Unit 1

Introduction to Java

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere.**

Java is –

- **Object Oriented** – In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** – Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** – Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** – With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** – Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

- **Portable** – Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** – Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** – With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** – Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** – With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** – Java is designed for the distributed environment of the internet.
- **Dynamic** – Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

History of Java

James Gosling initiated Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called ‘Oak’ after an oak tree that stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May, 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

Tools You Will Need

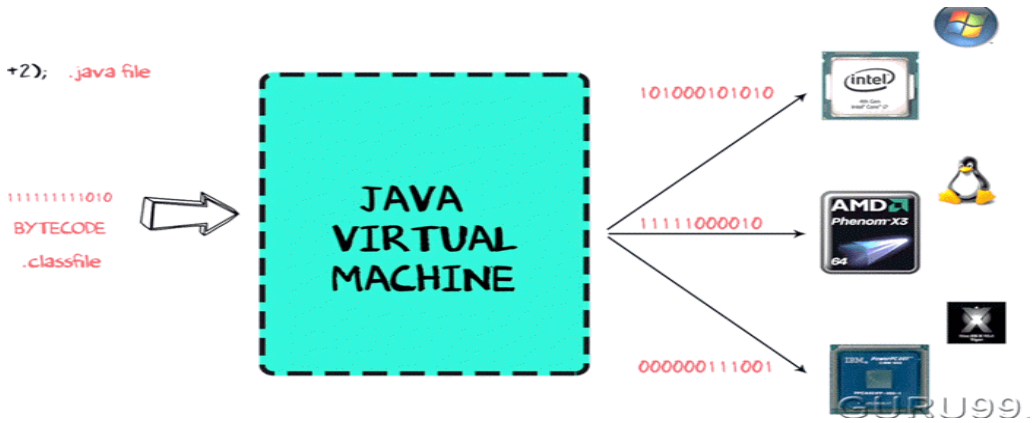
You will need a Pentium 200-MHz computer with a minimum of 64 MB of RAM (128 MB of RAM recommended).

You will also need the following softwares –

- Linux 7.1 or Windows xp/7/8 operating system
- Java JDK 8
- Microsoft Notepad or any other text editor

How Java Virtual Machine works?

By using Java Virtual Machine, this problem can be solved. But how it works on different processors and O.S. Let's understand this process step by step.



Step 1: The code to display addition of two numbers is System.out.println(1+2), and saved as .java file.

Step 2: Using the java compiler the code is converted into an intermediate code called the bytecode. The output is a .class file.

Step 3: This code is not understood by any platform, but only a virtual platform called the Java Virtual Machine.

Step 4: This Virtual Machine resides in the RAM of your operating system. When the Virtual Machine is fed with this bytecode, it identifies the platform it is working on and converts the bytecode into the native machine code.

In fact, while working on your PC or browsing the web whenever you see either of these icons be assured the java virtual machine is loaded into your RAM. But what makes java lucrative is that code once compiled can run not only on all PC platforms but also mobiles or other electronic gadgets supporting java.

Hence,

"Java is a programming language as well as a Platform"

How is Java Platform Independent?

Like C compiler, Java compiler does not produce native executable code for a particular machine. Instead, Java produces a unique format called bytecode. It executes according to the rules laid out in the virtual machine specification.

Bytecode is understandable to any JVM installed on any OS. In short, the java source code can run on all operating systems.

What is JVM?

JVM is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of JRE(Java Run Environment). It stands for Java Virtual Machine

- In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.
- First, Java code is complied into bytecode. This bytecode gets interpreted on different machines
- Between host system and Java source, Bytecode is an intermediary language.
- JVM is responsible for allocating memory space.

Why is Java both Interpreted and Compiled Language?

Programming languages are classified as

- Higher Level Language Ex. C++, Java
- Middle-Level Languages Ex. C
- Low-Level Language Ex Assembly
- finally the lowest level as the Machine Language.

A **compiler** is a program which converts a program from one level of language to another. Example conversion of C++ program into machine code.

The java compiler converts high-level java code into bytecode (which is also a type of machine code).

An **interpreter** is a program which converts a program at one level to another programming language at the **same level.** Example conversion of Java program into C++

In Java, the Just In Time Code generator converts the bytecode into the native machine code which are at the same programming levels.

Hence, Java is both compiled as well as interpreted language.

Why is Java slow?

The two main reasons behind the slowness of Java are

1. **Dynamic Linking:** Unlike C, linking is done at run-time, every time the program is run in Java.
2. **Run-time Interpreter:** The conversion of byte code into native machine code is done at run-time in Java which furthers slows down the speed.

Just-In-Time compiler (JIT)

In the Java programming language and environment, a just-in-time (JIT) compiler is a program that turns Java bytecode (a program that contains instructions that must be interpreted) into instructions that can be sent

directly to the processor. After you've written a Java program, the source language statements are compiled by the Java compiler into bytecode rather than into code that contains instructions that match a particular hardware platform's processor (for example, an Intel Pentium microprocessor or an IBM System/390 processor). The bytecode is platform-independent code that can be sent to any platform and run on that platform.

Using the Java just-in-time compiler (really a second compiler) at the particular system platform compiles the bytecode into the particular system code (as though the program had been compiled initially on that platform). Once the code has been (re-)compiled by the JIT compiler, it will usually run more quickly in the computer.

Comparison of C++ and Java

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Design Goal	C++ was designed for	Java was designed and

	systems and applications programming. It was an extension of C programming language.	created as an interpreter for printing systems but later extended as a support network computing. It was designed with a goal of being easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is

		platform independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are the

		child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in C language, single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Unit-2
Java - Basic Datatypes

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in the memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java –

- Primitive Data Types
- Reference/Object Data Types

Primitive Data Types

There are eight primitive datatypes supported by Java. Primitive datatypes are predefined by the language and named by a keyword. Let us now look into the eight primitive data types in detail.

byte

- Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

short

- Short data type is a 16-bit signed two's complement integer
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.
- Example: short s = 10000, short r = -20000

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 (-2^31)
- Maximum value is 2,147,483,647(inclusive) (2^31 -1)
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: int a = 100000, int b = -200000

long

- Long data type is a 64-bit signed two's complement integer
- Minimum value is -9,223,372,036,854,775,808(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)
- This type is used when a wider range than int is needed
- Default value is 0L
- Example: long a = 100000L, long b = -200000L

float

- Float data type is a single-precision 32-bit IEEE 754 floating point

- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example: float f1 = 234.5f

double

- double data type is a double-precision 64-bit IEEE 754 floating point
- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0d
- Example: double d1 = 123.4

boolean

- boolean data type represents one bit of information
- There are only two possible values: true and false
- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example: boolean one = true

char

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character
- Example: char letterA = 'A'

Java - Basic Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups –

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators –

Assume integer variable A holds 10 and variable B holds 20, then –

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200

/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

The Relational Operators

There are following relational operators supported by Java language.

