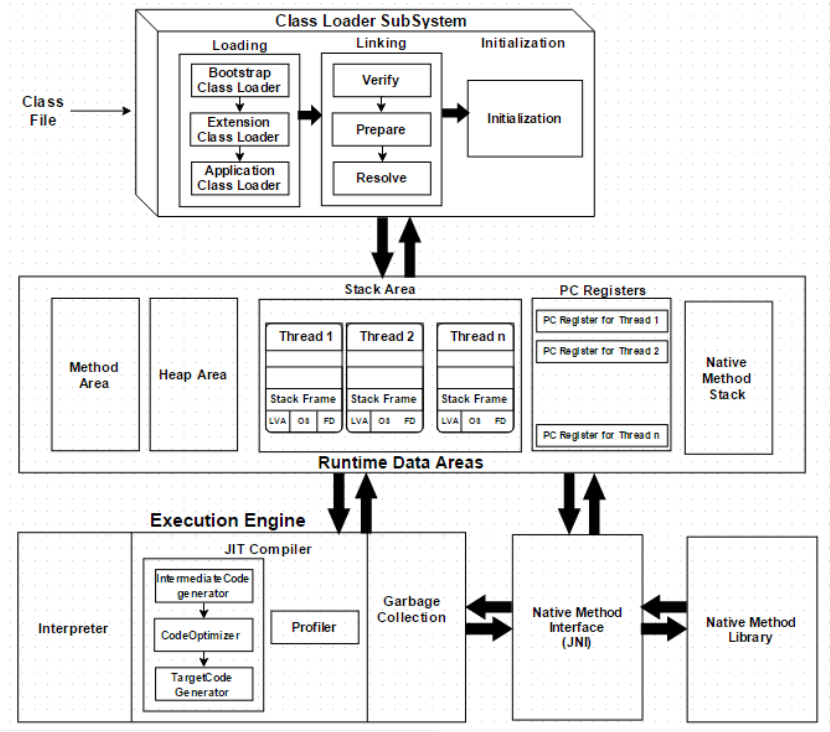
As shown in the below architecture diagram, JVM subsystems are :

- Class Loader Subsystem
- Runtime Data Area
- Execution Engine

Class Loader Subsystem

Loading : Class loader dynamically loads java classes. It loads, links and initializes the class file when it refers to a class for the first time at runtime, not compile time. Class Loaders follow Delegation Hierarchy Algorithm while loading the class files. 3 types are,

1. Boot Strap Class Loader – Responsible for loading classes from the bootstrap classpath, nothing but rt.jar. Highest priority will be given to this loader.
2. Extension Class Loader – Responsible for loading classes which are inside the ext folder (jre\lib).
3. Application Class Loader –Responsible for loading Application Level Class path, path mentioned Environment Variable etc.



Linking : Linking stage involves,

- Verify – Byte code verifier will verify byte code using checksum.
- Prepare – For all static variables memory will be allocated and assigned with default values.
- Resolve – All symbolic memory references are replaced with the original references from Method Area.
- Initialization: Here all static variables will be assigned with the original values, and the static block will be executed.

Runtime Data Area

The Runtime Data Area is divided into 5 major components:

- Method Area – All the class level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.
- Heap Area – All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe.
- Stack Area – For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called as Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource. The Stack Frame is divided into three sub entities:
    1. Local Variable Array – Related to the method how many local variables are involved and the corresponding values will be stored here.
    2. Operand stack – If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.
    3. Frame data – All symbols corresponding to the method is stored here. In the case of any exception, the catch block information will be maintained in the frame data.
- PC Registers – Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.
- Native Method stacks – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.

Execution Engine

The byte code which is assigned to the Runtime Data Area will be executed by the Execution Engine. The Execution Engine reads the byte code and executes it piece by piece.

- Interpreter – The interpreter interprets the byte code faster, but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.
- JIT Compiler – The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.
- Intermediate Code generator – Produces intermediate code
- Code Optimizer – Responsible for optimizing the intermediate code generated above
- Target Code Generator – Responsible for Generating Machine Code or Native Code
- Profiler – A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not.

Garbage Collector: Collects and removes unreferenced objects.

Java Native Interface (JNI): JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.

Native Method Libraries: This is a collection of the Native Libraries which is required for the Execution Engine.

