Biyani's Think Tank

Concept based notes

Java Programming

(B.Tech)

Deepti Gupta
Lecture
Deptt. of Information Technology
Biyani Girls College, Jaipur

Biyani's Group of Girls' Colleges

Published by :
Think Tanks
Biyani Group of Colleges

Concept & Copyright :
©Biyani Shikshan Samiti
Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph : 0141-2338371, 2338591-95 • Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website :www.gurukpo.com; www.biyanicolleges.org

Edition : 2013
Price :

Leaser Type Setted by :
Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs of the students. The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self-explanatory and adopts the "Teach Yourself" style. It is based on question-answer pattern. The language of book is quite easy and understandable based on scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this Endeavour. They played an active role in coordinating the various stages of this Endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Note: **A feedback form is enclosed along with think tank. Kindly fill the feedback form and submit it at the time of submitting to books of library, else NOC from Library will not be given.**

Author

Syllabus

JAVA: Introduction to Object Orientated Programming,
Abstraction, Object Oriented Programming Principles, Features of JAVA, Introduction to Java byte code, Java Virtual machine.

PROGRAM ELEMENTS: Primitive data types, variables, as signment, arithmetic, short circuit logical operators, Arithm etic operators, bit wise operators, relational operators, Boolean logic operators, the assignment operators, operator precedence, Decision and control statements, arrays.

CONTROL STATEMENTS: Java's Selection Statements, if st atement, switch statement, Iteration Statements, while, do-while, f or, for-each, Nested Loops, Jump Statements, Using break, Using continue, return
OBJECTS AND CLASSES: Objects, constructors, returning and passing objects as parameter, Nested and inner classes, Single and Multilevel Inheritance, Extended classes, Access Control, usage of super, Overloading and overriding methods, Abstract classes, Using final with inheritance.

PACKAGE AND INTERFACES: Defining package, concept of CLASSPATH, access modifiers, importing package, Defining and implementing interfaces.
STRING HANDLING: String constructors, special string operations, character extraction, searching and comparing strings, stringBuffer class.

EXCEPTION HANDLING: Exception handling fundamentals, Exception types, uncaught exceptions, try, catch and multiple catch statements. Usage of throw, throws and finally .FILE HANDLING: I/O streams, File I/O.

CONCURRENCY: Processes and Threads, Thread Objects, Defining and Starting a Thread, Pausing Execution with Sleep, Interrupts, Joins, Synchronization. APPLET: Applet Fundamentals, using paint method and
drawing polygons.

Unit – I

Introduction To Java Programming

Q1. What is the meaning of OOP's ?

Ans OOP's stands for Object Orientated Programming.Everything in *OOP* is grouped as self sustainable "*objects*". Hence, you gain re-usability by means of four main object-oriented programming concepts.

Let's take your "*hand*" as an example. The "*hand*" is a class. Your body has two objects of type hand, named left hand and right hand. Their main functions are controlled/ managed by a set of electrical signals sent through your shoulders (through an interface). So the shoulder is an interface which your body uses to interact with your hands. The hand is a well architected class. The hand is being re-used to create the left hand and the right hand by slightly changing the properties of it.

Q2. Explain the features of JAVA ?

Ans Java is an Object Oriented Language. As a language that has the Object Oriented feature Java supports the following fundamental concepts:

- **Polymorphism**

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared the type of a reference variable cannot be changed.

- **Inheritance**

Inheritance means to take something that is already made. It is one of the most important feature of Object Oriented Programming. It is the concept that is used for reusability purpose. Inheritance is the mechanism through which we can derived classes from other classes. The derived class is called as child class or the subclass or we can say the extended class and the class from which we are deriving the subclass is called the base class or the parent class.

- **Encapsulation**

  Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding.

- **Abstraction**

  Abstraction refers to the ability to make a class abstract in OOP. An abstract class is one that cannot be instantiated. All other functionality of the class still exists, and its fields, methods, and constructors are all accessed in the same manner. You just cannot create an instance of the abstract class.

- **Classes**

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

- **Objects**

  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.

- **Method**

  A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println method, for example, the system actually executes several statements in order to display a message on the console.

Q3. What do you understand by java bytecode ?

Ans A traditional compiler compiles the source code to machine language. Java compiles for a "ficticious CPU", not for a specific CPU (processor). The compiled code is called "bytecode". To run the program, this bytecode is interpreted by the Java Virtual Machine.

Q4. What do you understand by Java Virtual Machine ?

Ans A Java virtual machine (JVM), 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. Java was designed to allow application programs to be built that could be run on any platform without having to be rewritten or recompiled by the programmer for each separate platform. A Java virtual machine makes this possible because it is aware of the specific instruction lengths and other particularities of the platform.

Q5. Explain primitive data types ?

Ans There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word. Let us now look into detail about the eight primitive data types.

byte:

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

short:

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

int:

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

- Example : int a = 100000, int b = -200000

long:

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

float:

- Float data type is a single-precision 32-bit IEEE 754 floating point.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example : float f1 = 234.5f

double:

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

boolean:

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

char:

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

- Example . char letterA ='A'

Q6. What is the meaning of variables ?

Ans It is the name that stands for a value whose value can always be change. For example, x,y, a etc. all are variables. It represents numeric value, characters, string or memory address.eg.-> x can have **x = 5**, or it can have **x= 6.5.** Every variable has a name called variable name and the type of value that a variable contain is denoted by data types. **Eg-> int x** means x contains integer value.

Q7. Explain the operators in JAVA ?

Ans Operators are the operations that are performed on one or more variables. They are of various types :-

OPERATORS	SYMBOLS	DESCRIPTION
1) **Assignment**	=	Assigns the value to the variable.
2) **Arithmetic**	–	Subtracts the right operand from the left operand
	+	Add its operands
	*	Multiplies the operands
	/	Divides the left operand by the right operands
	%	Remainder of dividing the left operand by the right operands
	++	Adds one to the operand
	––	Subtract one to the operand
3) **Relational**	>	Left operand is greater than right operand
	<	Left operand is less than right operand
	>=	Left operand is greater than or equal to right operand
	<=	Left operand is less than or equal to right operand
	==	Left operand is equal to right operand
	!=	Left operand is not equal to right operand

4) Logical	`&&`
	`||`
	`!`
		Left and Right are both true
		Either Left or Right is true
		Right is false

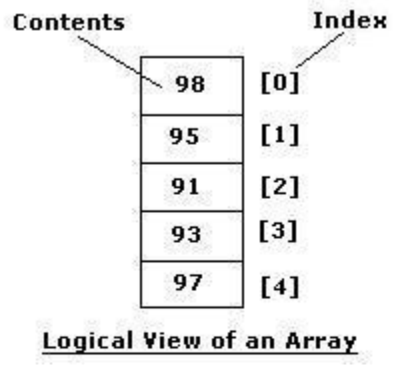
5) Bit wise	`&`
	`|`
	`^`
	`>>`
	`<<`
	`>>>`
		Bitwise **and** of the two operands
		Bitwise ***inclusive* or** of the two operands
		Bitwise ***exclusive* or** (xor) of the two operands
		Shift bits of OpLeft right by Distance bits (signed)
		Shift bits of OpLeft left by Distance bits
		Shift bits of OpLeft right by Distance bits (unsigned)
		Shift bits of OpLeft right by Distance bits with Zero fill

6) Compound Assignment	`+=`
	`- =`
	`*=`
	`/=`
	<=
	>=
		Add left side operand with right side operand and assigns the value to the left side operand
		Subtract left side operand with right side operand and assigns the value to the left side operand
		Multiply left side operand with right side operand and assigns the value to the left side operand
		Divide left side operand with right side operand and assigns the value to the left side operand
		Left operand is less than or equal to right operand
		Left operand is greater than or equal to right operand

7) Conditional	?:	

Q8. What are arrays ?

Ans When you need to store same 'type' of data that can be logically grouped together, then you can go for java array.



There are three types of array in java

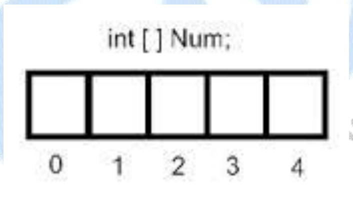
1-D Array

A **one-dimensional array** is a structured collection of components (often called *array elements*) that can be accessed individually by specifying the position of a component with a single *index* value.

