Unit-1

Brief History of Java

Java, having been developed in 1991, is a relatively new programming language. At that time, **James Gosling** from Sun Microsystems and his team began designing the first version of Java aimed at programming home appliances which are controlled by a wide variety of computer processors.

Gosling's new language needed to be accessible by a variety of computer processors. In 1994, he realized that such a language would be ideal for use with web browsers and Java's connection to the internet began. In 1995, Netscape Incorporated released its latest version of the Netscape browser which was capable of running Java programs.

Why is it called Java? It is customary for the creator of a programming language to name the language anything he/she chooses. The original name of this language was Oak, until it was discovered that a programming language already existed that was named Oak. As the story goes, after many hours of trying to come up with a new name, the development team went out for coffee and the name Java was born.

While Java is viewed as a programming language to design applications for the Internet, it is in reality a general all purpose language which can be used independent of the Internet.

JAVA

Java is a **programming language** and a **platform**.

Java is a high level, robust, object-oriented and secure programming language.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Java Example

Let's have a quick look at Java programming example. A detailed description of hello Java example is available in next page.

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

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some

of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

JVM (JAVA VIRTUAL MACHINE) ARCHITECTURE

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment

in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent)

JVM is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

The JVM performs following operation:

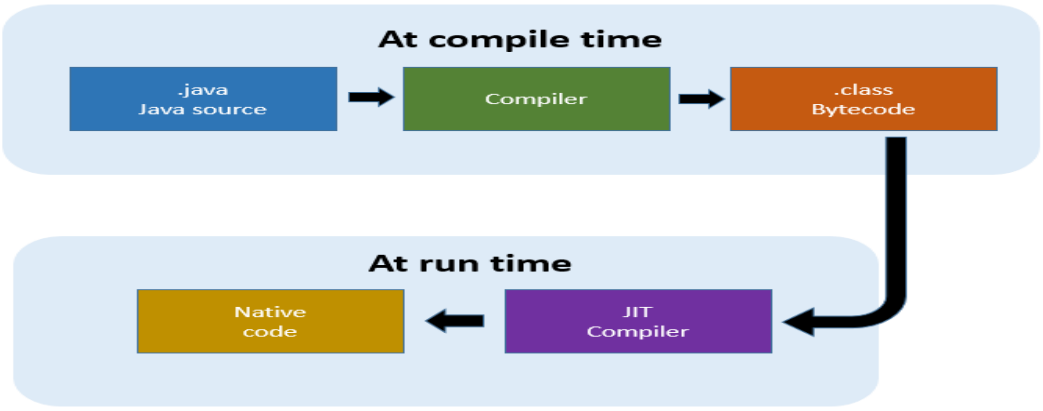
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

JAVA IN TIME COMPILER

The Just-In-Time (JIT) compiler is a component of the Java Runtime Environment that improves the performance of Java applications at run time. Java programs consists of classes, which contain platform neutral bytecode that can be interpreted by a JVM on many different computer architectures. At run time, the JVM loads the class files, determines the semantics of each individual bytecode, and performs the appropriate computation. The additional processor and memory usage during interpretation means that a Java application performs more slowly than a native application. The JIT compiler helps improve the performance of Java programs by compiling bytecode into native machine code at run time.



FEATURES OF JAVA

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java *buzzwords*.

A list of most important features of Java language is given below.

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object

2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer
- Java Programs run inside a virtual machine sandbox
- Classloader: Classloader in Java is a part of the Java Runtime Environment(JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- Bytecode Verifier: It checks the code fragments for illegal code that can violate access right to objects.
- Security Manager: It determines what resources a class can access such as reading and writing to the local disk.

Robust

Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media,

Web applications, etc.

Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++

C++ VS 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:

Java	C++
Java does not support pointers, unions, operator overloading and structure.	C++ supports pointers, unions, operator overloading structure.
Java supports garbage collection.	C++ does not supports garbage collection.
Java is platform independent.	C++ is platform dependent.
Java supports inheritance except for multiple inheritance	C++ supports inheritance including multiple inheritan
Java is interpreted.	C++ is compiled.
Java does not support destructor	C++ supports destructors.

UNIT-2

JAVA VARIABLES

A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype.

Variable is a name of memory location.

