Chapter 1
Introduction to Java

What is Java?

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

History of Java

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995. It was initially called Oak.

Features of Java

- **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 virtual Machine (JVM) on whichever platform it is being run.
- **Simple:**Java is designed to be easy to learn.
- **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architectural-neutral :**Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
- **Portable:**Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable.
- **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 do many tasks simultaneously.
- **Interpreted:**Java byte code is translated on the fly to native machine instructions and is not stored anywhere. **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.

Types of Java Program

There are two types of Java programs

1. Application Programs
2. Applet Programs

Application Programs

Application programs are stand-alone programs that are written to carry out certain tasks on local computer such as solving equations, reading and writing files etc.
The application programs can be executed using two steps

1. Compile source code to generate Byte code using Javac compiler.
2. Execute the byte code program using Java interpreter.

Applet programs:

Applets are small Java programs developed for Internet applications. An applet located in distant computer can be downloaded via Internet and executed on a local computer using Java capable browser. The Java applets can also be executed in the command line using appletviewer, which is part of the JDK.

What is JVM (Java Virtual Machine)

A Java virtual machine (JVM), an implementation of the Java Virtual Machine Specification, interprets compiled Java binary code (called bytecode) for a computer's processor (or "hardware platform") so that it can perform a Java program's instructions.

What is JIT Compiler(Just In-Time Compler)

The JIT compiler reads the bytecodes in many sections and compiles them dynamically into machine language so the program can run faster.

Java Development Environment or JDK (Java Development Kit)

Writing Java applets and applications needs development tools like JDK. The JDK includes the Java Runtime Environment, the Java compiler and the Java APIs. It's easy for both new and experienced programmers to get started. Java Development Kit (JDK) is a bundle of software components that is used to develop Java based applications**.**

These tools are the foundation of the JDK. They are the tools you use to create and build applications.

Tool Name Brief Description

appletviewer Run and debug applets without a web browser.

java The interpreter for the Java programming language.

javac The compiler for the Java programming language.

javadoc Java Documenation

javah To create C header files

javap Class file disassembler

jdb The Java Debugger.

What is Native Code?

Native code is computer programming (code) that is compiled to run with a particular processor and its set of instructions.

Differences between Java and C++

Java	**C++**
Java does not support pointers, unions, operator overloading, structures etc.	C++ supports structures, unions, operator overloading, pointers etc.
Java has built in support for threads. In Java, there is a Thread class that you inherit to create a new thread and override the run() method.	C++ has no built in support for threads.
There is no *goto* statement in Java.	C++ has *goto* statement.
Java doesn't support multiple inheritance.	C++ supports multiple inheritance.

Java has method overloading, but no operator overloading.	C++ supports both method overloading and operator overloading.
Java is interpreted for the most part and hence platform independent.	C++ generates object code and the same code may not run on different platforms.

Chapter 2
Starting The Program

Data Types in Java

Data Type means a kind of information we want to use or store in a variable.

There are two data types available in Java:

- Primitive Data Types
- Reference/Object Data Types

Primitive Data Types:
There are eight primitive data types supported by Java. Primitive data types are predefined by the language.

byte:
- Byte data type is an 8-bit signed integer.
- Minimum value is -128
- Maximum value is 127
- Default value is 0
- Example: byte a = 100 , byte b = -50

short:
- Short data type is a 16-bit signed integer.
- Minimum value is -32,768
- Maximum value is 32,767
- Default value is 0.
- Example: short s = 10000, short r = -20000

int:
- Int data type is a 32-bit signed integer.
- Minimum value is - 2,147,483,648.
- Maximum value is 2,147,483,647
- The default value is 0.
- Example: int a = 100000, int b = -200000

long:
- Long data type is a 64-bit signed integer.
- Minimum value is -9,223,372,036,854,775,808.
- Maximum value is 9,223,372,036,854,775,807
- Default value is 0L.
- Example: long a = 100000L, int b = -200000L

float:
- Float data type is a single-precision 32-bit floating point.
- Default value is 0.0f.
- Example: float f1 = 234.5f

double:
- double data type is a double-precision floating point.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- 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.
- Default value is false.
- Example: boolean one = true

char:
- char data type is a single 16-bit Unicode character.
- Char data type is used to store any character.
- Example: char letterA ='A'

Reference Data Types:
- Reference variables are created using defined constructors of the classes. They are used to access objects. For example, Employee, Student etc.
- Class objects, and various type of array variables come under reference data type.
- Default value of any reference variable is null.
- Example: Animal animal = new Animal("horse");

Variables

Variables are named locations in the memory that are used to hold or store a value.