Assume variable A holds 10 and variable B holds 20, then −

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.

< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Bitwise Operators

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows −

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators −

Assume integer variable A holds 60 and variable B holds 13 then −

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators

The following table lists the logical operators −

Assume Boolean variables A holds true and variable B holds false, then −

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

The Assignment Operators

Following are the assignment operators supported by Java language −

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C

		+ A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2

&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Miscellaneous Operators

There are few other operators supported by Java Language.

Conditional Operator (? :)

Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as −

```
variable x = (expression) ? value if true : value if false
```

Following is an example −

Example

```
public class Test {


   public static void main(String args[]) {
      int a, b;
      a = 10;
```

```
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

This will produce the following result –

Output

```
Value of b is : 30
Value of b is : 20
```

instanceof Operator

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as –

```
( Object reference variable ) instanceof  (class/interface type)
```

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is an example –

Example

```
public class Test {

   public static void main(String args[]) {

      String name = "James";
```

```
      // following will return true since name is type of String
      boolean result = name instanceof String;
      System.out.println( result );
   }
}
```

This will produce the following result –

Output

```
true
```

This operator will still return true, if the object being compared is the assignment compatible with the type on the right. Following is one more example –

Example

```
class Vehicle {}

public class Car extends Vehicle {

   public static void main(String args[]) {

      Vehicle a = new Car();
      boolean result =  a instanceof Car;
      System.out.println( result );
   }
}
```

This will produce the following result –

Output

```
true
```

Precedence of Java Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator −

For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3 * 2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

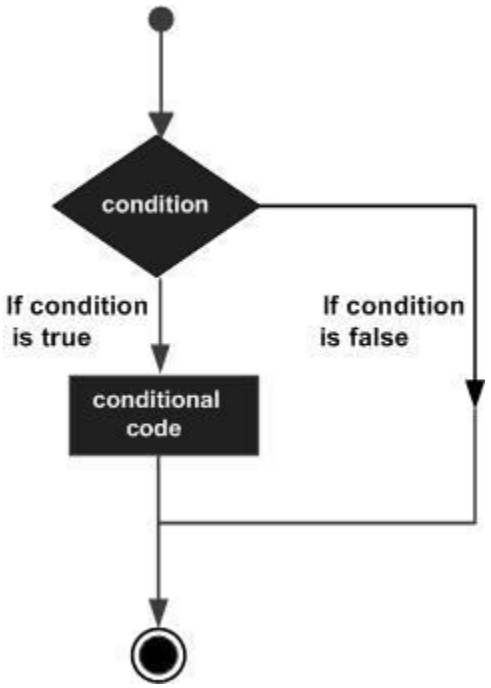
Category	Operator	Associativity
Postfix	expression++ expression--	Left to right
Unary	++expression –-expression +expression –expression ~ !	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >> >>>	Left to right
Relational	< > <= >= instanceof	Left to right

Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= ^= |= <<= >>= >>>=	Right to left

Java - Decision Making

Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages –



Java programming language provides following types of decision making statements. Click the following links to check their detail.

Sr.No.	Statement & Description
1	**if statement** An **if statement** consists of a boolean expression followed by one or more statements.
2	**if...else statement** An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
3	**nested if statement** You can use one **if** or **else if** statement inside another **if** or **else if**statement(s).

4	**switch statement** A **switch** statement allows a variable to be tested for equality against a list of values.

The ? : Operator

We have covered **conditional operator ? :** in the previous chapter which can be used to replace **if...else** statements. It has the following general form –

```
Exp1 ? Exp2 : Exp3;
```

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

To determine the value of the whole expression, initially exp1 is evaluated.

- If the value of exp1 is true, then the value of Exp2 will be the value of the whole expression.
- If the value of exp1 is false, then Exp3 is evaluated and its value becomes the value of the entire expression.

Unit 3

Defining A Class

Everything in Java is associated with classes and objects, along with its attributes and methods.

Example: in real life, a car is an object. The car has **attributes**, i.e: weight and color.

Methods i.e: drive and brake. A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword class:

Example

```
public class Student {
        int x = 5;
}
```

Where public is an accessifier, class is an keyword, and student is name of class.

Fields Declaration:-

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables.

Example:

```
Class Rectangle
{       int length;
        int width;
}
```

Methods Declaration:-

The general form of a method declaration is:-