Types of Variables

There are three types of variables in java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

DATA TYPES IN JAVA

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

CONTROL FLOW STATEMENTS

Java If-else Statement

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in java.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

Java if Statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

```
if(condition){
//code to be executed
}
```

Example:

```
//Java Program to demonstate the use of if statement.
public class IfExample {
public static void main(String[] args) {
   //defining an 'age' variable
   int age=20;
   //checking the age
   if(age>18){
      System.out.print("Age is greater than 18");
   }
}
}
```

Output:

```
Age is greater than 18
```

Java if-else Statement

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```

Java if-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Java Nested if statement

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
   }
```

}

Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.

In other words, the switch statement tests the equality of a variable against multiple values.

Points to Remember

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.
- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.

Syntax:

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....

default:
 code to be executed if all cases are not matched;
}
```

LOOPS IN JAVA

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some

conditions become true. There are three types of loops in java.

- for loop
- while loop
- do-while loop

Java Simple For Loop

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Statement**: The statement of the loop is executed each time until the second condition is false.
4. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.

Syntax:

```
for(initialization;condition;incr/decr){
//statement or code to be executed
}
```

Java While Loop

The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
while(condition){
//code to be executed
}
```

Java do-while Loop

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

do{
//code to be executed
}**while**(condition);

Java Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

jump-statement;
break;

Java Continue Statement

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

jump-statement;
continue;

Java Comments

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It

can also be used to hide program code for specific time.

Types of Java Comments

There are 3 types of comments in java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

1) Java Single Line Comment

The single line comment is used to comment only one line.

Syntax:

```
//This is single line comment
```

2) Java Multi Line Comment

The multi line comment is used to comment multiple lines of code.

Syntax:

```
/*
This
is
multi line
comment
*/
```

3) Java Documentation Comment

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

Syntax:

```
/**
This
is
documentation
```

comment
*/

JAVA COMMAND LINE ARGUMENTS

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you mus from the command prompt.

```
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo
```

Output: Your first argument is: sonoo

Example of command-line argument that prints all the values

In this example, we are printing all the arguments passed from the command-line. For this purpose, v using for loop.

```
class A{
public static void main(String args[]){

for(int i=0;i<args.length;i++)
System.out.println(args[i]);

}
}
compile by > javac A.java
run by > java A sonoo jaiswal 1 3 abc
```

Output: sonoo
jaiswal

```
1
3
abc
```

JAVA ARRAY

Normally, an array is a collection of similar type of elements that have a contiguous memory location.

Java array is an object which contains elements of a similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in java is index-based, the first element of the array is stored at the 0 index.

Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax

dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];

Instantiation of an Array in Java

arrayRefVar=**new** datatype[size];

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Output:

```
10
20
70
40
50
```

Multidimensional Array in Java

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

Syntax

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

Example

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

TYPE CASTING

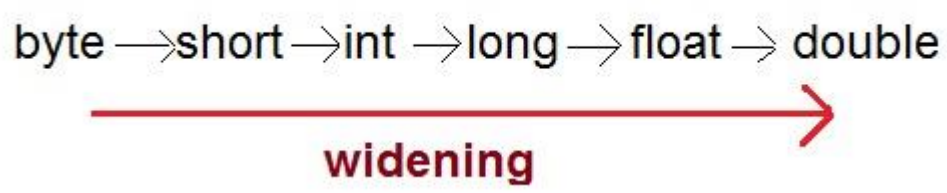
Assigning a value of one type to a variable of another type is known as **Type Casting**.

Example :

```
int x = 10;
byte y = (byte)x;
```

In Java, type casting is classified into two types,

- Widening Casting(Implicit)



Widening or Automatic type converion

Automatic Type casting take place when,

- the two types are compatible
- the target type is larger than the source type

Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

UNIT-3

CLASS

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface

Syntax to declare a class:

```
1. class <class_name>{
2.     field;
3.     method;
4. }
```

OBJECT

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- **State:**represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

INHERITANCE IN JAVA

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the relationship which is also known as a *parent-child* relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class

syntax

```
1. class Subclass-name extends Superclass-name
2. {
3.   //methods and fields
4. }
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

ENCAPSULATION IN JAVA

Encapsulation is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The Java Bean class is the example of a fully encapsulated class.

Advantage of Encapsulation in Java

By providing only a setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.

It provides you the control over the data**.** Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.

The encapsulate class is easy to test. So, it is better for unit testing.

The standard IDE's are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java.

POLYMORPHISM IN JAVA

Polymorphism in Java is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

RUNTIME POLYMORPHISM IN JAVA

Runtime polymorphism **or** Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

CONSTRUCTORS IN JAVA

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the object is created, and memory is allocated for the object.