Here is the syntax template of a one-dimensional array declaration:

DataType ArrayName [ConstIntExpression];

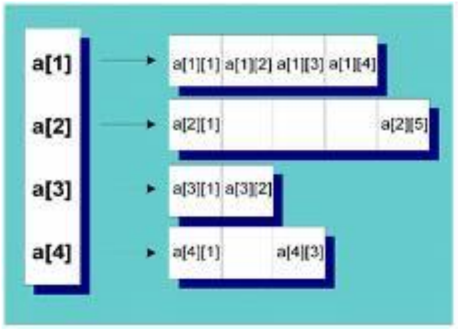
example, the declaration
int number[50];

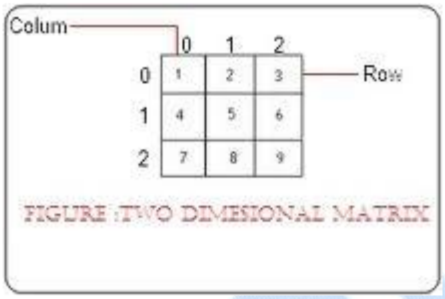


2-D Array

It denote rows and column. Like myArray has 4 rows and 4 column.

```
int[][] myArray = {  {0, 1, 2, 3},
             {3, 2, 1, 0},
             {3, 5, 6, 1},
             {3, 8, 3, 4}
           };
```


a[1]
a[2]
a[3]
a[4]
a[1][1] a[1][2] a[1][3] a[1][4]
a[2][1] a[2][5]
a[3][1] a[3][2]
a[4][1] a[4][3]


Colum
Row
0 1 2
1 2 3
4 5 6
7 8 9
FIGURE :TWO DIMESIONAL MATRIX

Unit – II

Control statements and objects and classes

Q1. What do you understand by selection statements ?

Ans There are two types of decision making statements in Java. They are:

- if statements
- switch statements

The if Statement:

An if statement consists of a Boolean expression followed by one or more statements.

Syntax:

The syntax of an if statement is:

```
if(Boolean_expression)
{
  //Statements will execute if the Boolean expression is true
}
```

If the boolean expression evaluates to true then the block of code inside the if statement will be executed. If not the first set of code after the end of the if statement(after the closing curly brace) will be executed.

Example:

```
public class Test {

 public static void main(String args[]){
  int x = 10;

  if( x < 20 ){
    System.out.print("This is if statement");
 }
 }
}
```

This would produce following result:

```
This is if statement
```

The if...else Statement:

An if statement can be followed by an optional *else* statement, which executes when the Boolean expression is false.

Syntax:

The syntax of a if...else is:

```
if(Boolean_expression){
  //Executes when the Boolean expression is true
}else{
  //Executes when the Boolean expression is false
}
```

Example:

```
public class Test {

  public static void main(String args[]){
    int x = 30;

    if( x < 20 ){
      System.out.print("This is if statement");
    }else{
      System.out.print("This is else statement");
    }
  }
}
```

This would produce following result:

```
This is else statement
```

The if...else if...else Statement:

An if statement can be followed by an optional *else if...else* statement, which is very usefull to test various conditions using single if...else if statement.

When using if , else if , else statements there are few points to keep in mind.

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of he remaining else if's or else's will be tested.

Syntax:

The syntax of a if...else is:

```
if(Boolean_expression 1){
  //Executes when the Boolean expression 1 is true
}else if(Boolean_expression 2){
  //Executes when the Boolean expression 2 is true
}else if(Boolean_expression 3){
  //Executes when the Boolean expression 3 is true
}else {
  //Executes when the none of the above condition is true.
}
```

Example:

```
public class Test {

  public static void main(String args[]){
    int x = 30;

    if( x == 10 ){
      System.out.print("Value of X is 10");
    }else if( x == 20 ){
      System.out.print("Value of X is 20");
    }else if( x == 30 ){
      System.out.print("Value of X is 30");
    }else{
      System.out.print("This is else statement");
    }
  }
}
```

This would produce following result:

```
Value of X is 30
```

Nested if...else Statement:

It is always legal to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement.

Syntax:

The syntax for a nested if...else is as follows:

```
if(Boolean_expression 1){
   //Executes when the Boolean expression 1 is true
   if(Boolean_expression 2){
      //Executes when the Boolean expression 2 is true
   }
}
```

You can nest *else if...else* in the similar way as we have nested *if* statement.

Example:

```
public class Test {

   public static void main(String args[]){
      int x = 30;
      int y = 10;

      if( x == 30 ){
         if( y == 10 ){
         System.out.print("X = 30 and Y = 10");
      }
   }
}
```

This would produce following result:

```
X = 30 and Y = 10
```

The switch Statement:

A *switch* statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

The syntax of enhanced for loop is:

```
switch(expression){
   case value :
     //Statements
     break; //optional
   case value :
     //Statements
     break; //optional
   //You can have any number of case statements.
   default : //Optional
     //Statements
}
```

Example:

```
public class Test {

  public static void main(String args[]){
    char grade = args[0].charAt(0);

    switch(grade)
    {
      case 'A' :
        System.out.println("Excellent!");
        break;
      case 'B' :
      case 'C' :
        System.out.println("Well done");
        break;
      case 'D' :
        System.out.println("You passed");
      case 'F' :
        System.out.println("Better try again");
        break;
      default :
        System.out.println("Invalid grade");
    }
    System.out.println("Your grade is " + grade);
  }
}
```

Q2. Explain loops in java ?

Ans There may be a sitution when we need to execute a block of code several number of times, and is often referred to as a loop.There are various loops:

1) For
2) While
3) Do-while

<u>The For Loop</u>

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

Syntax:

The syntax of a for loop is:

```
for(initialization; Boolean_expression; update)
{
  //Statements
}
```

Here is the flow of control in a for loop:

- The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.

- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step,then Boolean expression). After the Boolean expression is false, the for loop terminates.

Example:

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

     for(int x = 10; x < 20; x = x+1) {
        System.out.print("value of x : " + x );
        System.out.print("\n");
     }
  }
}
```

Enhanced for loop in Java:

As of java 5 the enhanced for loop was introduced. This is mainly used for Arrays.

Syntax:

The syntax of enhanced for loop is:

```
for(declaration : expression)
{
  //Statements
}
```

- **Declaration** . The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
- **Expression** . This evaluate to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Example:

```
public class Test {

  public static void main(String args[]){
    int [] numbers = {10, 20, 30, 40, 50};

    for(int x : numbers ){
      System.out.print( x );
      System.out.print(",");
    }
    System.out.print("\n");
    String [] names ={"James", "Larry", "Tom", "Lacy"};
    for( String name : names ) {
      System.out.print( name );
      System.out.print(",");
    }
  }
}
```

<u>The While Loop</u>

A while loop is a control structure that allows you to repeat a task a certain number of times.

Syntax:

The syntax of a while loop is:

```
while(Boolean_expression)
{
  //Statements
}
```

When executing, if the *boolean_expression* result is true then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Example:

```
public class Test {

  public static void main(String args[]) {
    int x = 10;

    while( x < 20 ) {
        System.out.print("value of x : " + x );
        x++;
        System.out.print("\n");
      }
   }
}
```

The do...while Loop:

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

The syntax of a do...while loop is:

```
do
{
  //Statements
}while(Boolean_expression);
```

The Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Example:

```
public class Test
{

  public static void main(String args[]){
    int x = 10;

    do{
      System.out.print("value of x : " + x );
      x++;
      System.out.print("\n");
    }while( x < 20 );
  }
}
```

Q3. Explain jump statements in java ?

Ans Java supports three jump statements:

break
continue
return

These statements transfer control to another part of your program.

The break statement

The *break* keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax:

The syntax of a break is a single statement inside any loop:

```
break;
```

Example:

```
public class Test {

  public static void main(String args[]) {
    int [] numbers = {10, 20, 30, 40, 50};

    for(int x : numbers ) {
      if( x == 30 ) {
       break;
      }
      System.out.print( x );
      System.out.print("\n");
    }
  }
}
```

This would produce following result:

```
10
20
```

Continue statement

The *continue* keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

Syntax:

The syntax of a continue is a single statement inside any loop:

```
continue;
```

Example:

```
public class Test {

  public static void main(String args[]) {
    int [] numbers = {10, 20, 30, 40, 50};

    for(int x : numbers ) {
      if( x == 30 ) {
      continue;
      }
      System.out.print( x );
      System.out.print("\n");
    }
  }
}
```

This would produce following result:

```
10
20
40
50
```

The return statement

- The last control statement is return. The return statement is used to explicitly return from a method.
- That is, it causes program control to transfer back to the caller of the method.
- the return statement immediately terminates the method in which it is executed.

The program below shows the use of return statement.