```
type method name(parameter list)
{
        method body;
}
```

Creating Objects:-

Objects in java are created using the **new** operator. The new operator creates an object of the specified class and returns a reference to that object.

Rectangle rect 1 ; //declare the object

rect 1 = new Rectangle () ; //instantiate the object

Rectangle rect 1 = new Rectangle () ;

Rectangle is a class name,rect1 is an object

Accessing Class Members:-

- **Accessing Attributes**

  The instance variables of the Rectangle class may be accessed and assinged value as follows:

  rect 1. length = 15 ;

  rect 1. width = 10 ;

- **Accessing Methods**

  Name of object .(dotoperator) name of method (parameter list)

  rect 1. get Data (15,10) ;

  name of object (rect1)

Modifiers

We divide modifiers into two groups:

- **Access Modifiers** - controls the access level
- **Non-Access Modifiers** - do not control access level, but provides other functionality

Access Modifiers

For **classes**, you can use either public or *default*:

Modifier	**Description**	

public	The class is accessible by any other class	
default	The class is only accessible by classes in the same package.	

For **attributes, methods and constructors**, you can use the one of the following:

Modifier	**Description**	
public	The code is accessible for all classes	
private	The code is only accessible within the declared class	
default	The code is only accessible in the same package.	
protected	The code is accessible in the same package and **subclasses**.	

Very Short and short type questions :-

1. Define class.
2. How will you create object in java?
3. What operator is used in object creation in java?
4. What do you mean by accessifier in java

Long type questions :-

1. Explain with example class and create object in java.
2. Explain in detail java Accessifier.

Inheritance

Inheritance is a mechanism in which one class acquires the property of another class. Inheritance is the process of creating new class called derived class from existing ones i.e. base class.

Visibility Modes:

There are three visibility modes:

Private mode: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class. Private members of the base class will never get inherited in sub class.

Protected mode: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class. Private members of the base class will never get inherited in sub class

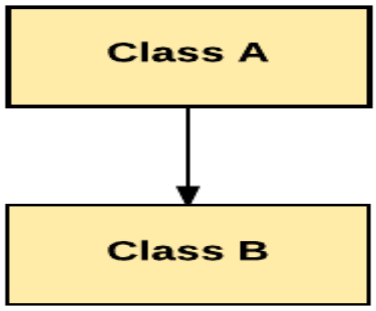
Public mode: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class. Private members of the base class will never get inherited in sub class.

Types of Inheritance

There are various types of inheritance in Java:

1. Single Inheritance:

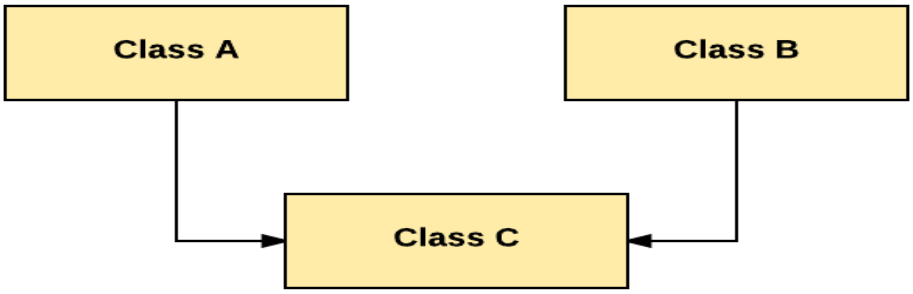
In Single Inheritance one class extends another class (one class only).



In above diagram, Class B extends only Class A. Class A is a super class and Class B is a Sub-class.

2. Multiple Inheritance:

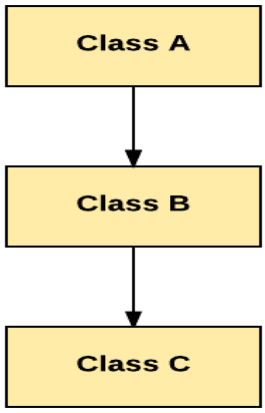
In Multiple Inheritance, one class extending more than one class. Java does not support multiple inheritance.



As per above diagram, Class C extends Class A and Class B both.

3. Multilevel Inheritance:

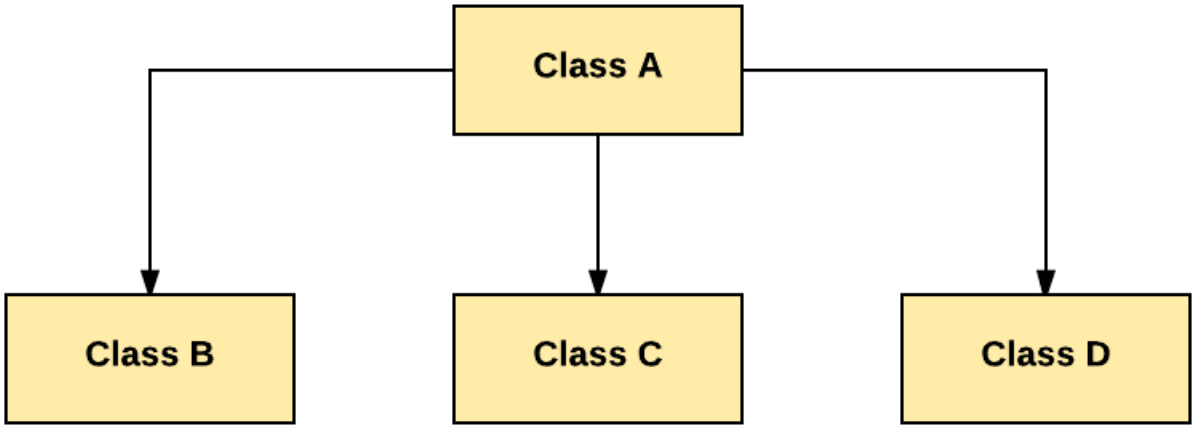
In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.



As per shown in diagram Class C is subclass of B and B is a of subclass Class A.

4. Hierarchical Inheritance:

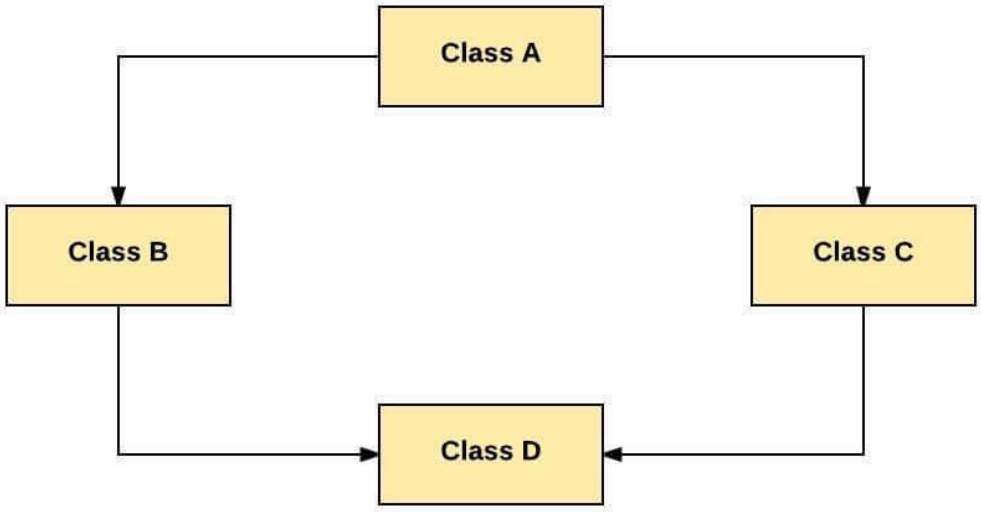
In Hierarchical Inheritance, one class is inherited by many sub classes.



As per above example, Class B, C, and D inherit the same class A.

5. Hybrid Inheritance:

Hybrid inheritance is a combination of Single and Multiple inheritance.



As per above example, all the public and protected members of Class A are inherited into Class D, first via Class B and secondly via Class C.

Very Short and short type questions :-

1. What is Inheritance in Java?
2. What are different types of Inheritance supported by Java?
3. Why multiple Inheritance is not supported by Java?
4. What is the syntax of Inheritance?

Long type questions :-

1. What is inheritance in Java and types of inheritance?

Polymorphism

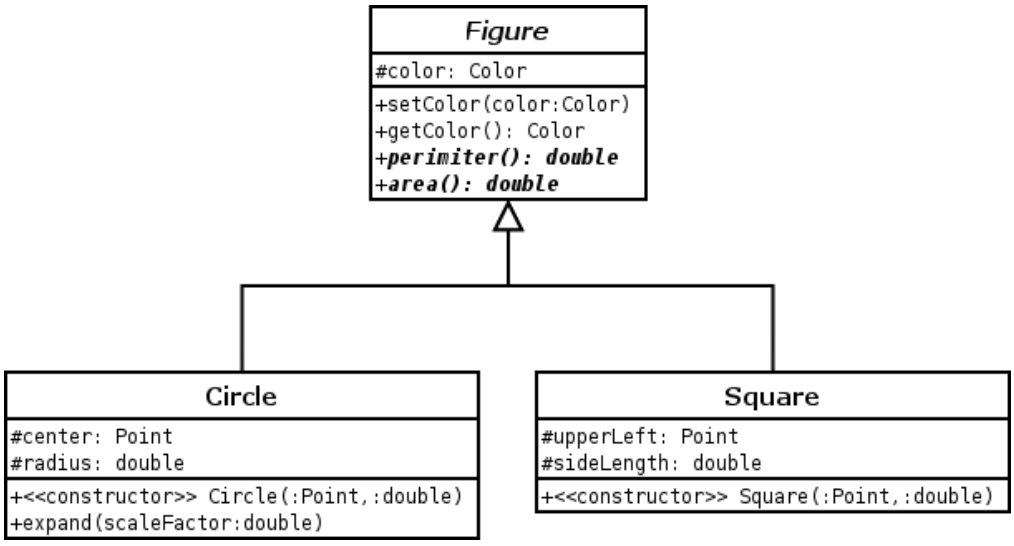
"Poly means Many and Morphism means forms"

Polymorphism in Java is a concept by which we can perform a single action in different ways. So **polymorphism** means many forms.