It is a special type of method which is used to initialize the object.

When is a constructor called

Every time an object is created using new() keyword, at least one constructor is called. It calls a default constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter

Syntax of default constructor:

```
1. <class_name>(){}
```

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of obje

```
1. //Java Program to create and call a default constructor
2. class Bike1{
3. //creating a default constructor
4. Bike1(){System.out.println("Bike is created");}
5. //main method
6. public static void main(String args[]){
7. //calling a default constructor
8. Bike1 b=new Bike1();
9. }
10. }
```

Output:

```
Bike is created
```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to the distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
1. //Java Program to demonstrate the use of parameterized constructor
2. class Student4{
3.     int id;
4.     String name;
5.     //creating a parameterized constructor
6.     Student4(int i,String n){
7.     id = i;
8.     name = n;
9.     }
10.    //method to display the values
11.    void display(){System.out.println(id+" "+name);}
12.
13.    public static void main(String args[]){
14.    //creating objects and passing values
15.    Student4 s1 = new Student4(111,"Karan");
16.    Student4 s2 = new Student4(222,"Aryan");
17.    //calling method to display the values of object
18.    s1.display();
19.    s2.display();
20.   }
21. }
```

Output:

```
111 Karan
222 Aryan
```

GARBAGE COLLECTION

In Java destruction of object from memory is done automatically by the JVM. When there is no reference to an object, then that object is assumed to be no longer needed and the memory occupied by the object are released. This technique is called Garbage Collection. This is accomplished by the JVM.

Unlike C++ there is no explicit need to destroy object.

Advantages of Garbage Collection

1. Programmer doesn't need to worry about dereferencing an object.

2. It is done automatically by JVM.
3. Increases memory efficiency and decreases the chances for memory leak.

FINALIZE() METHOD

Sometime an object will need to perform some specific task before it is destroyed such as closing an open connection or releasing any resources held. To handle such situation **finalize()** method is used. **finalize()** method is called by garbage collection thread before collecting object. Its the last chance for any object to perform cleanup utility.

Signature of **finalize()** method

```
protected void finalize()
{
 //finalize-code
}
```

Some Important Points to Remember

1. finalize() method is defined in **java.lang.Object** class, therefore it is available to all the classes.
2. finalize() method is declare as **proctected** inside Object class.
3. finalize() method gets called only once by a Daemon thread named GC (Garbage Collector)thread.

gc() Method

gc() method is used to call garbage collector explicitly. However **gc()** method does not guarantee that JVM will perform the garbage collection. It only request the JVM for garbage collection. This method is present in **System** and **Runtime** class.

ACCESS MODIFIERS IN JAVA

There are two types of modifiers in java: access modifiers and non-access modifiers.

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private

2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

1) **private access modifier**

The private access modifier is accessible only within class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and p accessing these private members from outside the class, so there is compile time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

2) default access modifier

If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outsi class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
```

3) **protected access modifier**

The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
1. //save by A.java
2. package pack;
3. public class A{
4. protected void msg(){System.out.println("Hello");}
5. }
```

4) **public access modifier**

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
1. //save by A.java
2.
3. package pack;
4. public class A{
5. public void msg(){System.out.println("Hello");}
6. }
```

UNIT-4

ABSTRACT CLASS IN JAVA

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

ABSTRACT CLASS IN JAVA

A class which is declared as abstract is known as an abstract class**.** It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

ABSTRACT METHOD IN JAVA

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

abstract void printStatus();//no method body and abstract

Example of Abstract class that has an abstract method

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

```
1. abstract class Bike{
2.   abstract void run();
3. }
4. class Honda4 extends Bike{
5. void run(){System.out.println("running safely");}
6. public static void main(String args[]){
7.  Bike obj = new Honda4();
8.  obj.run();
9. }
10. }
```

Output:
running safely

INTERFACE IN JAVA

An interface in java is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also represents the IS-A relationship**.**

It cannot be instantiated just like the abstract class.

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.

- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
1. interface <interface_name>{
2.
3.     // declare constant fields
4.     // declare methods that abstract
5.     // by default.
6. }
```

Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
1. interface printable{
2. void print();
3. }
4. class A6 implements printable{
5. public void print(){System.out.println("Hello");}
6.
7. public static void main(String args[]){
8. A6 obj = new A6();
9. obj.print();
10.  }
11. }
```

Output:

```
Hello
```

MULTIPLE INHERITANCE IN JAVA BY INTERFACE

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

```
1. interface Printable{
2. void print();
3. }
4. interface Showable{
5. void show();
6. }
7. class A7 implements Printable,Showable{
8. public void print(){System.out.println("Hello");}
9. public void show(){System.out.println("Welcome");}
10.
11. public static void main(String args[]){
12. A7 obj = new A7();
13. obj.print();
14. obj.show();
15.  }
16. }
```

```
Output:Hello
       Welcome
```

JAVA PACKAGE

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Simple example of java package

The **package keyword** is used to create a package in java.

```
1. //save as Simple.java
2. package mypack;
3. public class Simple{
4.  public static void main(String args[]){
5.     System.out.println("Welcome to package");
6.    }
7. }
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

javac -d directory javafilename For **example**

javac -d . Simple.java

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

Output:Welcome to package

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represen

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

UNIT-5

Exception

Exception is an event that interrupts the normal flow of execution. It is a disruption during the execution of the Java program.

There are two types of errors:

1. Compile time errors
2. Runtime errors

Compile time errors can be again classified again into two types:

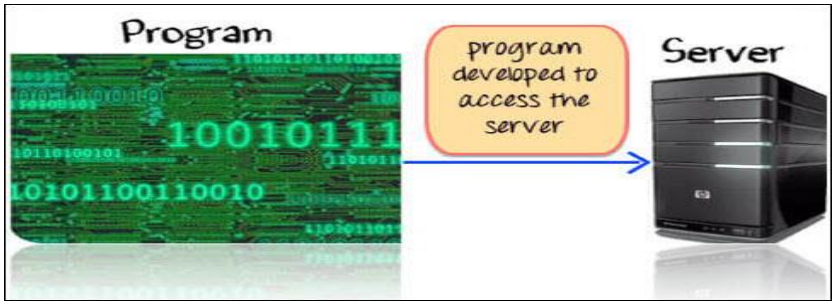
- Syntax Errors
- Semantic Errors

Syntax Errors Example:

- Instead of declaring `int a;` you mistakenly declared it as `in a;` for which compiler will throw an error.
- Example: You have declared a variable `int a;` and after some lines of code you again declare an integer as `int a;`. All these errors are highlighted when you compile the code.
- **Runtime Errors Example**
- A Runtime error is called an **Exceptions** error. It is any event that interrupts the normal flow of program execution.
- Example for exceptions are, arithmetic exception, Nullpointer exception, Divide by zero exception, etc.
- Exceptions in Java are something that is out of developers control.

Why do we need Exception?

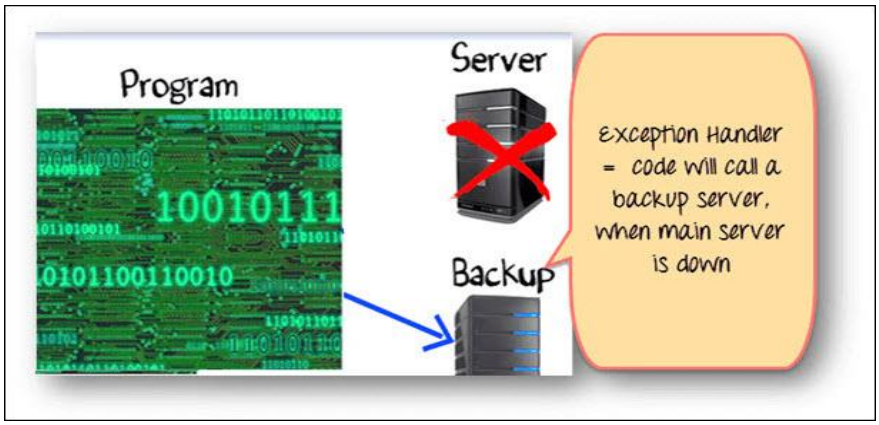
- Suppose you have coded a program to access the server. Things worked fine while you were developing the code.



How to Handle Exception

So far we have seen, exception is beyond developer's control. But blaming your code failure on environmental issues is not a solution. You need a Robust Programming, which takes care of exceptional situations. Such code is known as **Exception Handler.**

In our example, good exception handling would be, when the server is down, connect to the backup server.



To implement this, enter your code to connect to the server (Using traditional if and else conditions).

You will check if the server is down. If yes, write the code to connect to the backup server.

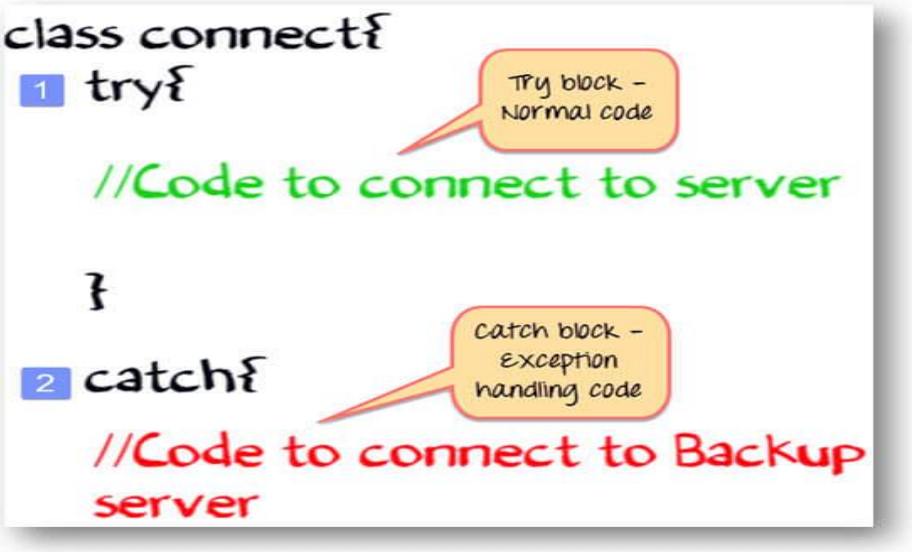
Such organization of code, using "if" and "else" loop is not effective when your code has multiple java exceptions to handle.