Examples :-

```
int a, b, c;
int a = 10, b = 10;
```

There are three kinds of variables in Java:

- Local variables
- Instance variables
- Class/static variables

Local variables:

- Local variables are declared in methods, constructors, or blocks.
- Local variables are visible only within the declared method, constructor or block.

Instance variables:

- Instance variables are declared in a class, but outside a method, constructor or any block.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- The instance variables are visible for all methods, constructors and block in the class.

Class/static variables:

- Class variables also known as static variables are declared with the *static* keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.

Operators in Java

Operators are the symbols that are used to perform various operations on variables, constants and expressions. These variables or constants are called operands.

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

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:

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
++	Increment - Increases the value of operand by 1
--	Decrement - Decreases the value of operand by 1

The Relational Operators:

There are following relational operators supported by Java language

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

Operator	Description
&	Binary AND Operator copies a bit to the result if it exists in both operands.
|	Binary OR Operator copies a bit if it exists in either operand.
^	Binary XOR Operator copies the bit if it is set in one operand but not both.
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.

<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.

The Logical Operators:

The Logical Operators are used to combine or negate the expressions containing relational operators

Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.
!	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.

The Assignment Operators:

There are following assignment operators supported by Java language:

Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand

Conditional Operator (? :)

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions.

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

Control Flow Statements in Java

Java Control statements control the order of execution in a java program, based on data values and conditional logic. There are three main categories of control flow statements;

· **Selection statements:** if, if-else and switch.
· **Loop statements:** while, do-while and for.
· **Transfer statements:** break, continue, return, try-catch-finally.

Selection Statements

The If Statement

The if statement executes a block of code only if the specified expression is true. If the value is false, then the if block is skipped and execution continues with the rest of the program.
The simple if statement has the following syntax:
if (<conditional expression>)
{
    <statement>
}
Below is an example that demonstrates conditional execution based on if statement condition.

```
public class IfStatementDemo
{

        public static void main(String[] args)
        {
                int a = 10, b = 20;
                if (a > b)
                        System.out.println("a > b");
                if (a < b)
                        System.out.println("b > a");
        }
}
```

The If-else Statement

The if/else statement is an extension of the if statement. If the statements in the if statement fails, the statements in the else block are executed. You can either have a single statement or a block of code within if-else blocks.
The if-else statement has the following syntax:
if (<conditional expression>)
    <statements>
else
    <else statements>
Below is an example that demonstrates conditional execution based on if else statement condition.

```
public class IfElseStatementDemo
{

        public static void main(String[] args)
        {
                int a = 10, b = 20;
                if (a > b)
                {
                        System.out.println("a > b");
                } else
                {
                        System.out.println("b > a");
                }
        }
}
```

Switch Case Statement

The switch statement successfully tests the value of an expression against a list of integer or character constants. If a match is found, then the statements following that particular case are executed. If no match is found, then the statement following default are executed.

```
switch (<expression)
{

case value1:
        <statement 1>
          break;
case value2:
        <statement2>
          break;
…
default:
         <else statement>
         break;
}
```

Iteration Statements

While Statement

The while statement is a looping construct control statement that executes a block of code while a condition is true. The syntax of the while loop is

while (<loop condition>)
{
<statements>
}

```
public class WhileLoopDemo {

        public static void main(String[] args) {
                int count = 1;
                System.out.println("Printing Numbers from 1 to 10");
                while (count <= 10) {
                        System.out.println(count++);
                }
        }
}
```