There are two types of **polymorphism in Java**:

1. Compile-time **polymorphism**
2. Runtime **polymorphism**

We can perform **polymorphism in java** by method overloading and method overriding.



Method Overloading

Method Overloading is a feature that allows a class to have more than one **method** having the same name, if their argument lists are different.

It is similar to constructor **overloading in Java**, that allows a class to have more than one constructor having different argument lists.

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Constructor Overloading

In addition to **overloading** methods, we can also **overload constructors** in **java**.

Overloaded constructor is called based upon the parameters specified when new is executed. Sometimes there is a need of initializing an object in different ways. This can be done using **constructor overloading**.

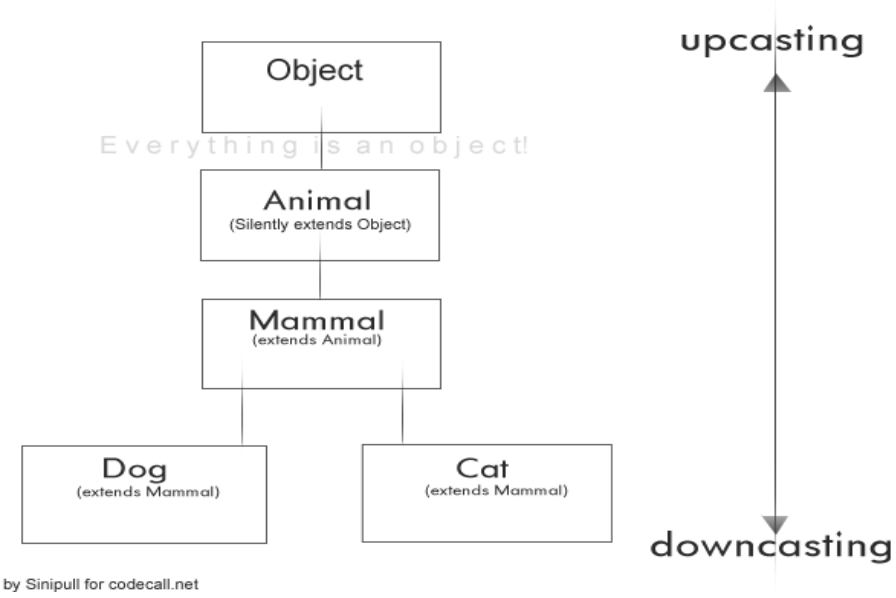
Constructor overloading is a concept of having more than one **constructor** with different parameters list, in such a way so that each **constructor** performs a different task. For e.g. Vector class has 4 types of **constructors**.

Up casting And Down casting

Up casting means casting the object to a super type, while **down casting means** casting to a subtype.

In **java**, **up casting** is not necessary as it's done automatically. And it's usually referred as implicit casting. Actually at runtime it could be different and **java** will throw a Class Cast Exception , would that happen.

Down casting is used more frequently than **up casting**. Use down casting when we want to access specific behaviors of a subtype. Here, in the teach method, we check if there is an instance of a Dog object passed in, downcast it to the Dog type and invoke its specific method.



Very Short and short type questions :-

1. What is polymorphism in java?
2. What is Static member Class?
3. What is difference between 'Overloading and Overriding?

Long type questions :-

1. What is Function overloading and Overriding in Java?

Unit -4
Interface

Abstract class: - Abstract class can implement interfaces without even providing the implementation of interface methods. Java Abstract class is used to provide common method implementation to all the subclasses or to provide default implementation.

Abstract interface:- The interface is a blueprint that can be used to implement a class. The interface does not contain any concrete methods (methods that have code). All the methods of an interface are abstract methods. An interface cannot be instantiated. However, classes that implement interfaces can be instantiated.

Use of abstract types:-

1. Abstract types are an important feature in statically typed OOP languages.

2. Abstract types are useful in that they can be used to define and enforce a protocol; a set of operations that all objects implementing the protocol must support.

Difference between Abstract class and Interface

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract**methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.

4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class**can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class**can be extended using keyword "extends".	An **interface class**can be implemented using keyword "implements".
8) A Java**abstract class**can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Very Short and short type questions :-

1. What is Abstract class in java?
2. What is Interaface in java?
3. Can abstract class have constructors in Java?
4. Can abstract class implements interface in Java?
5. Can abstract class have static methods in Java?

Long type questions :-

1. What is the difference between interface and abstract class in Java?

Unit – 5

Exception Handling

An exception is an "unwanted or unexpected event", which occurs during the execution of the program i.e, at run-time, that disrupts the normal flow of the program's instructions. When an exception occurs, execution of the program gets terminated.

Why does an Exception occurs?

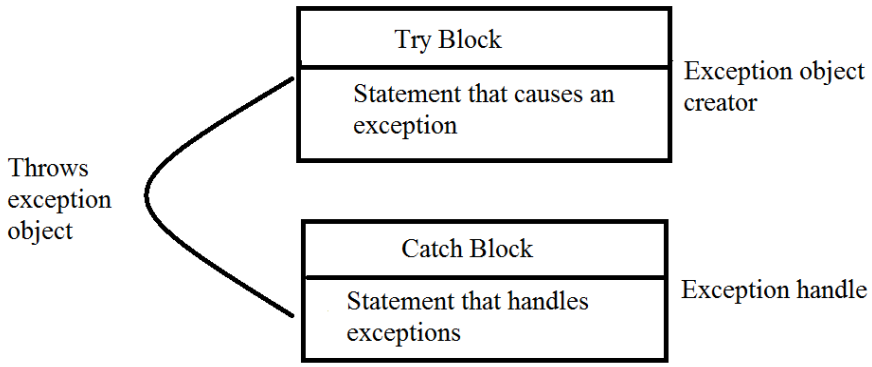
An exception can occur due to several reasons like Network connection problem, Bad input provided by user, Opening a non-existing file in your program etc

Keywords used for exception handling:-

1. **Try**: The try block contains set of statements where an exception can occur.