```
class connect{
            if(Server Up){
             // code to connect to server
            }
            else{
             // code to connect to BACKUP server
            }
}
```

Try Catch Block

Java provides an inbuilt exceptional handling.

1. The normal code goes into a **TRY** block.
2. The exception handling code goes into the **CATCH** block



In our example, TRY block will contain the code to connect to the server. CATCH block will contain the code to connect to the backup server.

In case the server is up, the code in the CATCH block will be ignored. In case the server is down, an exception is raised, and the code in catch block will be executed

So, this is how the exception is handled in Java.

Syntax for using try & catch

```
try{
    statement(s)
}
catch (exceptiontype name){
            statement(s)
}
```

Example

Step 1) Copy the following code into an editor

```
class JavaException {
   public static void main(String args[]){
      int d = 0;
      int n = 20;
      int fraction = n/d;
     System.out.println("End Of Main");
   }
}
```

Step 2) Save the file & compile the code. Run the program using command, java JavaException

Step 3) An Arithmetic Exception - divide by zero is shown as below for line # 5 and line # 6 is never executed

Step 4) Now let's see examine how try and catch will help us to handle this exception. We will put the exception causing the line of code into a **try** block, followed by a **catch** block. Copy the following code into the editor.

```
class JavaException {
 public static void main(String args[]) {
  int d = 0;
  int n = 20;
  try {
   int fraction = n / d;
   System.out.println("This line will not be Executed");
  } catch (ArithmeticException e) {
   System.out.println("In the catch Block due to Exception = " + e);
  }
  System.out.println("End Of Main");
 }
}
```

Step 5) Save, Compile & Run the code.You will get the following output

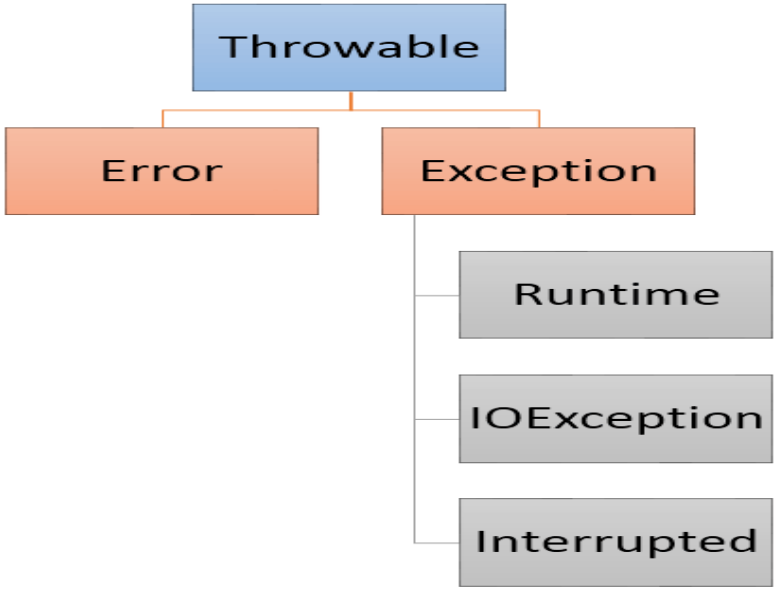
```
C:\workspace>java JavaException
In the catch clock due to Exception = java.lang.ArithmeticException: / by zero
End Of Main
```

As you observe, the exception is handled, and the last line of code is also executed. Also, note that Line #7 will not be executed because **as soon as an exception is raised the flow of control jumps to the catch block.**

Note: The AritmeticException Object "e" carries information about the exception that has occurred which can be useful in taking recovery actions.

Java Exception class Hierarchy

After one catch statement executes, the others are bypassed, and execution continues after the try/catch block. The nested catch blocks follow Exception hierarchy.



Exception Hierarchy

- All exception classes in Java extend the class 'Throwable'. Throwable has two subclasses, Error and Exception
- The Error class defines the exception or the problems that are not expected to occur under normal circumstances by our program, example Memory error, Hardware error, JVM error, etc
- The Exception class represents the exceptions that can be handled by our program, and our program can be recovered from this exception using try and catch block
- A Runtime exception is a sub-class of the exception class. The Exception of these type represents exception that occur at the run time and which cannot be tracked at the

compile time. An excellent example of same is divide by zero exception, or null pointer exception, etc

- IO exception is generated during input and output operations