Output

Do-while Loop Statement

The do-while loop is similar to the while loop, except that the test is performed at the end of the loop instead of at the beginning. This ensures that the loop will be executed at least once. A do-while loop begins with the keyword do, followed by the statements that make up the body of the loop. Finally, the keyword while and the test expression completes the do-while loop. The syntax of the do-while loop is

do
{
<loop body>
}while (<loop condition>);

```
public class DoWhileLoopDemo {

        public static void main(String[] args) {
                int count = 1;
                System.out.println("Printing Numbers from 1 to 10");
                do {
                        System.out.println(count++);
                } while (count <= 10);
        }
}
```

For Loops

The for loop is a looping construct which can execute a set of instructions a specified number of times. It's a counter controlled loop.
The syntax of the loop is as follows:

for (<initialization>; <loop condition>; <increment expression>)
{
<loop body>
}

Below is an example that demonstrates the looping construct namely for loop used to print numbers from 1 to 10.

```
public class ForLoopDemo {

	public static void main(String[] args) {
		System.out.println("Printing Numbers from 1 to 10");
		for (int count = 1; count <= 10; count++) {
			System.out.println(count);
		}
	}
}
```

Transfer Statements

Continue Statement

A continue statement stops the iteration of a loop and causes execution to resume at the top of the nearest enclosing loop.
The syntax of the continue statement is
continue; // the unlabeled form
continue <label>; // the labeled form

Below is a program to demonstrate the use of continue statement to print Odd Numbers between 1 to 10.

```
public class ContinueExample {

	public static void main(String[] args) {
		System.out.println("Odd Numbers");
		for (int i = 1; i <= 10; ++i) {
			if (i % 2 == 0)
				continue;
			// Rest of loop body skipped when i is even
			System.out.println(i + "\t");
		}
	}
}
```

Break Statement

The break statement transfers control out of the enclosing loop (for, while, do or switch statement).
The Syntax for break statement is as shown below;
break; // the unlabeled form
break <label>; // the labeled form

Below is a program to demonstrate the use of break statement to print numbers Numbers 1 to 10.

```
public class BreakExample {

	public static void main(String[] args) {
		System.out.println("Numbers 1 - 10");
```

```
                for (int i = 1;; ++i) {
                        if (i == 11)
                                break;
                        // Rest of loop body skipped when i is even
                        System.out.println(i + "\t");
                }
        }
}
```

Arrays

An array is a collection of data elements of same type that are referred to by the same name.

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
dataType[] arrayRefVar;   // preferred way.

or

dataType arrayRefVar[];  //  works but not preferred way.
```

Example:

```
double[] myList;         // preferred way.

or

double myList[];         //  works but not preferred way.
```

Creating Arrays:

You can create an array by using the new operator with the following syntax:

```
arrayRefVar = new dataType[arraySize];
```

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.

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 must pass at least one argument from the command prompt.

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

compile the program by **javac CommandLineExample.java**
run the program by **java CommandLineExample sonoo**
Output: Your first argument is: sonoo

Chapter 3
Java Classes and Memory Management

Object - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.

Class - A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.

Syntax :-

```
public class  <class-name>
{
        public static void main(String[] args)
        {
                //Body of Function
        }
}
```

Example :-

```
public class  xyz
{
        public static void main(String[] args)
        {
                System.out.println("Good Morning India");
        }
}
```

Features of OOPs

1) Data Abstraction :- Data Abstraction refers to the act of representing essential features without including the background details or explanations.

2) Polymorphism :- Polymorphism is made of two terms- Poly and Morph. Poly means 'Many' and Morph means 'Forms'. Polymorphism is the property by which the same message can be sent to the objects of several different classes.

3) Data Encapsulation or Data Hiding :- The wrapping up of data and functions into a single unit is called Data Encapsulation or Data Hiding.

4) Inheritance :- Inheritance is the property by which one class of things inherit or derive the properties of another class of things. A class from which another class is derived, is called **Base Class or Super Class**. A class which is derived from Base Class, is called **Sub Class or Derived Class.** Java doesn't support multiple inheritance.

Constructors

A constructor is a special member function that has the same name as that of class. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example of a constructor is given below:

```
public class Student
{
   public Student()  //Default Constructor
   {
         // This constructor has no parameter.
   }

   public Student(String name)  //Parameterized Constructor
   {
      // This constructor has one parameter, name.
   }
}
```

Creating an Object:

A class provides the blueprints for objects. So basically an object is created from a class. In Java, the new key word is used to create new objects.

There are three steps when creating an object from a class:

- **Declaration:** A variable declaration with a variable name with an object type.
- **Instantiation:** The 'new' key word is used to create the object.
- **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:

```
public class Student
{

   public Student(String name)
   {
      // This constructor has one parameter, name.
      System.out.println("Passed Name is :" + name );
   }
   public static void main(String []args){
      // Following statement would create an object myStudent
      Student myStudent = new Student( "Mukesh" );
   }
}
```

If we compile and run the above program, then it would produce the following result:

```
Passed Name is :Mukesh
```

Finalize method in java

Finalize method is a special method much like main method in java. `finalize()` is called before Garbage collector reclaim the Object, its last chance for any object to perform cleanup activity i.e. releasing any system resources held, closing connection if open etc.

Java Garbage Collection

In java, garbage means unreferenced objects.
Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

Access Specifiers or Modifiers in Java

Java Access Specifiers (also known as Visibility Specifiers) regulate access to classes, fields and methods in Java. These Specifiers determine whether a field or method in a class, can be used or invoked by another method in another class or sub-class. Access Specifiers can be used to restrict access.

Types Of Access Specifiers

In java we have four Access Specifiers and they are listed below.

1. public
2. private
3. protected
4. default(no specifier)

public specifiers

Public Specifiers achieves the highest level of accessibility. Classes, methods, and fields declared as public can be accessed from any class in the Java program, whether these classes are in the same package or in another package.

private specifiers

Private Specifiers achieves the lowest level of accessibility. private methods and fields can only be accessed within the same class to which the methods and fields belong. private methods and fields are not visible within subclasses and are not inherited by subclasses.

protected specifiers

Methods and fields declared as protected can only be accessed by the subclasses in other package or any class within the package of the protected members' class. The protected access specifier cannot be applied to class and interfaces.

default(no specifier)

When you don't set access specifier for the element, it will follow the default accessibility level. There is no default specifier keyword. using default specifier we can access class, method, or field which belongs to same package, but not from outside this package.

Chapter 4
Interfaces and Packages

Interfaces in Java

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.
An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

An interface is similar to a class in the following ways:

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The bytecode of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces:

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface:

Example:

```
/* File name : NameOfInterface.java */
import java.lang.*;
//Any number of import statements

public interface NameOfInterface
{
   //Any number of final, static fields
   //Any number of abstract method declarations\
}
```

Interfaces have the following properties:

- An interface is implicitly abstract. You do not need to use the **abstract** keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Example:

```
/* File name : Animal.java */
interface Animal
{

   public void eat();
   public void travel();
}
```

Implementing Interfaces:

When a class implements an interface, it agrees to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.
A class uses the **implements** keyword to implement an interface.

```
/* File name : MammalInt.java */
public class MammalInt implements Animal
{

   public void eat()
   {
      System.out.println("Mammal eats");
   }
```

```
   public void travel()
   {
      System.out.println("Mammal travels");
   }

   public int noOfLegs()
   {
      return 0;
   }

   public static void main(String args[]){
      MammalInt m = new MammalInt();
      m.eat();
      m.travel();
   }
}
```

This would produce the following result:

```
Mammal eats
Mammal travels
```

Packages in Java

A Package can be defined as a group of related types(classes, interfaces, enumerations etc).
Some of the existing packages in Java are::

- **java.lang** - bundles the fundamental classes
- **java.io** - classes for input , output functions are bundled in this package

Creating a package:

The **package** statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

Example:

```
/* File name : Animal.java */
package animals;

interface Animal
{
   public void eat();
   public void travel();
}
```

```
package animals;