```
try
{
// statement(s) that might cause exception
}
```

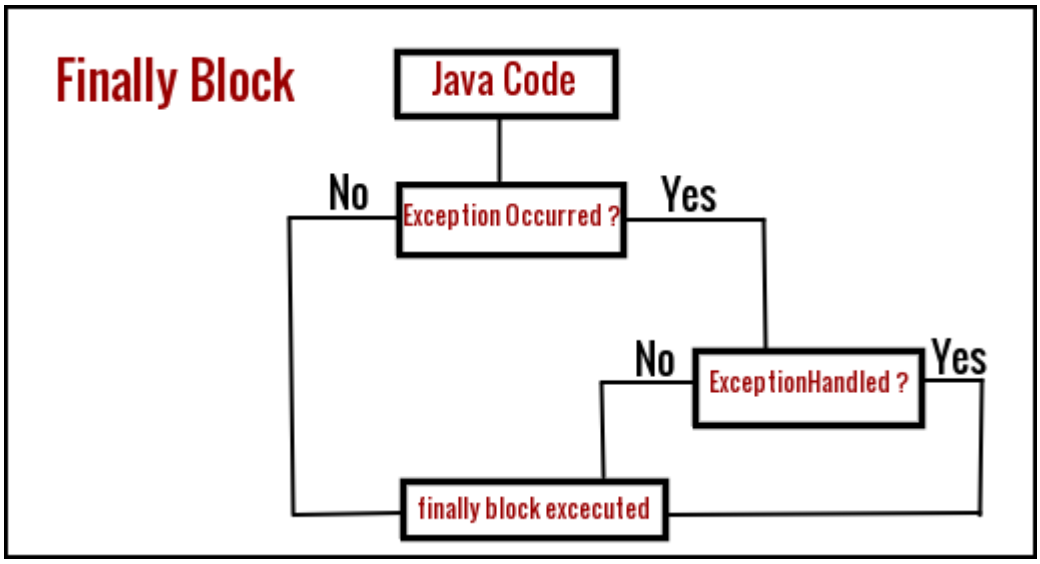
2. **Catch :** Catch block is used to handle the uncertain condition of try block. A try block is always followed by a catch block, which handles the exception that occurs in associated try block.



3. **Throw:** Throw keyword is used to transfer control from try block to catch block.

4. **Throws**: Throws keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

5. **Finally**: It is executed after catch block. We basically use it to put some common code when there are multiple catch blocks.



Very Short and short type questions :-

1. What is Exception in Java?
2. What are the Exception Handling Keywords in Java?
3. What is difference between throw and throws keyword in Java?

Long type questions :-

1. What is exception and types of exception in Java?
2. Explain in Detail TRY and CATCH block in java?

Unit-6

Multi threading

Any application can have multiple processes (instances). Each of this process can be assigned either as a single thread or multiple threads.

We will see in this tutorial how to perform multiple tasks at the same time and also learn more about threads and synchronization between threads.

What is Single Thread?

A single thread is basically a lightweight and the smallest unit of processing. Java uses threads by using a "Thread Class".

There are two types of thread – **user thread and daemon thread** (daemon threads are used when we want to clean the application and are used in the background).

When an application first begins, user thread is created. Post that, we can create many user threads and daemon threads.

Single Thread Example:

```
package demotest;


public class GuruThread
{
	public static void main(String[] args) {
		System.out.println("Single Thread");
	}
}
```

Advantages of single thread:

- Reduces overhead in the application as single thread execute in the system
- Also, it reduces the maintenance cost of the application.

What is Multithreading?

Multithreading in java is a process of executing two or more threads simultaneously to maximum utilization of CPU.

Multithreaded applications are where two or more threads run concurrently; hence it is also known as **Concurrency** in Java. This multitasking is done, when multiple processes share common resources like CPU, memory, etc.

Each thread runs parallel to each other. Threads don't allocate separate memory area; hence it saves memory. Also, context switching between threads takes less time.

Example of Multi thread:

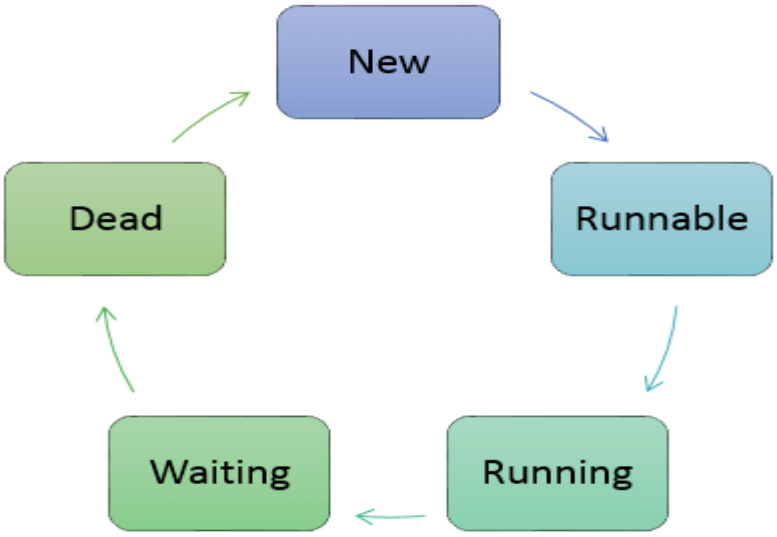
```
package demotest;

public class GuruThread1 implements Runnable
{
       public static void main(String[] args) {
        Thread guruThread1 = new Thread("Guru1");
        Thread guruThread2 = new Thread("Guru2");
        guruThread1.start();
        guruThread2.start();
        System.out.println("Thread names are following:");
        System.out.println(guruThread1.getName());
        System.out.println(guruThread2.getName());
    }
    @Override
    public void run() {
    }

}
```

Advantages of multithread:

- The users are not blocked because threads are independent, and we can perform multiple operations at times
- As such the threads are independent, the other threads won't get affected if one thread meets an exception.

Thread Life Cycle in Java

The Lifecycle of a thread:



There are various stages of life cycle of thread as shown in above diagram:

New

Runnable

Running

Waiting

Dead

New: In this phase, the thread is created using class "Thread class".It remains in this state till the program starts the thread. It is also known as born thread.

Runnable: In this page, the instance of the thread is invoked with a start method. The thread control is given to scheduler to finish the execution. It depends on the scheduler, whether to run the thread.

Running: When the thread starts executing, then the state is changed to "running" state. The scheduler selects one thread from the thread pool, and it starts executing in the application.

Waiting: This is the state when a thread has to wait. As there multiple threads are running in the application, there is a need for synchronization between threads. Hence, one thread has to wait, till the other thread gets executed. Therefore, this state is referred as waiting state.

Dead: This is the state when the thread is terminated. The thread is in running state and as soon as it completed processing it is in "dead state".

Some of the commonly used methods for threads are:

Method	Description
start()	This method starts the execution of the thread and JVM calls the run() method on the thread.
Sleep(int milliseconds)	This method makes the thread sleep hence the thread's execution will pause for milliseconds provided and after that, again the thread starts executing. This help in synchronization of the threads.
getName()	It returns the name of the thread.
setPriority(int newpriority)	It changes the priority of the thread.
yield ()	It causes current thread on halt and other threads to execute.

Example: In this example we are going to create a thread and explore built-in methods available for threads.

```
package demotest;
public class thread_example1 implements Runnable {
    @Override
    public void run() {
    }
    public static void main(String[] args) {
        Thread guruthread1 = new Thread();
        guruthread1.start();
        try {
            guruthread1.sleep(1000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        guruthread1.setPriority(1);
        int gurupriority = guruthread1.getPriority();
        System.out.println(gurupriority);
        System.out.println("Thread Running");
  }
}
```

Explanation of the code:

Code Line 2: We are creating a class "thread_Example1" which is implementing the Runnable interface (it should be implemented by any class whose instances are intended to be executed by the thread.)

Code Line 4: It overrides run method of the runnable interface as it is mandatory to override that method

Code Line 6: Here we have defined the main method in which we will start the execution of the thread.

Code Line 7: Here we are creating a new thread name as "guruthread1" by instantiating a new class of thread.

Code Line 8: we will use "start" method of the thread using "guruthread1" instance. Here the thread will start executing.

Code Line 10: Here we are using the "sleep" method of the thread using "guruthread1" instance. Hence, the thread will sleep for 1000 milliseconds.

Code 9-14: Here we have put sleep method in try catch block as there is checked exception which occurs i.e. Interrupted exception.

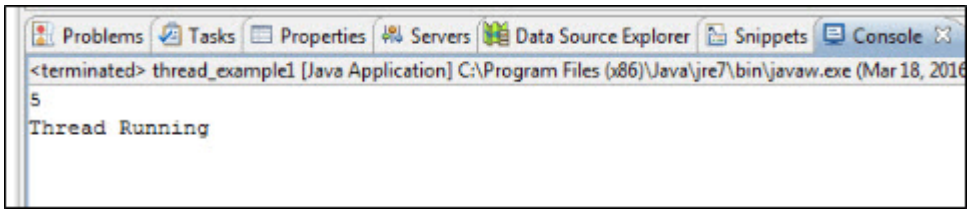
Code Line 15: Here we are setting the priority of the thread to 1 from whichever priority it was

Code Line 16: Here we are getting the priority of the thread using getPriority()

Code Line 17: Here we are printing the value fetched from getPriority

Code Line 18: Here we are writing a text that thread is running.

When you execute the above code, you get the following output:



Output:

5 is the Thread priority, and Thread Running is the text which is the output of our code.

Java Thread Synchronization

In multithreading, there is the asynchronous behavior of the programs. If one thread is writing some data and another thread which is reading data at the same time, might create inconsistency in the application.

When there is a need to access the shared resources by two or more threads, then synchronization approach is utilized.

Java has provided synchronized methods to implement synchronized behavior.

In this approach, once the thread reaches inside the synchronized block, then no other thread can call that method on the same object. All threads

have to wait till that thread finishes the synchronized block and comes out of that.

In this way, the synchronization helps in a multithreaded application. One thread has to wait till other thread finishes its execution only then the other threads are allowed for execution.

It can be written in the following form:

```
Synchronized(object)
{
        //Block of statements to be synchronized
}
```

Unit-7

Java Applets

Java Applet Basics

Applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. Applet is embedded in a HTML page using the APPLET or OBJECT tag and hosted on a web server.

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Applets are used to make the web site more dynamic and entertaining.

Some important points :

1. All applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class.
2. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
3. In general, execution of an applet does not begin at main() method.
4. Output of an applet window is not performed by *System.out.println()*. Rather it is handled with various AWT methods, such as *drawString()*.

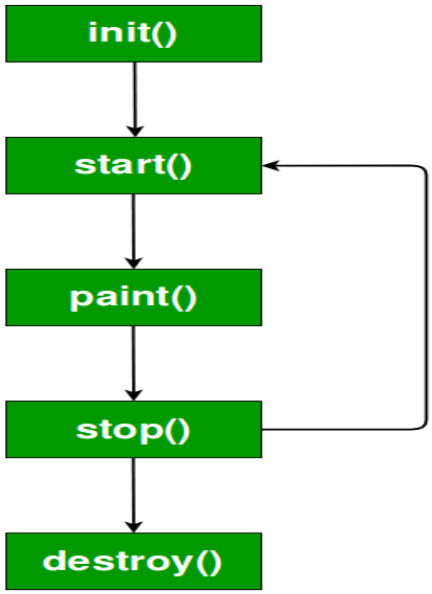
Advantage of Applet

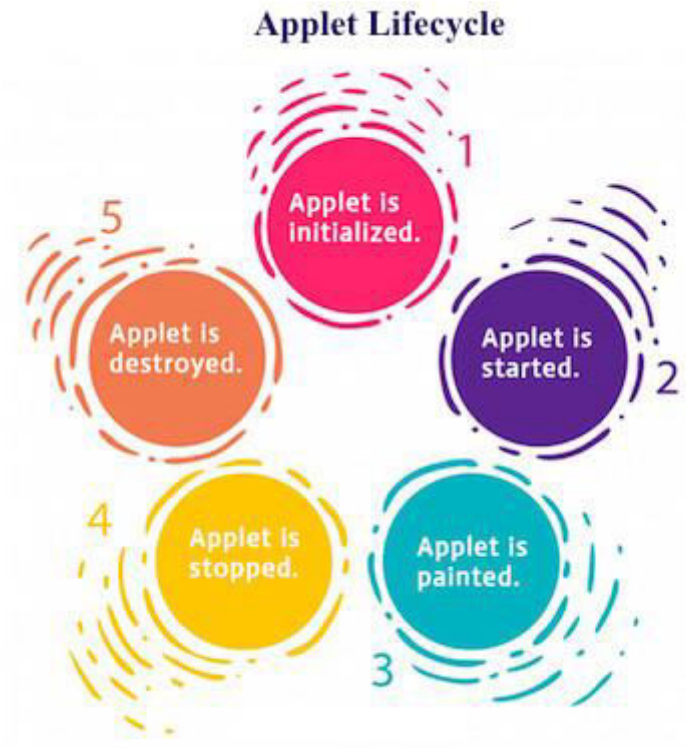
1. There are many advantages of applet. They are as follows:
2. It works at client side so less response time.
3. Secured
4. It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

1. Plugin is required at client browser to execute applet.

Life cycle of an applet :





It is important to understand the order in which the various methods shown in the above image are called. When an applet begins, the following methods are called, in this sequence:

1. init( )
2. start( )
3. paint( )

When an applet is terminated, the following sequence of method calls takes place:
1. stop( )
2. destroy( )

Let's look more closely at these methods.

1. **init( ) :** The **init( )** method is the first method to be called. This is where you should initialize variables. This method is called **only once** during the run time of your applet.
2. **start( ) :** The **start( )** method is called after **init( )**. It is also called to restart an applet after it has been stopped. Note that **init( )** is called once i.e. when the first time an applet is loaded whereas **start( )** is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.
3. **paint( ) :** The **paint( )** method is called each time an AWT-based applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored.
   paint( ) is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, **paint( )** is called.
   The **paint( )** method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.
4. **stop( ) :** The **stop( )** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When **stop( )** is called, the applet is probably running. You should use **stop( )** to suspend threads that don't need to run when the applet is not visible. You can restart them when **start( )** is called if the user returns to the page.

5. **destroy( ) :** The **destroy( )** method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The **stop( )** method is always called before **destroy( )**.

Creating Hello World applet :

Let's begin with the HelloWorld applet :

```
// A Hello World Applet

// Save file as HelloWorld.java


import java.applet.Applet;

import java.awt.Graphics;


// HelloWorld class extends Applet

public class HelloWorld extends Applet

{

    // Overriding paint() method

    @Override

    public void paint(Graphics g)

    {

        g.drawString("Hello World", 20, 20);