```
class Return1
{
       public static void main(String args[])
      {
            boolean t = true;
            System.out.println("Before the return.");
            if(t)
                return;       // return to caller
           System.out.println("This won't execute.");
     }
}
```

Output :

```
Before the return.
```

Q4. What do understand by objects ?

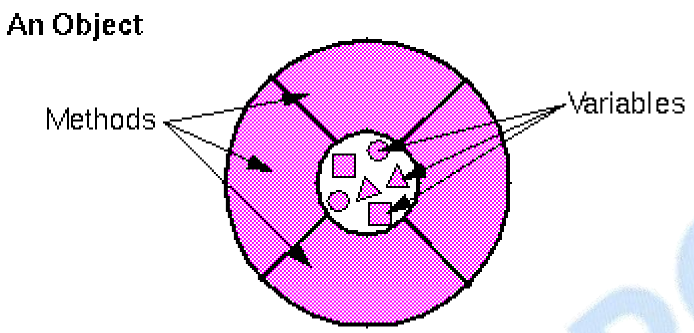
Ans If we consider the real-world we can find many objects around us, Cars, Dogs, Humans etc. All these objects have a state and behavior.

If we consider a dog then its state is - name, breed, color, and the behavior is - barking, wagging, running

If you compare the software object with a real world object, they have very similar characteristics.

Software objects also have a state and behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development methods operate on the internal state of an object and the object-to-object communication is done via methods.



Q4. What do you understand by classes ?

Ans A class is a blue print from which individual objects are created.

A sample of a class is given below:

```
public class Dog{
   String breed;
   int age;
   String color;

   void barking(){
   }

   void hungry(){
   }

   void sleeping(){
   }
}
```

A class can contain any of the following variable types.

- **Local variables .** variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables .** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded.

Q5. What are constructors ?

Ans

A **java constructor** has the same name as the name of the class to which it belongs. Constructor's syntax does not include a return type, since constructors never return a value. Java provides a default constructor which takes no arguments and performs no special actions or initializations, when no explicit constructors are provided. **Example**

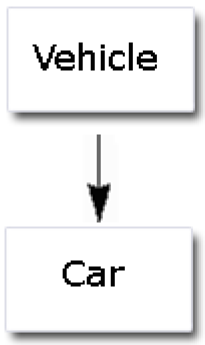
```
class Cube1
{

  int length;
  int breadth;
  int height;

         int getVolume()
        {
      return (length * breadth * height);
        }
       Cube1()
       {
      length = 10;
      breadth = 10;
      height = 10;
       }
      Cube1(int l, int b, int h)
      {
     length = l;
     breadth = b;
     height = h;
      }

     public static void main(String[] args)
    {
    Cube1 cubeObj1, cubeObj2;
    cubeObj1 = new Cube1();
    cubeObj2 = new Cube1(10, 20, 30);
System.out.println("Volume of Cube1 is : " + cubeObj1.getVolume());
System.out.println("Volume of Cube1 is : " +  cubeObj2.getVolume());

        }
}
```

Q6. Explain the concept of "Inheritance" ?

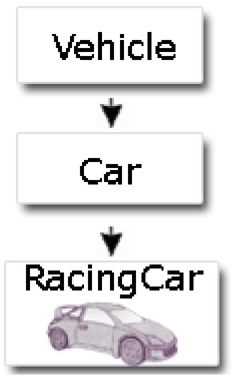
Ans Inheritance means to take something that is already made. It is one of the most important feature of Object Oriented Programming. It is the concept that is used for reusability purpose. Inheritance is the mechanism through which we can derived classes from other classes. The derived class is called as child class or the subclass or we can say the extended class and the class from which we are deriving the subclass is called the base class or the parent class.

The following kinds of inheritance are there in java.

- **Simple Inheritance**



- **Multilevel Inheritance**



Example

```
class A

{
  int x;
  int y;
  int get(int p, int q)

{
      x=p;

       y=q;

       return(0);
 }
 void Show()

{
             System.out.println(x);
 }
}

Class B extends A

{
 public static void main(String args[])

{
  A a = new A();
  a.get(5,6);
  a.Show();
}
 void display()
```

```
    {
        System.out.println(x);
         }
    }
```

Java does not support multiple Inheritance

Q7. What is the use of “super” keyword in JAVA ?
Ans

super is used to access the members of the super class.It is used for two purposes in java.

1) To access the hidden data variables of the super class hidden by the sub class.

e.g. Suppose class A is the super class that has two instance variables as int a and float b. class B is the subclass that also contains its own data members named a and b. then we can access the super class (class A) variables a and b inside the subclass class B just by calling the following command.

super.member;

Here member can either be an instance variable or a method. This form of super most useful to handle situations where the local members of a subclass hides the members of a super class having the same name. The following example clarify all the confusions.

```
class A
{
  int a;
  float b;
  void Show()
  {
     System.out.println("b in super class:  " + b);
  }

}

class B extends A
{
    int a;
    float b;
    B ( int p, float q)
   {
      a = p;
```

```
            super.b = q;
         }
         void Show(){
         super.Show();
         System.out.println("b in super class:  " + super.b);
         System.out.println("a in sub class:  " + a);
        }

        public static void main(String[] args){
        B subobj = new B(1, 5);
        subobj.Show();
        }
      }
```

Q8. What is the difference between overloading and overriding ?

Ans Overloading and overriding are the two concepts in java which have different meaning but students either consider them same or get confuse between them.

Overloading :- It is the concept in which the two or more methods have the same name but can have different number of arguments and different return type. Or we can say that the name of two or more methods are same but the signatures are different. Example,

```
   Int  add (int a, int b)
 Float add (int a, float c, int x)
```

Overriding :- It means the two or more methods have the same name and signature. Its coding body can be different but the number of argument and the return type is same. Example,

```
 Class    A
{
         Void show()
         {
         }
 }
 Class   B
 {
       Void show()
        {
       }
    }
```

Q9. What are "abstract" classes ?

Ans If a class is abstract and cannot be instantiated, the class does not have much use unless it is subclassed. This is typically how abstract classes come about during the design phase. A parent class contains the common functionality of a collection of child classes, but the parent class itself is too abstract to be used on its own.

Use the **abstract** keyword to declare a class abstract. The keyword appears in the class declaration somewhere before the class keyword.

Declaration of abstract class

An abstract class should be declared as follows :

```
   public abstract class AbstractClassDemo {
    // declare fields
    // declare non-abstract methods
    abstract void display();
}
```

Q9. What is the use of "final" Keyword in java ?

Ans In java final is a key word used in several different contexts. It is used before a variable, method, and classes so that they cannot be changed latter.

Final Variables

A final variable can only once assigned a value. This value cannot be changed latter. If final variable used in class then it must assigned a value in class constructor. Attempting to change the value of final variable/field will generate error.

Example-

```
public class FinalField {

        public final int fistField;
        public final int secondField;

        FinalField(int first, int second) {
                fistField = first;
                secondField = second;
        }

         [...]
}
```

Final Method

A final method cannot be overridden by sub class. To declare a final method put the final after the access pacifier and before the return type.

Example-

```
public class FinalField {
                public final int addNumber(){
                final int result;
                // Your code
                return result;
                }
        }
```

Final Class

A class declared final cannot be sub classed. Other classes cannot extend final class. It provides some benefit to security and thread safety.

Example-

```
final class MyFinalClass {
    public final int fistField;
    public final int secondField;

    MyFinalClass(int first, int second) {
        fistField = first;
        secondField = second;
    }
}
```

Unit – III

Package and Interfaces and String Handling

Q1. What are packages ?

Ans Package is a mechanism for organizing a group of related files in the same directory. In a computer system, we organize files into different directories according to their functionality, usability and category.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

Naming convention (Rules) for using the packages

- Package names are written in all lowercase to avoid conflict with the names of classes or interfaces.
- The directory name must be same as the name of package that is created using "**package"** keyword in the source file.
- Before running a program, the class path must be picked up till the main directory (or package) that is used in the program.
- If we are not including any package in our java source file then the source file automatically goes to the default package.
- In general, we start a package name begins with the order from top to bottom level.
- In case of the internet domain, the name of the domain is treated in reverse (prefix) order.

Q2. Explain access modifiers ?

Ans Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:

1) Default
2) Public
3) Private

4) Protected

Default Access Modifier - No keyword:

Default access modifier means we do not explicitly declare an access modifier for a class, field, method etc.
A variable or method declared without any access control modifier is available to any other class in the same package. The default modifier cannot be used for methods, fields in an interface.
Example:

Variables and methods can be declared without any modifiers, as in the following examples:

```
String version = "1.5.1";

boolean processOrder()
{
  return true;

}
```

Private Access Modifier

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.

Example:

The following class uses private access control:

```
public class Logger {
  private String format;
  public String getFormat() {
    return this.format;
  }
  public void setFormat(String format) {
    this.format = format;
  }
}
```

Public Access Modifier

A class, method, constructor, interface etc declared public can be accessed from any other class.

Example:

The following function uses public access control:

```
public static void main(String[] arguments)
 {
   // ...
 }
```

Protected Access Modifier

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

Example:

The following parent class uses protected access control, to allow its child class override *openSpeaker()* method:

```
class AudioPlayer {
   protected boolean openSpeaker(Speaker sp) {
      // implementation details
   }
}

class StreamingAudioPlayer {
   boolean openSpeaker(Speaker sp) {
      // implementation details
   }
}
```

Here if we define *openSpeaker()* method as private then it would not be accessible from any other class other than *AudioPlayer*. If we define it as public then it would become accessible to all the outside world. But our intension is to expose this method to its subclass only, thats why we used *protected* modifier.

Q4. Define interfaces and how do you implement interfaces ?

Ans 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.

Declaring Interfaces:

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

Syntax:

```
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:

A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```
/* 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 following result:

```
Mammal eats
Mammal travels
```

Q5. Explain string ?