- Interrupted exceptions in Java, is generated during multiple threading.

Example: To understand nesting of try and catch blocks

Step 1) Copy the following code into an editor.

```
class JavaException {
 public static void main(String args[]) {
  try {
   int d = 1;
   int n = 20;
   int fraction = n / d;
   int g[] = {
    1
   };
   g[20] = 100;
  }
  /*catch(Exception e){
            System.out.println("In the catch clock due to Exception = "+e);
  }*/
  catch (ArithmeticException e) {
   System.out.println("In the catch clock due to Exception = " + e);
  } catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("In the catch clock due to Exception = " + e);
  }
  System.out.println("End Of Main");
 }
}
```

Step 2) Save the file & compile the code. Run the program using command, **java JavaException**.

Step 3) An ArrayIndexOutOfBoundsException is generated. Change the value of int d to 0. Save, Compile & Run the code.

Step 4) An ArithmeticException must be generated.

Step 5) Uncomment line #10 to line #12. Save, Compile & Run the code.

Step 6) Compilation Error? This is because Exception is the base class of ArithmeticException Exception. Any Exception that is raised by ArithmeticException can be handled by Exception class as well.So the catch block of ArithmeticException will never get a chance to be executed which makes it redundant. Hence the compilation error.

Java Finally Block

The finally block is **executed irrespective of an exception being raised** in the try block. It is **optional** to use with a try block.

```
try {
 statement(s)
} catch (ExceptiontType name) {

 statement(s)

} finally {

 statement(s)

}
```

In case, an exception is raised in the try block, finally block is executed after the catch block is executed.

Example

Step 1) Copy the following code into an editor.

```
class JavaException {
   public static void main(String args[]){
    try{
       int d = 0;
       int n =20;
       int fraction = n/d;
    }
  catch(ArithmeticException e){
    System.out.println("In the catch clock due to Exception = "+e);
  }
  finally{
              System.out.println("Inside the finally block");
  }
}
}
```

Step 2) Save, Compile & Run the Code.

Step 3) Expected output. Finally block is executed even though an exception is raised.

Step 4) Change the value of variable d = 1. Save, Compile and Run the code and observe the output.Bottom of Form

Summary:

- An **Exception is a run-time error** which interrupts the normal flow of program execution.Disruption during the execution of the program is referred as error or exception.
- Errors are classified into two categories
    - Compile time errors – Syntax errors, Semantic errors
    - Runtime errors- Exception
- A **robust program should handle all exceptions** and continue with its normal flow of program execution. Java provides an inbuilt exceptional handling method
- Exception Handler is a set of code that **handles an exception**. Exceptions can be handled in Java using try & catch.
- **Try block**: Normal code goes on this block.
- **Catch block**: If there is error in normal code, then it will go into this block

UNIT-6

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class exte
Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Starting a thread:

start() method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.

- When the thread gets a chance to execute, its target run() method will run.

1) Java Thread Example by extending Thread class