Bytecode

Bytecode is in a compiled Java programming language [by javac command] format and has the .class extension executed by Java Virtual Machine (JVM). The Java bytecode gets processed by the Java virtual machine (JVM) instead of the processor. The JVM transforms program code into readable machine language for the CPU because platforms utilize different code interpretation techniques. A JVM converts bytecode for platform interoperability, but bytecode is not platform-specific. JVM is responsible for processing & running the bytecode.

JIT

The magic of java "Write once, run everywhere" is bytecode. JIT improves the performance of Java applications by compiling bytecode to native machine code at run time. JIT is activated when a Java method is called. The JIT compiler compiles the bytecode of that method into native machine code, compiling it "just in time" to run. When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it.

Typical compilers take source code and completely convert it into machine code, JITs take the same source code and convert it into an intermediary “assembly language,” which can then be pulled from when it’s needed. And that’s the key. Assembly code is interpreted into machine code on call—resulting in a faster translation of only the code that you need. JIT have access to dynamic runtime information and are able to optimize code. JITs monitor and optimize while they run by finding code more often called to make them run better in the future.

JITs reduce the CPU’s workload by not compiling everything all at once, but also because the resulting compiled code is optimized for that particular CPU. It’s why languages with JIT compilers are able to be so “portable” and run on any platform or OS.

Platform independence

Java is a platform independent programming language, because your source code can be executed on any platform [e.g. Windows, Mac or Linux etc..]. When you install JDK software on your system , JVM is automatically installed on your system. When we compile Java code then .class file or bytecode is generated by javac compiler. For every operating system separate JVM is available which is capable to read the .class file or byte code and execute it by converting to native code for that specific machine. We compile code once and run everywhere.

Language Fundamentals

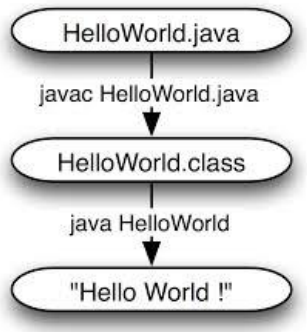
Sample Java Program

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello World!");
   }
```

- All code is contained within a class, in this case HelloWorld.
- The file name must match the class name and have a .java extension, for example: HelloWorld.java
- All executable statements are contained within a method, in this case named main().
- Use System.out.println() to print text to the terminal.
- Classes and methods (including other flow-control structures) are always defined in blocks of code enclosed by curly braces ({ }).
- All other statements are terminated with a semi-colon (;).
- Java language is case-sensitive.

Compiling Java Programs

- The JDK comes with a command-line compiler: javac.
- It compiles source code into Java bytecode, which is low-level instruction set similar to binary machine code.
- The bytecode is executed by a Java virtual machine (JVM), rather than a specific physical processor.
- To compile our HelloWorld.java, you could go to the directory containing the source file and execute: javac HelloWorld.java
- This produces the file HelloWorld.class, which contains the Java bytecode.
- You can view the generated byte-code, using the -c option to javap, the Java class disassembler. For example: javap -c HelloWorld



To run the bytecode, execute:

```
java HelloWorld
```

The main() Method

- A Java application is a public Java class with a main() method.

- The main() method is the entry point into the application.
- The signature of the method is always:
  public static void main(String[] args)
- Command-line arguments are passed through the args parameter, which is an array of Strings

Primitive Data Types

Type	Size	Range	Default Value
boolean	1 bit	true or false	false
byte	8 bits	[-128, 127]	0
short	16 bits	[-32,768, 32,767]	0
char	16 bits	['\u0000', '\uffff'] or [0, 65535]	'\u0000'
int	32 bits	[-2,147,483,648 to 2,147,483,647]	0
long	64 bits	[-263, 263-1]	0
float	32 bits	32-bit IEEE 754 floating-point	0.0
double	64 bits	64-bit IEEE 754 floating-point	0.0

Variables

Variable is nothing but identification of memory location.

Types of variables

Local

Variables declared inside method are local variable they have scope only within that methods.

Instance

Instance variables are declared inside class but used outside method.

Static

Static variable are same as that of instance variable but having keyword static.

They are also called as class variable because they are specified and can be access either class name or object name.

Operators in Java

1. Java Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and data.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

2. Java Assignment Operators

Assignment operators are used in Java to assign values to variables.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

3. Java Relational Operators

Relational operators are used to check the relationship between two operands.It returns either true or false.

Operator	Description	Example
==	Is Equal To	2 == 8 returns false
!=	Not Equal To	2 != 8 returns true
>	Greater Than	2 > 8 returns false
<	Less Than	2 < 8 returns true
>=	Greater Than or Equal To	2 >= 8 returns false
<=	Less Than or Equal To	2 <= 8 returns false

4. Java Logical Operators

Logical operators are used to check whether an expression is true or false. They are used in decision making.

Operator	Example	Meaning
&& (Logical AND)	expression1 && expression2	true only if both expression1 and expression2 are true
|| (Logical OR)	expression1 || expression2	true if either expression1 or expression2 is true
! (Logical NOT)	!expression	true if expression is false and vice versa

5. Java Unary Operators

Unary operators are used with only one operand.

Operator	Meaning
+	Unary plus: not necessary to use since numbers are positive without using it
-	Unary minus: inverts the sign of an expression
++	Increment operator: increments value by 1

--	Decrement operator: decrements value by 1
!	Logical complement operator: inverts the value of a boolean

6. Java Ternary Operator

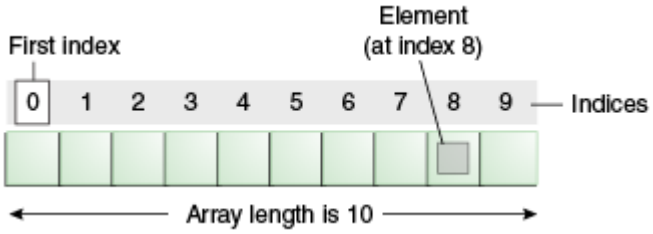
The ternary operator (conditional operator) is shorthand for the if-then-else statement. For example,

variable = Expression ? expression1 : expression2

Here,
If the Expression is true, expression1 is assigned to the variable.
If the Expression is false, expression2 is assigned to the variable.

Java Arrays

An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. You have seen an example of arrays already, in the main method of the "Hello World!" application. This section discusses arrays in greater detail.



An array of 10 elements.

Each item in an array is called an element, and each element is accessed by its numerical index. As shown in the preceding illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.In Java, array is an object of a dynamically generated class. Java array inherits the Object class.

Types of Array in java

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];

Instantiation of an Array in Java
arrVar=new datatype[size];

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

```
dataType[][] arrVar; (or)
dataType [][]arrVar; (or)
dataType arrVar [][]; (or)
dataType []arrVar [];
```

Instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Jagged Array in Java

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

```
int arr[][] = new int[3][];
    arr[0] = new int[3];
    arr[1] = new int[4];
    arr[2] = new int[2];
```

Copying Arrays

The System class has an arraycopy method that you can use to efficiently copy data from one array into another:

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos, int length)
```

The two Object arguments specify the array to copy from and the array to copy to. The three int arguments specify the starting position in the source array, the starting position in the destination array, and the number of array elements to copy.

for-each Loop for Java Array

We can also print the Java array using for-each loop. The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop.

The syntax of the for-each loop is given below:

```
for(data_type variable:array){
    //body of the loop
}
```

Object Oriented Concepts in Java

Classes in Java

- Everything in Java is defined in a class.
- In its simplest form, a class just defines a collection of data
  e.g.

```
class Employee {
        int empid;
        String name;
        double salary;
}
```

Objects in Java

- To create an object (instance) of a particular class, use the new operator, followed by an invocation of a constructor for that class, such as:

```
new Employee();
```

- The constructor method initializes the state of the new object.
- The new operator returns a reference to the newly created object.
- As with primitives, the variable type must be compatible with the value type when using object references, as in:

```
Employee e = new Employee();
```

- To access member data or methods of an object, use the dot (.) notation: variable.field or variable.method().