Ans Strings, which are widely used, are a sequence of characters. In the Java programming language, strings are objects.

Creating Strings:

The most direct way to create a string is to write:

```
String greeting = "Hello world!";
```

The String class has eleven constructors that allow you to provide the initial value of the string using different sources, such as an array of characters:

```
public class StringDemo{

   public static void main(String args[]){
      char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.'};
      String helloString = new String(helloArray);
      System.out.println( helloString );
   }
}
```

This would produce following result:

```
hello
```

Q6. Explain string operations in java ?
Ans

1 **<u>char charAt(int index)</u>**
Returns the character at the specified index.

2 **<u>int compareTo(Object o)</u>**
Compares this String to another Object.

3 **<u>int compareTo(String anotherString)</u>**
Compares two strings lexicographically.

4 **<u>int compareToIgnoreCase(String str)</u>**
Compares two strings lexicographically, ignoring case differences.

5 **<u>String concat(String str)</u>**
Concatenates the specified string to the end of this string.

6 **<u>boolean contentEquals(StringBuffer sb)</u>**
Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.

7 **<u>static String copyValueOf(char[] data)</u>**
Returns a String that represents the character sequence in the array specified.

8 **static String copyValueOf(char[] data, int offset, int count)**
Returns a String that represents the character sequence in the array specified.

9 **boolean endsWith(String suffix)**
Tests if this string ends with the specified suffix.

10 **boolean equals(Object anObject)**
Compares this string to the specified object.

11 **boolean equalsIgnoreCase(String anotherString)**
Compares this String to another String, ignoring case considerations.

byte getBytes()
12 Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

byte[] getBytes(String charsetName
Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.

int lastIndexOf(String str)
Returns the index within this string of the rightmost occurrence of the specified substring.

int lastIndexOf(String str, int fromIndex)
Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

int length()
Returns the length of this string.

String replace(char oldChar, char newChar)
Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

String replaceFirst(String regex, String replacement)
Replaces the first substring of this string that matches the given regular expression with the given replacement.

boolean startsWith(String prefix, int toffset)
Tests if this string starts with the specified prefix beginning a specified index.

Unit - IV

Exception Handling Fundamentals

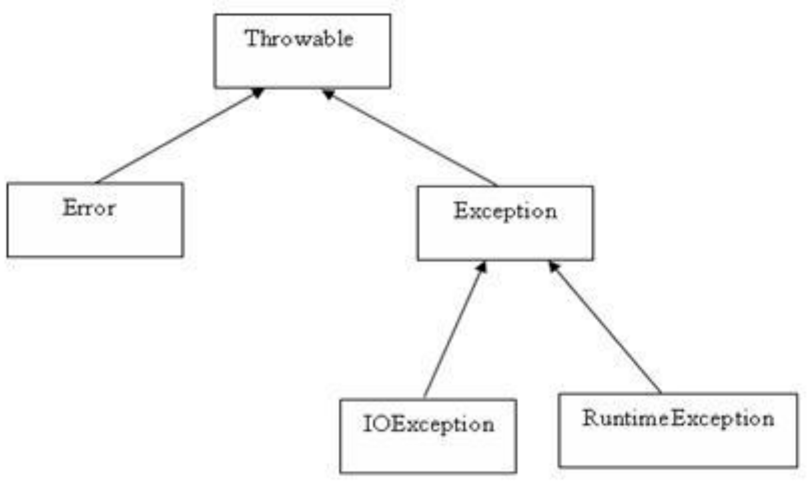
Q1. Explain the concept of exception handling ?

Ans

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications, or the JVM has run out of memory.

The Exception class has two main subclasses : IOException class and RuntimeException Class.



Q2. Explain various types of Exception ?

Ans Exception is a run-time error which arises during the execution of java program. The term exception in java stands for an exceptional event. It can be defined as abnormal event that arises during the execution and normal flow of program.

1) ArithmeticException Arithmetic error, such as divide-by-zero.
2) ArrayIndexOutOfBoundsException Array index is out-of-bounds.
3) ArrayStoreException Assignment to an array element of an incompatible type.
4) ClassCastException Invalid cast.
5) IllegalArgumentException Illegal argument used to invoke a method.
6) IllegalMonitorStateException Illegal monitor operation, such as waiting on an unlocked thread.
7) IllegalStateException Environment or application is in incorrect state.
8) IllegalThreadStateException Requested operation not compatible with current thread state.

- IndexOutOfBoundsException Some type of index is out-of-bounds.
- NegativeArraySizeException Array created with a negative size.
- NullPointerException Invalid use of a null reference.
- NumberFormatException Invalid conversion of a string to a numeric format.
- SecurityException Attempt to violate security.
- StringIndexOutOfBounds Attempt to index outside the bounds of a string.
- UnsupportedOperationException An unsupported operation was encountered

Q3. What are uncaught exception ?

Ans Uncaught exceptions are those exception which are not handled in the program.

```
class Exc2

{
public static void main(String args[])

 {
int d, a;



d = 0;
a = 42 / d;
System.out.println("This will not be printed.");
}
}
```

In this the program terminates abnormally.

Q4. Explain try, catch and multiple catch statements ?

Ans The try Block

The first step in constructing an exception handler is to enclose the code that might throw an exception within a `try` block. In general, a `try` block looks like the following:

```
try {
 code
}
Catch ()
{}
```

The catch Blocks

You associate exception handlers with a `try` block by providing one or more `catch` blocks directly after the `try` block. No code can be between the end of the `try` block and the beginning of the first `catch` block.

```
try {

} catch (ExceptionType name) {

} catch (ExceptionType name) {

}
```

Each `catch` block is an exception handler and handles the type of exception indicated by its argument. If the program has multiple catch statement , it comes under the category of multiple catch statement.

class Exc2

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

 {
int d, a;
try

{ // monitor a block of code.
d = 0;
a = 42 / d;
System.out.println("This will not be printed.");
}

catch (ArithmeticException e)

{ // catch divide-by-zero error
    System.out.println("Division by zero.");
}
    System.out.println("After catch statement.");
}
}
```

This program generates the following output:

```
Division by zero.
After catch statement.
```

Q5. Explain throw ?

Ans

The throw keyword in java programming language is used to throw an exception.
-- The throw statement in java programming language takes the object of the class java.lang.Throwable as an argument. This Throwable is propagates up the call stack till it is caught by an appropriate catch block lying among the multiple catch blocks.
-- If any method throws an exception which is not a RuntimeException, then it must be declared in advance to handle the exceptions it throws using a throws modifier on the method declaration.

```
import java.io.IOException;

public class Class1{

public method readtheFile(String file) throws IOException{

<statements>
<statements>
<statements>

if (error){
throw new IOException("error reading file");
}
}

}
```

Q6. What are "throws" ?

Ans The throws keyword in java programming language is applicable to a method to indicate that the method raises particular type of exception while being processed.
-- The throws keyword in java programming language takes arguments as a list of the objects of type java.lang.Throwables class.
-- when we use the throws with a method it is known as ducking. The method calling a method with a throws clause is needed to be enclosed within the try catch blocks.