/* File name : MammalInt.java */
public class MammalInt implements Animal
{

   public void eat()
   {
      System.out.println("Mammal eats");
   }

   public void travel()
   {
      System.out.println("Mammal travels");
   }

   public int noOfLegs()
   {
      return 0;
   }

   public static void main(String args[])
   {
      MammalInt m = new MammalInt();
```

```
      m.eat();
      m.travel();
   }
}
```

Now, you compile these two files and put them in a sub-directory called **animals** and try to run as follows:

The import Keyword:

Example:

The package can be imported using the import keyword and the wild card (*). For example:

```
import payroll.*;
```

The class itself can be imported using the import keyword. For example:

```
import payroll.Employee;
```

Chapter 5
Exception Handling and Stream Files

Exception Handling

Exception is an error event that can happen during the execution of a program and disrupts its normal flow. Java provides a robust and object oriented way to handle exception scenarios, known as **Java Exception Handling**.

Catching Exceptions

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. the syntax for using try/catch looks like the following:

```
try
{
   //Protected code
}catch(ExceptionName e1)
{
   //Catch block
}
```

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block.

The throws/throw Keywords:

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.
You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

The following method declares that it throws a RemoteException:

```
import java.io.*;
public class className
{
   public void deposit(double amount) throws RemoteException
   {
      // Method implementation
      throw new RemoteException();
   }
   //Remainder of class definition
}
```

The finally Keyword

The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.

A finally block appears at the end of the catch blocks and has the following syntax:

```
try
{
   //Protected code
}
catch(ExceptionType1 e1)
{
   //Catch block
}
catch(ExceptionType2 e2)
{
   //Catch block
}
catch(ExceptionType3 e3)
{
   //Catch block
}finally
{
   //The finally block always executes.
}
```

Creating you own Exception Class:

You can create your own exceptions in Java.

We can define our own Exception class as below:

```
class MyException extends Exception
{
}
```

You just need to extend the Exception class to create your own Exception class.

Chapter 6
Threads and Multi-threading

Multithreading

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

Thread Creation

There are two ways to create thread in java;

- Implement the Runnable interface (java.lang.Runnable)
- By Extending the Thread class (java.lang.Thread)

The Thread Control Methods

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

SN	Methods with 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.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public void interrupt()** Interrupts this thread.
6	**public final boolean isAlive()** Returns true if the thread is alive.

Life Cycle of a Thread

- **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 changes its state to the waiting state while the thread waits for another thread to perform a task.
- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time.
- **Terminated:** A runnable thread enters the terminated state when it completes its task.

Thread 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.

The synchronization is mainly used to

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

Chapter 7
Introduction to Applet, Application and JDK

Java Applet

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.

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.

Difference between Applet and Application

	Java Applets	Java Applications
1	Applets don't contain main() method	Applications contain main() method
2	Applets can be executed in a web browser	Applications can be executed at DOS prompt
3	An Applet is an Internet application	It is not internet application
4	They cannot access anything on the system except browser's services.	They can access any data or software available on the system.

Building an Applet with JDK

Java applets are like Java applications, their creation follows the same three step process of write, compile and run. The difference is, instead of running on your desktop, they run as part of a web page.

These basic steps are as follows:-

a) Write a simple applet in Java
b) Compile the Java source code
c) Create a HTML page that references the applet
d) Open the HTML page in a browser

Embedding Java Applets in HTML

The HTML <applet> tag specifies an applet. It is used for embedding a Java applet within an HTML document.

Example

```
<html>
<head>
<title>HTML applet Tag</title>
</head>
<body>
<applet code="newClass.class" width="300" height="200">
</applet>
</body>
</html>
```

Here is the *newClass.java* file:

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

public class newClass extends Applet
{
 public void paint (Graphics g)
   {
      g.drawString("Tutorialspoint.com", 300, 150);
   }
}
```

Managing Input – Output Stream

The java.io package contains every class you might need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination.

A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

Byte Streams

Java byte streams are used to perform input and output of 8-bit. The most frequently used classes are **FileInputStream** and **FileOutputStream**.

Character Streams

Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are **FileReader** and **FileWriter.**