```
1. class Multi extends Thread{
2. public void run(){
3. System.out.println("thread is running...");
4. }
5. public static void main(String args[]){
6. Multi t1=new Multi();
7. t1.start();
8.  }
9. }
```

Output:thread is running...

2) Java Thread Example by implementing Runnable interface

```
1. class Multi3 implements Runnable{
2. public void run(){
3. System.out.println("thread is running...");
4. }
5. public static void main(String args[]){
6. Multi3 m1=new Multi3();
7. Thread t1 =new Thread(m1);
8. t1.start();
9.  }
10. }
```

Output: thread is running...

If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitely create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

Thread Scheduler in Java

Thread scheduler in java is the part of the JVM that decides which thread should run.

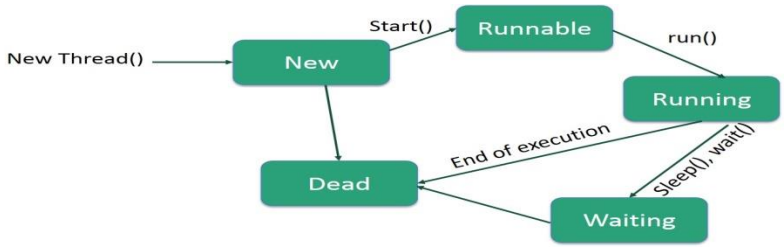
There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.

Only one thread at a time can run in a single process.

The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle −

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Synchronization in Java

Synchronization in java is the capability *to control the access of multiple threads to any shared resource.*

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
    3. static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by static synchronization

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package java.util.concurrent.locks contains several lock implementations.

Synchronized block in java

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method

Syntax to use synchronized block

```
1. synchronized (object reference expression) {
2.   //code block
3. }
```

Example of synchronized block

Let's see the simple example of synchronized block.

Program of synchronized block

```
1. class Table{
2.
3.  void printTable(int n){
4.    synchronized(this){//synchronized block
5.      for(int i=1;i<=5;i++){
6.       System.out.println(n*i);
7.       try{
8.        Thread.sleep(400);
9.       }catch(Exception e){System.out.println(e);}
10.     }
11.   }
12. }//end of the method
13. }
14.
15. class MyThread1 extends Thread{
16. Table t;
17. MyThread1(Table t){
18. this.t=t;
19. }
20. public void run(){
21. t.printTable(5);
22. }
23.
24. }
25. class MyThread2 extends Thread{
26. Table t;
27. MyThread2(Table t){
28. this.t=t;
```

```
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronizedBlock1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Output:5
10
15
20
25
100
200
300
400
500

Thread Methods

Following is the list of important methods available in the Thread class.

Sr.No.	Method & Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)**

	Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

Java is a multi-threaded programming language which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

UNIT-7

Applet

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following −

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet

Four methods in the Applet class gives you the framework on which you build any serious applet −

- **init** − This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start** − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

- **paint** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

A "Hello, World" Applet

Following is a simple applet named HelloWorldApplet.java −

```
import java.applet.*;
import java.awt.*;


public class HelloWorldApplet extends Applet {
  public void paint (Graphics g) {
    g.drawString ("Hello World", 25, 50);
  }
}
```

These import statements bring the classes into the scope of our applet class −

- java.applet.Applet
- java.awt.Graphics

Without those import statements, the Java compiler would not recognize the classes Applet and Graphics, which the applet class refers to.

The Applet Class

Every applet is an extension of the java.applet.Applet class. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following −

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip

The Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may −

- Request information about the author, version, and copyright of the applet
- Request a description of the parameters the applet recognizes
- Initialize the applet
- Destroy the applet
- Start the applet's execution
- Stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

Invoking an Applet

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.

The <applet> tag is the basis for embedding an applet in an HTML file. Following is an example that invokes the "Hello, World" applet −

```
<html>
   <title>The Hello, World Applet</title>
   <hr>
   <applet code = "HelloWorldApplet.class" width = "320" height = "120">
      If your browser was Java-enabled, a "Hello, World"
      message would appear here.
   </applet>
   <hr>
</html>
```

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5. 
6. public void paint(Graphics g){
7. g.drawString("welcome",150,150);
8. }
9. 
10. }
```

> Note: class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

```
1. <html>
2. <body>
3. <applet code="First.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5.
6. public void paint(Graphics g){
7. g.drawString("welcome to applet",150,150);
8. }
9.
10. }
11. /*
12. <applet code="First.class" width="300" height="300">
13. </applet>
14. */
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

Difference between Applet and Application in Java:

In other words, how is Java Applet different from Java Application?