```
public class Class1{

public method readingFile(String file) throws IOException{

<statements>

if (error){

throw new IOException("error reading file");
  }
   }
}
```

Q7. Explain the concept of "Finally" ?

Ans

The `finally` block *always* executes when the `try` block exits. This ensures that the `finally` block is executed even if an unexpected exception occurs. But `finally` is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a `return`, `continue`, or `break`

UNIT - V

THREAD

Q.1 What is process ?

Ans A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space. Processes are often seen as synonymous with programs or applications. To facilitate communication between processes, most operating systems support *Inter Process Communication* (IPC) resources, such as pipes and sockets. IPC is used not just for communication between processes on the same system, but processes on different systems.

Q.2 What is "Thread" in java ?

Ans Multithreading refers to two or more tasks executing concurrently within a single program. A thread is an independent path of execution within a program. Many threads can run concurrently within a program. Every thread in Java is created and controlled by the **java.lang.Thread class**. A Java program can have many threads, and these threads can run concurrently, either asynchronously or synchronously.

Q.3 How do you create thread in java ?

Ans

There are two ways to create thread in java;

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

<u>Implement the Runnable interface</u>

EXAMPLE

```
public class CreateThreadRunnableExample implements Runnable
{

        public void run( )
       {
             for(int i=0; i < 5; i++)
```

```
            {
                System.out.println("Child Thread : " + i);

                 Try
                {
                        Thread.sleep(50);
        }
                catch(InterruptedException ie)
                {
                        System.out.println("Child thread interrupted! " + ie);
                }
             }

           System.out.println("Child thread finished!");
        }

        public static void main(String[] args) {

            /*
              * To create new thread, use
                * Thread(Runnable thread, String threadName)
                * constructor.
                *
              */

              Thread t = new Thread(new CreateThreadRunnableExample(), "My
Thread");

               /* To start a particular thread, use void start() method of Thread class.

                 * Please note that, after creation of a thread it will not start running until
                   we call start method.
                */

             t.start();

             for(int i=0; i < 5; i++){
                    System.out.println("Main thread : " + i);

                    try{
                          Thread.sleep(100);
                    }
                  catch(InterruptedException ie){
                          System.out.println("Child thread interrupted! " + ie);
                     }
            }
            System.out.println("Main thread finished!");
      }
}
```

Typical output of this thread example would be

```
Main thread : 0
Child Thread : 0
Child Thread : 1
Main thread : 1
Main thread : 2
Child Thread : 2
Child Thread : 3
Main thread : 3
Main thread : 4
Child Thread : 4
Child thread finished!
Main thread finished!
```

Extending the Thread class

```
class XThread extends Thread
{
```

```
            XThread()
      {}
            XThread(String threadName)
      {
                        super(threadName); // Initialize thread.
                        System.out.println(this);
                        start();
            }
            public void run() {
                        //Display info about this particular thread
                        System.out.println(Thread.currentThread().getName());
            }
}

public class ThreadExample {

            public static void main(String[] args) {
                        Thread thread1 = new Thread(new XThread(), "thread1");
                        Thread thread2 = new Thread(new XThread(), "thread2");

                        //The below 2 threads are assigned default names

                        Thread thread3 = new XThread();
                        Thread thread4 = new XThread();
                        Thread thread5 = new XThread("thread5");

                        //Start the threads
                        thread1.start();
                        thread2.start();
                        thread3.start();
                        thread4.start();

                        try {

            //The sleep() method is invoked on the main thread to cause a one second delay.
                        Thread.currentThread().sleep(1000);
                        }
            catch (InterruptedException e)
            {
                  }

                        //Display info about the main thread
```

```
                    System.out.println(Thread.currentThread());
          }
}
```

Output