Static vs. Instance members

Static (or class) data members

- Unique to the entire class
- Shared by all instances (objects) of that class
- Accessible using ClassName.fieldName
- The class name is optional within static and instance methods of the class, unless a local variable of the same name exists in that scope.
- Subject to the declared access mode, accessible from outside the class using the same syntax

Instance or data members

- Unique to each instance (object) of that class (that is, each object has its own set of instance fields)
- Accessible within instance methods and constructors using this.fieldName
- The **this** qualifier is optional, unless a local variable of the same name exists in that scope.
- Subject to the declared access mode, accessible from outside the class from an object reference using objectRef.fieldName

Static vs. Instance Methods

- Static methods can access only static data and invoke other static methods.
  - Often serve as helper procedures/functions
  - Use when the desire is to provide a utility or access to class data only
- Instance methods can access both instance and static data and methods.
  - Implement behavior for individual objects
  - Use when access to instance data/methods is required
- An example of static method use is Java's Math class.
  - All of its functionality is provided as static methods implementing mathematical functions (e.g., Math.sin()).
  - The Math class is designed so that you don't (and can't) create actual Math instances.
- Static methods also are used to implement factory methods for creating objects, a technique discussed later in this class.

Constructors

- Constructors are like special methods that are called implicitly as soon as an object is instantiated (i.e. on new ClassName()).
  - Constructors have no return type (not even void).
  - The constructor name must match the class name.
- If you don't define an explicit constructor, Java assumes a default constructor
  - The default constructor accepts no arguments.
  - The default constructor automatically invokes its base class constructor with no arguments, as discussed later in this module.
- You can provide one or more explicit constructors to:
  - Simplify object initialization (one line of code to create and initialize the object)
  - Enforce the state of objects (require parameters in the constructor)
  - Invoke the base class constructor with arguments, as discussed later in this module.
- Adding any explicit constructor disables the implicit (no argument) constructor.
- As with methods, constructors can be overloaded.
- Each constructor must have a unique signature.
  - The parameter type list must be different, either different number or different order.
  - Only parameter types determine the signature, not parameter names.
- One constructor can invoke another by invoking this(param1, param2, …) as the first line of its implementation.

Access Modifiers: Enforcing Encapsulation

- Access modifiers are Java keywords you include in a declaration to control access.
- You can apply access modifiers to:
  - Instance and static fields
  - Instance and static methods
  - Constructors
  - Classes
  - Interfaces (discussed later in this module)

Access Modifier	Description
public	Accessible from any class
protected	Accessible from all classes in the **same package** or any **child classes** regardless of the package
Default (no modifier)	Accessible only from the classes in the **same package** (also known as *friendly*)
private	Accessible only from within the same class (or any nested classes)

Primitive Wrappers / Wrapper Classes

- The java.lang package includes a class for each Java primitive type:
  - Boolean, Byte, Short, *Character*, *Integer*, Float, Long, Double, *Void*
- Used for:
  - Storing primitives in Object based collections
  - Parsing/decoding primitives from Strings, for example:
    int value = Integer.parseInt(str);
  - Converting/encoding primitives to Strings
- Since Java 5, Java supports implicit wrapping/unwrapping of primitives as needed. This compiler-feature is called auto-boxing/auto-unboxing.
- **Autoboxing:** Converting a primitive value into an object of the corresponding wrapper class is called autoboxing.
- **Unboxing:** Converting an object of a wrapper type to its corresponding primitive value is called unboxing

  Integer Iobj = 10; //Boxing
  int k = Iobj //Unboxing

Java Packages

A package is simply a container that groups related types (Java classes, interfaces, enumerations, and annotations). For example, in core Java, the System class belongs to the java.lang package. The package contains all the related types that are needed for the basic java development.

- **Built-in Package**
  Built-in packages are existing java packages that come along with the JDK. For example, java.lang, java.util, java.io

- **User-defined Package**
  Java also allows you to create packages as per your need. These packages are called user-defined packages.

Defining a Java package

To define a package in Java, you use the keyword package.

package packageName;
Java uses file system directories to store packages.

For example:

```
└── com
  └── test
    └── TestPlanet.java
```

Now, edit Test.java file, and at the beginning of the file, write the package statement as:
package com.test;

Here, any class that is declared within the test directory belongs to the com.test package.

Importing packages in Java

Java has an import statement that allows you to import an entire package, or use only certain classes and interfaces defined in the package.

The general form of import statement is:

```
import package.name.ClassName;   // To import a certain class only
import package.name.*   // To import the whole package
```
For example,

```
import java.util.Date; // imports only Date class
import java.io.*;      // imports everything inside java.io package
```