The major differences between an applet and an application in JAVA are as follows:

Sr.No.	Characteristic	Java Application	Java Aapplet
1.	Definition	An application is a standalone Java program which can be run independently on client/server without the need of a web browser.	An applet is a form of Java program which is embedded with HTML page and loaded by a web server to be run on a web browser.
2.	main() method	The execution of the program starts from the main() method.	There is no requirement of main() method for the execution of the program.
3.	Access Restrictions	Application can access local disk files/ folders and network system.	Applet doesn't have access to the local network files and folders.
4.	GUI	It doesn't require any Graphical User Interface (GUI).	It must run within GUI.
5.	Security	It is a trusted application and doesn't require much security.	It requires high-security constraints as applets are

			untrusted.
6.	Environment for Execution	It requires Java Runtime Environment (JRE) for its successful execution.	It requires a web browser like Chrome, Firefox, etc for its successful execution.
7.	Installation	It is explicitly run and installed on a local system.An applet doesn't have access to local files and so it cannot perform read and write operations on files stored on local disk.	It doesn't require any explicit installation to be done.
8.	Read/ Write Operation	An application can perform read and write operations on files stored on the local disk.	An applet doesn't have access to local files and so cannot perform read and write operations on files stored on local disk.

Examples [Java Applet Vs Application]:

Here are examples of Java application and applet. You can test these programming codes by running on your system.

Java Application Code Example:

```
public class MyApplication {

   public static void main(String args[ ])

   {

      System.out.println("Hello, Welcome to csestack.org");

   }

}
```

Java Applet Code Example:

```
import java.awt.*;
import java.applet.*;

public class Myclass extends Applet  {
    public void init() { }
    public void start() { }
    public void stop() {}
    public void destroy() {}
```

```
    public void paint(Graphics g) {}
}
```

Conclusion:

Java Applications and Java Applets in perspective with Java are two varied types of programs which are different in function. But both have their own particular usage and magnitude.

What is the Java Virtual Machine (JVM)?

First of all, let us talk about the Java Virtual Machine. Before getting introduced to this term, we go through various features of Java like robustness, object-oriented, strongly typed, platform independence and so on.

Platform independence is the most important one among them, which means that once compiled, the bytecode can be executed on any machine.

But how is this achieved? How can machines with different architectures and operating systems run the same code?

This is possible because Java creates its own machine (virtual) to run on which makes it independent of other machines. This is JVM.

So what is the use of JVM?

JVM loads, verifies and executes code and gives the code a virtual machine or an environment to run on. It is a kind of interpreter. It also provides memory management and garbage collection functions.

What is the Java Run Time Environment (JRE)?

Java Run Time Environment provides everything that is needed to run a program. It mainly includes the environment to run the program along with the set of libraries or functions that are necessary for the execution of a program.
However, JRE cannot alone help us develop a program. It is physical (not virtual) and provides us with features to simply runs them.

What is the Java Development Kit (JDK)?

JDK is a kit, which means a **collection of tools or a bundle of software components to compile as well as run the program**.

It contains the environment to bytecode the program but also helps a programmer to develop applications. It consists of a compiler and debugger as well.

Java development kit is an implementation of *Java SE, Java EE* or *Java ME*. We often begin with Java SE to get a grip on the basic features of Java.

So, I hope this clarifies the difference between JRE, JDK and JVM in Java.

Difference between JVM, JRE and JDK

In case there's still a doubt, I'd share with you three simple lines that my teacher asked me to always remember which assures that I do not get confused between these terms.

They are:

JVM: Virtual Machine for platform independence

JRE: JVM + API

JDK: Compiler + JVM + API (or JRE + API)

Here, API stands for Application Programming Interface, or simply stated, a set of functionalities or a library.

Understanding these differences is very important. If you are applying for a Java developer role, in interviews, you will be asked questions on this topics.

We may also come across a term, Just in Time Compiler (JIT).

What is Just in Time Compiler (JIT Compiler)?

It is simply a module or component or a subset of JRE. It compiles some part the byte code into machine code along with working to improve the efficiency of the program. Rest all parts of the bytecode are interpreted and executed.

Note if you are installing Java on your system. When we are working with Java, we need to install only two packages, JDK and JRE. All other functionalities and tools are available in them itself.