    }

}
```

Explanation :

1. The above java program begins with two import statements. The first import statement imports the Applet class from applet package. Every AWT-based(Abstract Window Toolkit) applet that you create must be a subclass (either directly or indirectly) of Applet class. The second statement import the Graphics class from awt package.
2. The next line in the program declares the class HelloWorld. This class must be declared as public, because it will be accessed by code that is outside the program. Inside HelloWorld, **paint( )** is declared. This method is defined by

the AWT and must be overridden by the applet.

3. Inside **paint( )** is a call to *drawString( )*, which is a member of the Graphics class. This method outputs a string beginning at the specified X,Y location. It has the following general form:
4. void drawString(String message, int x, int y)

   Here, message is the string to be output beginning at x,y. In a Java window, the upper-left corner is location 0,0. The call to *drawString( )* in the applet causes the message "Hello World" to be displayed beginning at location 20,20.

Notice that the applet does not have a **main( )** method. Unlike Java programs, applets do not begin execution at **main( )**. In fact, most applets don't even have a **main( )** method. Instead, an applet begins execution when the name of its class is passed to an applet viewer or to a network browser.

Running the HelloWorld Applet :

After you enter the source code for HelloWorld.java, compile in the same way that you have been compiling java programs(using *javac* command). However running HelloWorld with the *java* command will generate an error because it is not an application.

There are **two** standard ways in which you can run an applet :

1. Executing the applet within a Java-compatible web browser.
2. Using an applet viewer, such as the standard tool, appletviewer. An applet viewer executes your applet in a window. This is generally the fastest and easiest way to test your applet.

Each of these methods is described next.

1. **Using java enabled web browser :** To execute an applet in a web browser we have to write a short HTML text file that contains a tag that loads the applet. We can use APPLET or OBJECT tag for this purpose. Using APPLET, here is the HTML file that executes HelloWorld :

```
<applet code="HelloWorld" width=200 height=60>
</applet>
```

   The width and height statements specify the dimensions of the display area used by the applet. The APPLET tag contains several other options. After you create this html file, you can use it to execute the applet.

4. **Using appletviewer :** This is the easiest way to run an applet. To execute

HelloWorld with an applet viewer, you may also execute the HTML file shown earlier. For example, if the preceding HTML file is saved with RunHelloWorld.html,then the following command line will run HelloWorld :

5. appletviewer RunHelloWorld.html

6. **appletviewer with java source file :** If you include a comment at the head of your Java source code file that contains the APPLET tag then your code is documented with a prototype of the necessary HTML statements, and you can run your compiled applet merely by starting the applet viewer with your Java source code file. If you use this method, the HelloWorld source file looks like this :

```
// A Hello World Applet
// Save file as HelloWorld.java

import java.applet.Applet;
import java.awt.Graphics;

/*
<applet code="HelloWorld" width=200 height=60>
</applet>
*/

// HelloWorld class extends Applet
public class HelloWorld extends Applet
{
    // Overriding paint() method
    @Override
    public void paint(Graphics g)
    {
        g.drawString("Hello World", 20, 20);
    }

}
```

7. With this approach, first compile HelloWorld.java file and then simply run below command to run applet :

```
8. appletviewer HelloWorld
```

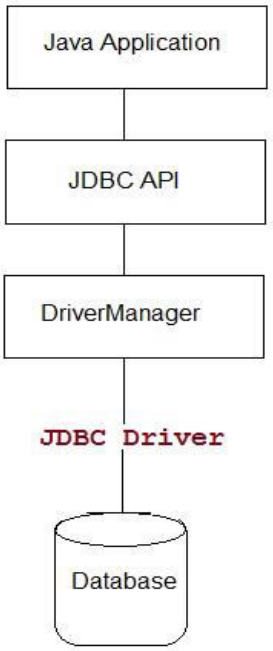
Features of Applets over HTML

- Displaying dynamic web pages of a web application.
- Playing sound files.
- Displaying documents
- Playing animations

Unit – 8

JDBC

Java Database Connectivity (JDBC) is an **Application Programming Interface(API)** used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.



JDBC 4.0

JDBC 4.0 is new and advance specification of JDBC. It provides the following advance features:-

- Connection Management
- Auto loading of Driver Interface.

- Better exception handling
- Support for large object
- Annotation in SQL query.

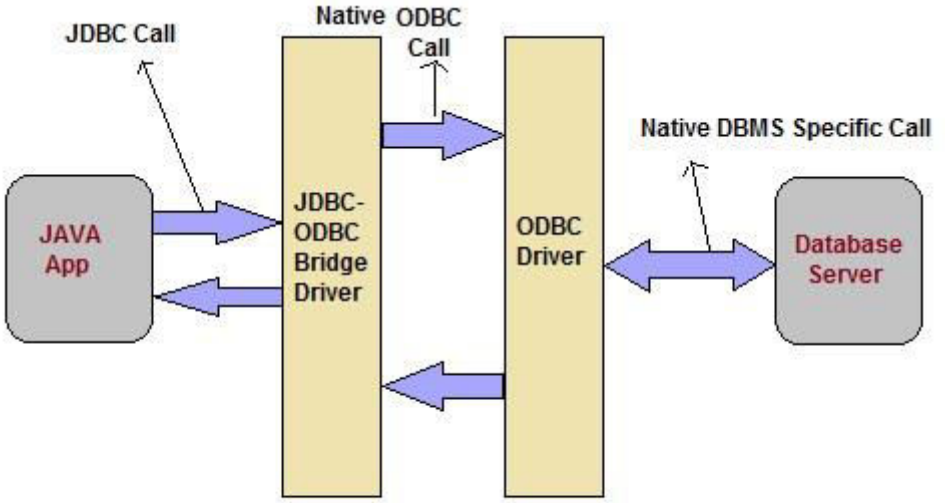
JDBC Driver

JDBC Driver is required to process SQL requests and generate result. The following are the different types of driver available in JDBC.

- **Type-1 Driver** or **JDBC-ODBC bridge**
- **Type-2 Driver** or **Native API Partly Java Driver**
- **Type-3 Driver** or **Network Protocol Driver**
- **Type-4 Driver** or **Thin Driver**

JDBC-ODBC bridge

Type-1 Driver act as a bridge between JDBC and other database connectivity mechanism(ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.



Advantages

- Easy to use
- Allow easy connectivity to all database supported by the ODBC Driver.

Disadvantage

- Slow execution time
- Dependent on ODBC Driver.
- Uses Java Native Interface(JNI) to make ODBC call.