```
Thread[thread5,5,main]
thread1
thread5
thread2
Thread-3
Thread-2
Thread[main,5,main]
```

Q.4 Explain the life cycle of Thread ?

Ans Thread has many different state through out its life.

1 Newborn State

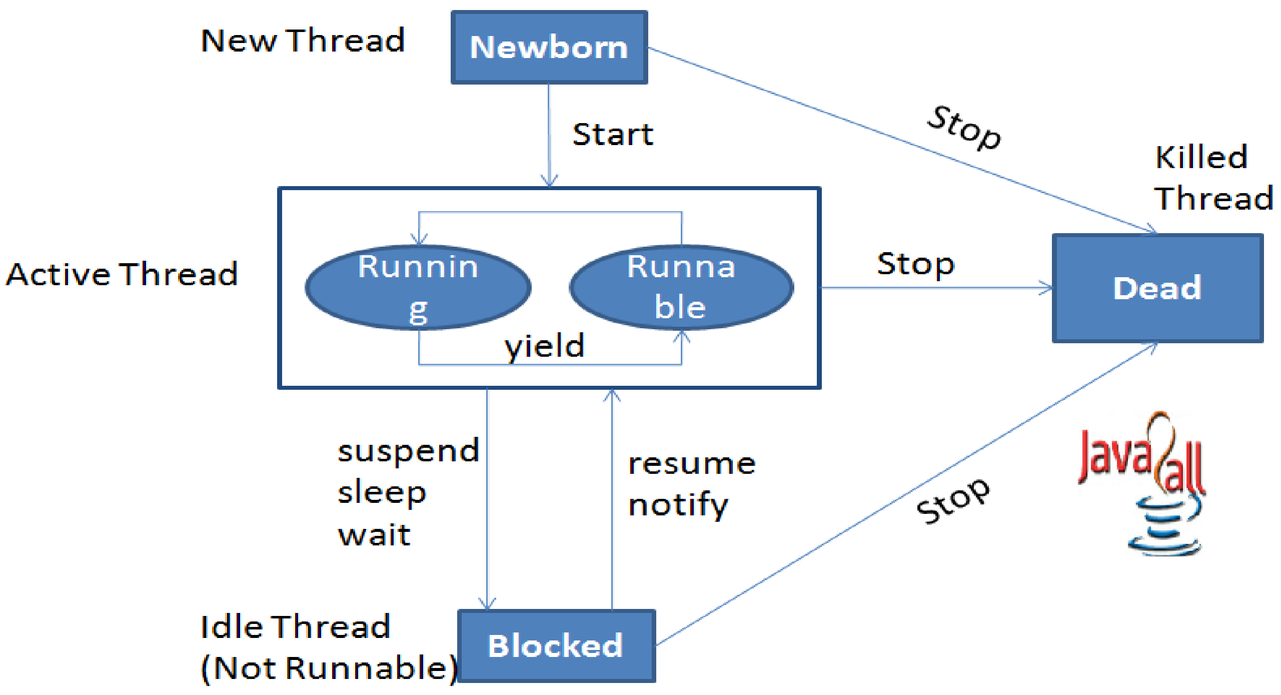
2 Runnable State

3 Running State

4 Blocked State

5 Dead State

Thread should be in any one state of above and it can be move from one state to another by different methods and ways.



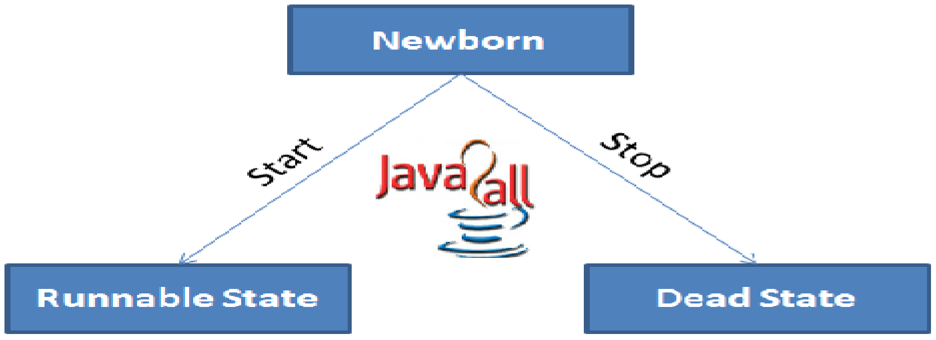
State transition diagram of a thread

1 Newborn State

When we create a thread it will be in Newborn State.

The thread is just created still its not running.

We can move it to running mode by invoking the start() method and it can be killed by using stop() method.

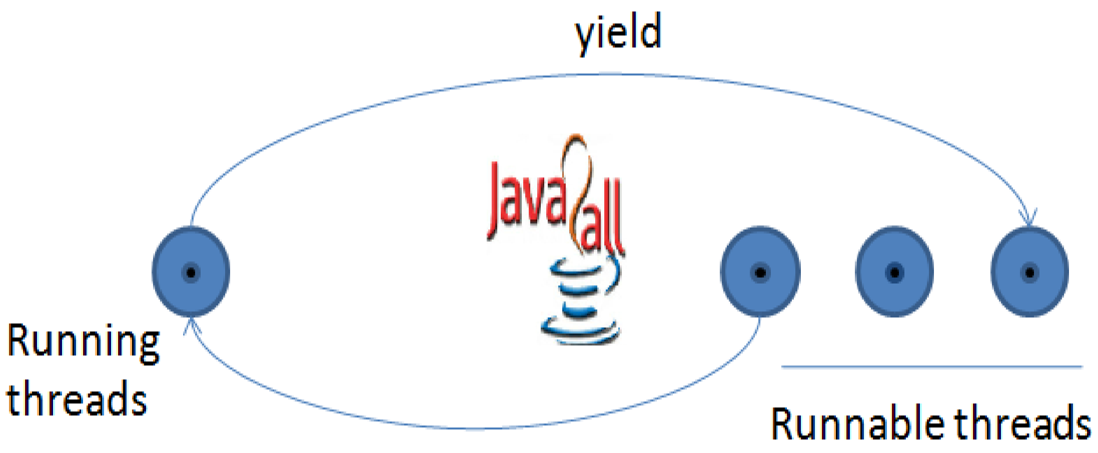


Scheduling a newborn thread

2 Runnable State

It means that thread is now ready for running and its waiting to give control.

We can move control to another thread by yield() method.

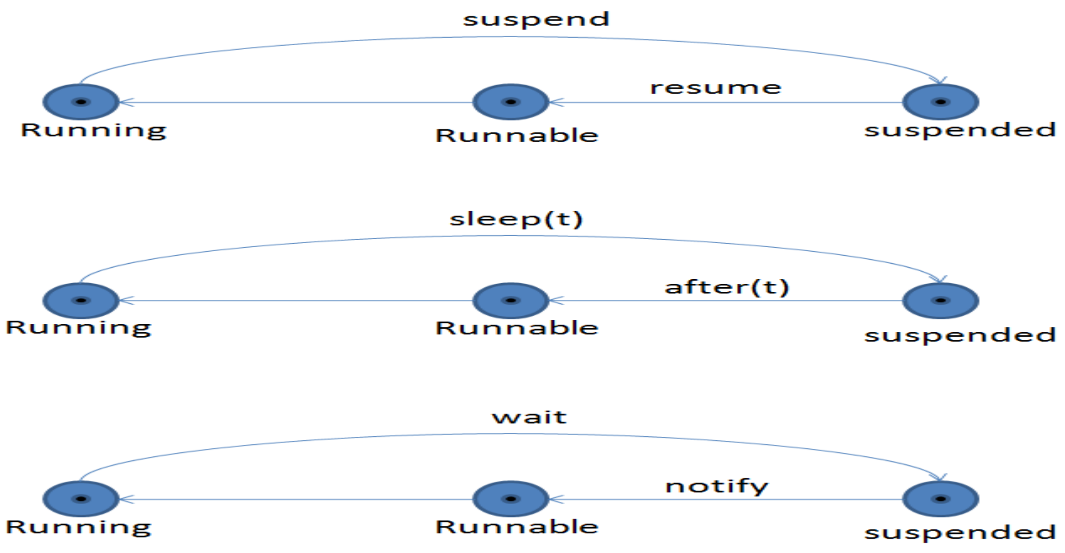


Relinquishing control using yield() method

3 Running State

It means thread is in its execution mode becaause the control of cpu is given to that particular thread.

It can be move in three different situation from running mode.



These all are different methods which can be apply on running thread and how the state is changing and how we can come in our original previous state using different methods are shown in above figure.

4 Blocked State

A thread is called in Blocked State when it is not allowed to entering in Runnable State or Running State.

It happens when thread is in waiting mode, suspended or in sleeping mode.

5 Dead State

When a thread is completed executing its run() method the life cycle of that particular thread is end.

We can kill thread by invoking stop() method for that particular thread and send it to be in Dead State

Q.5 What are APPLET ?

Ans A program designed to be executed from within another application. Unlike an application, applets cannot be executed directly from the operating system. With the growing popularity of OLE (object linking and embedding), applets are becoming more prevalent. A well-designed applet can be invoked from many different applications.

Web browsers, which are often equipped with Java virtual machines, can interpret applets from Web servers. Because applets are small in files size, cross-platform compatible, and highly secure (can't be used to access users' hard drives), they are ideal for small Internet applications accessible from a browser.

Q.6 Explain the life cycle of Applet ?

Ans This is a program, which can create graphical user interface to make visible dynamic content in the web browser. It creates a window for the web content visible inside the web browser. The compiled java program can be loaded into the web browser with the help of html document. To execute the applet in the web browser it must contain java virtual machine. The applet can deal with presentation logic and business logic. An applet does not have main method as the starting point of the execution.

Life cycle of applet

It is of five types

1. Init()
2. Start()
3. Paint()
4. Stop()
5. Destroy()

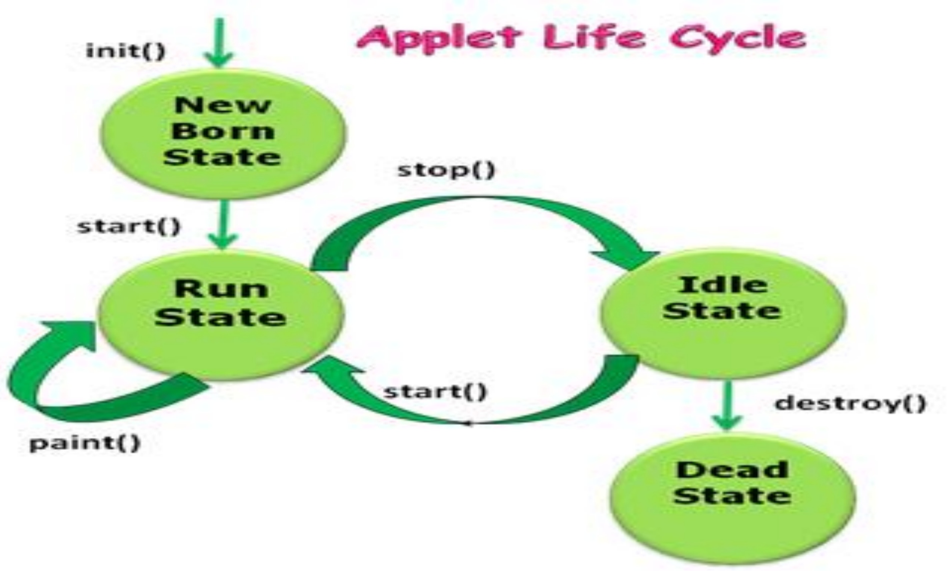
Init():- This is a method which initializes the applet with the required components inside it. This method executes only once of the life cycle of an applet.

Start():- This method is responsible for activating the applet. This method executes when applet will be restored or visible in the web browser. This method executes more than once in the life cycle of an applet.

Paint():- This method can be used to draw different components or graphical shape into the applet. This method can be executed repeatedly during the applet execution.

Stop():- When an applet will be minimized or made invisible by using back or forward button of the web browser then the applet will be deactivated by calling this method.

Destroy():- When the browser containing the applet will be closed then this method will be called. This method execute only once in the life cycle of an applet.



Q.7 How do you run applet ?
Ans

Applet viewer is a command line program to run Java applets. It is included in the SDK. The **applet viewer command** connects to the documents or resources designated by urls. It displays each applet referenced by the documents in its own window.

The syntax for the applet viewer is:

appletviewer Options URL

Where the URL specifies the location of the applet program and the Options argument specifies how to run the Java applet.

The following program shows how to build an applet and the HTML file for it. Firstly create a class. Then start the applet using **init** method. After that enter a string as str = "This is my first applet". Use **paint** method to give the dimensions of the applet. Beneath that is the HTML file which shows how to give the body for applet.

Here is the Java File:

```
import java.applet.*;
import java.awt.*;
public class Myapplet extends Applet{
  String str;
  public void init(){
          str = "This is my first applet";
  }
  public void paint(Graphics g){
          g.drawString(str, 50,50);
  }
}
```

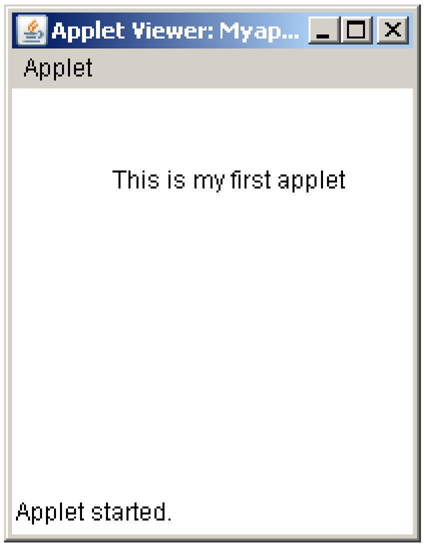
Here is the HTML File:

```
<HTML>
<BODY>
<applet code="Myapplet",height="200" width="200">
</applet>
</BODY>
</HTML>
```

After building the program, run the applet and the applet viewer as shown below.

```
C:\javac> javac Myapplet.java

C:\javac>appletviewer Myapplet.html
```

When we run the applet viewer it will display the window as shown below.



```
This is my first applet
```

```
Applet started.
```

Q.8 Give an example to draw shapes in java ?

Ans In this program we will see how to draw the different types of shapes like line, circle and rectangle. There are different types of methods for the **Graphics** class of the *java.awt.*;* package have been used to draw the appropriate shape. Explanation of the methods used in the program is given just ahead :

Graphics.drawLine() **:**
The drawLine() method has been used in the program to draw the line in the applet. Here is the syntax for the drawLine() method :

drawLine(int X_from_coordinate, int Y_from_coordinate, int X_to_coordinate, int Y_to_coordinate);

Graphics.drawString() **:**
The drawSring() method draws the given string as the parameter. Here is the syntax of the drawString() method :

drawString(String string, int X_coordinate, int Y_coordinate);

Graphics.drawOval() **:**
The drawOval() method draws the circle. Here is the syntax of the drawOval() method :

```
g.drawOval(int X_coordinate, int Y_coordinate, int Wdth, int
height);
```

Graphics.drawRect() **:**

The drawRect() method draws the rectangle. Here is the syntax of the drawRect() method :

```
g.drawRect(int X_coordinate, int Y_coordinate, int Wdth, int height)
```

Here is the java code of the program :.

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

public class  CircleLine extends Applet{
  int x=300,y=100,r=50;

  public void paint(Graphics g){
  g.drawLine(3,300,200,10);
  g.drawString("Line",100,100);
  g.drawOval(x-r,y-r,100,100);
  g.drawString("Circle",275,100);
  g.drawRect(400,50,200,100);
  g.drawString("Rectangel",450,100);
  }
}
```

Here is the HTML code of the program:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<div align="center">
<APPLET CODE="CircleLine.class" WIDTH="800" HEIGHT="500"></APPLET>
</div>
</BODY>
</HTML>
```

MCQ's

1. A good reason to use an object – oriented language is that. :

A. You can define your data type
B. program statements are simpler than in procedural languages
C. an object – oriented program can correct its own mistakes
D. it is easier to conceptualize an object-oriented program and reuse code.

2. Sharing of common information is achieved by concept of :
A. polymorphism
B. encapsulation
C. inheritance
D. none of above

3.Java was first developed in ____
A. 1990
B. 1991
C. 1993
D. 1996

4. The old name of java was ____
A. J language
B. Oak
C. Ock
D. none of above

5. full form of ADT is ____
A. Abstract Definition Type
B. Abstract Data Type
C. Abstraction Data Type
D. Abstract Data Toolkit

6. Full form of JVM is ____
A. Java Virtual Mechanism
B. Java Variable machine

C. Java Virtual Machine
D. Java variable Mechanism

7. Full form of JDK is _____
A. Java Development Kit
B. Java Definition Kit
C. Java Data Kit
D. Java Declaration Kit

8. To compile a java program ____ command is used
A. javac
B. java
C. javad
D.javadoc

9. Which of the following is a valid main() method?

A. public static void main()
B. public static void main(String s[])
C. void main(String args[])
D. none of above

10. When source file is successfully compiled , ____ is generated
A. output
B. bytecode
C. error
D. none of above

11. From these which one is the compulsory section in a java program?
A. Package statement
B. Import statement
C. Documentation section
D. Class declaration section

12. Java compiled programs have ______ extension
A .java
B .source
C .compile
D .Class

13. Which of the following is valid java comment?
A. \\ this is a comment
B. /* This is a comment */
C. /** This is a comment
D. * This is a comment *\

14. Which is the valid identifier from following
A. $rate
B. Discount%
C. Number.1
D. First-Rank

15. Which one of these lists contains only Java programming language keywords?

A. class, if, void, long, Int, continue
B. goto, instanceof, native, finally, default, throws
C. try, virtual, throw, final, volatile, transient
D. strictfp, constant, super, implements, do

16.Which of the following is a primitive data types?
A.byte
B.String
C.integer
D.Float

17.To create an array of 5 integers which statement is correct
A. int a[]=int [5]

B. int a[5]=new a[]

C. int a[]=new int [5]

D. int a[]=new a[5]

18. In java , if you do not give a value to a variable before using it ,______
A. It will contain a garbage value

B. It will initialized with zero
C. Compiler will give an error
D. None of above

19.What is the value of expression 9+8%7+6?
A. 17
B. 16
C. 13
D. 4

20. if a=10 and b=20 then a++ +--b=_____
A. 30
B. 29
C. 31
D.32

21. Relational operations returns a ____ value.
A. int
B. char
C. float
D. Boolean

22. Bitwise operators are used with ____ type of operands
A. integer
B. float
C. boolean
D. Any of these

23. 10 & 11 =___
A. 21
B.11
C. 10
D. 9

24.The instanceof operator returns a ____ value
A. Boolean
B. char
C. int
D. byte

25.The character literal represents ____ Unicode character
A. 8 bit
B. 16 bit
C. 32 bit
C. 64 bit

26. A variable defined within a block is visible _____
A. From the point of definition onwards in the program
B. From the point of definition onwards in the function
C. from the point of definition onwards in the block
D. none of above

27. From the following which is not a branching statement.
A. if
B. switch
C. while
D. if else ladder

28. In switch case if a matching case is not found___ is executed
A. else
B. break
C. default
D. none of these

29.Which of the following loop is exit- controlled loop?
A. while
B. do while
C. for
D.All of these

30. for(i=1,j=0;i<10;i++)
```
    j +=i;
  System.out.println(i);
```

A. 10

B. 11
C. 55
D.45

31. AppletViewer tool is available in which of the folder of JDK :

A. bin
B. lib
C. source
D. class

32 . The Applet class is in ……….package
A.java.applet
B.java.awt
C.java.io
D.java.util

33. To run an applet……….command is used.
A.run
B.java
C.appletviewer
D.applet

34. To display text on the applet…….method is used.
A.showString()
B.drawString()
C.println()
D.printString()

35. Which method is called first by an applet?
A.start()

B.run()
C.init()
D.paint()
36. which of the following is an optional attribute of applet tag?
A.Code

B.Name
C.Width
D.Height

37. To display text in the applet status bar………method is used.
A.showStatus()
B.showStatusBar()
C.drawStatus()
D.drawStatusBar()

38. which method can be used to draw a circle in the applet?
A.drawCircle()
B.drawOval()
C.drawArc()
D.both b and c

39. Which method can be used to draw a rectangle in the a Applet?
A.drawRect()
B.drawPolygon()
C.drawLine()
D.all of these

40. which package contains color class?
A.java.applet
B.java.awt
C.java.graphics
D.java.lang

41. Which class is used to get dimensions of an applet?
A.Dimension
B.metrics
C.Applet
D.fontMetrics

42. The constructor for Font class is……………
A.Font(String name)
B.Font(string name,int style,int size)
C.Font(String name,int size)
D.Font(String name,int style)

43. The syntax of paint()method is…………….
A. public void paint()
B. public void paint(String s)
C. public void paint(Graphics g)
D. public void paint(Graphics g,String s)

44. The syntax of drawstring() method is……………..
A. void drawstring(String s)
B. void drawstring(String s,int x,int y)
C. void drawstring(int x,int y,string s)
D. void drawstring(String x,int x)

45. The(0,0) coordinates of applet window is located at……
A. at the center of the applet
B. at the center of the right edge of the applet
C. at the center of the left edge of the applet
D. at the upper-left corner of the applet
46. The exception class is in ____ package
A. java.file
B. java.io
C. java.lang
D. java.util

47. Which keyword is used to monitor statement for exception?
A. try
B.ctach
C. throw
D. throws

48. If an exception is generated in try block , then it is caught in ____ block
A. finally
B. throw
C. throws
D. catch

49. To explicitly throw an exception , ______ keyword is used.
A. try
B.ctach

C. throw
D. throwing

50. ______ is a superclass of all exception classes.
A. Exception
B. Throwable
C. RuntimeException
D. IOException

51. Exception is subclass of ____ class
A. Exception
B. Throwable
C. RuntimeException
D. IOException

52. If a statement tries to divide by zero which exception is thrown?
A.ArithemticException
B.NullPointerException
C.NumberFormatException
D. None of these

53. When a method can throw an exception then it is specified by _____ keyword
A. finally
B. throw
C. throws
D. catch

54. Which block gets executed compulsory whether exception is caught or not.
A. finally
B. throw
C. throws
D. catch

55. To create our own exception class , we have to _______
A. Extend exception class
B. Create our own try and catch block
C. use finally block
D. Use throws keyword

56. In case of multiple catch blocks,______

A. The superclass exception must be caught first

B. A. The superclass exception can not caught first

C. Either super or subclass can be caught first.

D. None of these

57. When an array element is accessed beyond the array size , ____ exception is thrown

A. ArrayElementOutOfLimit

B. ArrayIndexOutOfBounds Exception

C. ArrayIndexOutOfBounds

D. ArrayElementOutOfBounds

58. Which method is used to print the description of the exception?

A. printStackTrace()

B. printExceptionMessage()

C. printStackMessage()

D. printExceptionTrace()

59. What is the output of following try catch block

```
try
{
   int i;
   return;
 }
 catch(Exception e)
  {
     System.out.println("Hello India");
  }
finally
  {
     System.out.println("Hello Morbi");
  }
```

A. Hello India

B. Hello India

C. Hello Morbi

D. the program will return without printing anything

60. Which of the following must be true of the object thrown by a throw statement?

A. It must be assignable to the Throwable type
B. It must be assignable to the Error type
C. It must be assignable to the Exception type
D. It must be assignable to the String type

Programs

***Program 1*:** **Add two numbers**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

class addtwonumbers
{
	public static void main(String[] args)
	{
		InputStreamReader ISR = new InputStreamReader(System.in);

		BufferedReader BR = new BufferedReader(ISR);

		// the default values for the two numbers
		int val1 = 0;
		int val2 = 0;
		try
		{

			System.out.print("Enter first number : ");

			val1 = Integer.parseInt(BR.readLine());
			System.out.print("Enter second number : ");
			val2 = Integer.parseInt(BR.readLine());
		}
		catch (Exception ex)
		{
			System.out.println(ex.toString());
		}

		System.out.println("Answer = " + (val1 + val2));
	}
}
```

Compilation

```
Enter first number : 30
Enter second number : 23
```

Output

```
Answer = 53
```

Program 2: Create a matrix

```
class MatrixExample
{
  public static void main(String[] args)
  {
     int array[][]= {{1,3,5},{2,4,6}};
     System.out.println("Row size= " + array.length);
     System.out.println("Column size = " + array[1].length);
     outputArray(array);
  }

 public static void outputArray(int[][] array)
 {
   int rowSize = array.length;
   int columnSize = array[0].length;
   for(int i = 0; i <= 1; i++)
   {
     System.out.print("[");
     for(int j = 0; j <= 2; j++)
     {
       System.out.print(" " + array[i][j]);
     }
     System.out.println(" ]");
   }
   System.out.println();
 }
}
```

Program 3: Average of Array

```
class ArrayAverage
```

```
  {
      public static void main(String[] args)
      {
         double nums[]={1.0,2.3,3.4,4.5,40.5};
         double result=0.0;
         int i=0;
         for(i=0; i < nums.length; i++)
         {
             result=result + nums[i];
         }

System.out.println("Average is =" + result/nums.length);
      }
 }
```

Program 4: Write a program to list all even numbers between two numbers

```
import java.io.*;

class AllEvenNum{
  public static void main(String[] args) {
  try{
  BufferedReader br1 = new BufferedReader(new InputStreamReade
r(System.in));
  System.out.println("Enter number : ");
  int num = Integer.parseInt(br1.readLine());
  System.out.println("Even Numbers:");
  for (int i=1;i <=num ; i++){
  if(i%2==0 ){
  System.out.print(i+",");
  }
  }
  }
  catch(Exception e){}

  }
}
```

Keyterms

Variable
The primary way programming languages store state.

Method
A way of doing something; A programmer-defined expression. Takes inputs, does actions, and then returns outputs

Identifier
The name of a variable or class used to access the value it stores or to which it refers.

Signature
A method's name and parameters; used to call the method.

Class
A blueprint for a type of object; a user-defined type of object. In the car metaphor, the engineers' blueprints and designs for the car.

Instance
A member of a class; an object. In the car metaphor, a physical, driveable car, which may differ in state and behavior from other cars.

Constructor
A special kind of method that initializes a newly created object. Its return type and name are combined, and must be the name of the class in which this special method is declared.

. (Dot operator)
The message-passing operator; can be used to utilize an object's state or behavior

Object
Something that has state and behavior. Interchangeable with "instance of a class."

abstract

The abstract keyword is used to declare a class or method to be abstract]. An abstract method has no implementation; all classes containing abstract methods must themselves be abstract, although not all abstract classes have abstract methods. Objects of a class which is abstract cannot be instantiated, but can be extended by other classes. All subclasses of an abstract class must either provide implementations for all abstract methods, or must also be abstract.

assert

The assert keyword, which was added in J2SE 1.4 is used to make an assertion—a statement which the programmer believes is always true at that point in the program. If assertions are enabled when the program is run and it turns out that an assertion is false, an AssertionError is thrown and the program terminates. This keyword is intended to aid in debugging.

boolean

The boolean keyword is used to declare a field that can store a boolean value; that is, either true or false. This keyword is also used to declare that a method returns a value of the primitive type boolean.

break

Used to resume program execution at the statement immediately following the current enclosing block or statement. If followed by a label, the program resumes execution at the statement immediately following the enclosing labeled statement or block.

byte

The byte keyword is used to declare a field that can store an 8-bit signed two's complement integer. This keyword is also used to declare that a method returns a value of the primitive type byte.

case

The case keyword is used to create individual cases in a switch statement.

catch

Defines an exception handler—a group of statements that are executed if an exception is thrown in the block defined by a preceding try keyword. The code is executed only if the class of the thrown exception is assignment compatible with the exception class declared by the catch clause.

char

The char keyword is used to declare a field that can store a 16-bit Unicode character. This keyword is also used to declare that a method returns a value of the primitive type char.

class

A type that defines the implementation of a particular kind of object. A class definition defines instance and class fields, methods, and inner classes as well as specifying the interfaces the class implements and the immediate superclass of the class. If the superclass is not explicitly specified, the superclass is implicitly Object.

const

Although reserved as a keyword in Java, const is not used and has no function. For defining constants in java, see the 'final' reserved word.

continue

Used to resume program execution at the end of the current loop body. If followed by a label, continue resumes execution at the end of the enclosing labeled loop body.

default

The default can optionally be used in a switch statement to label a block of statements to be executed if no case matches the specified value.

final

Define an entity once that cannot be changed nor derived from later. More specifically: a final class cannot be subclassed, a final method cannot be overridden, and a final variable can occur at most once as a left-hand expression. All methods in a final class are implicitly final.

finally

Used to define a block of statements for a block defined previously by the try keyword. The finally block is executed after execution exits the try block and any associated catch clauses regardless of whether an exception was thrown or caught, or execution left method in the middle of the try or catch blocks using the return keyword.

float

The float keyword is used to declare a field that can hold a 32-bit single precision IEEE 754 floating-point number.[7][8] This keyword is also used to declare that a method returns a value of the primitive type float

implements

Included in a class declaration to specify one or more interfaces that are implemented by the current class. A class inherits the types and abstract methods declared by the interfaces.

import

Used at the beginning of a source file to specify classes or entire Java packages to be referred to later without including their package names in the reference. Since J2SE 5.0, import statements can import static members of a class.

instanceof

A binary operator that takes an object reference as its first operand and a class or interface as its second operand and produces a boolean result. The instanceof operator evaluates to true if and only if the runtime type of the object is assignment compatible with the class or interface.

int

The int keyword is used to declare a field that can hold a 32-bit signed two's complement integer. This keyword is also used to declare that a method returns a value of the primitive type int.

interface

Used to declare a special type of class that only contains abstract methods, constant (static final) fields and static interfaces. It can later be implemented by classes that declare the interface with the implements keyword.

long

The long keyword is used to declare a field that can hold a 64-bit signed two's complement integer. This keyword is also used to declare that a method returns a value of the primitive type long

Bibliography

1) http://en.wikipedia.org/wiki/Java_%28programming_language%29

2) https://www.java.net

3) http://www.javaworld.com

4) https://www.ibm.com/developerworks/java

5) http://www.devx.com/Java

References:

1) Complete Reference by Herbert Schidlt
2) Java Programming by E. Balagurswamy
3) Java Programming by Genius Publication