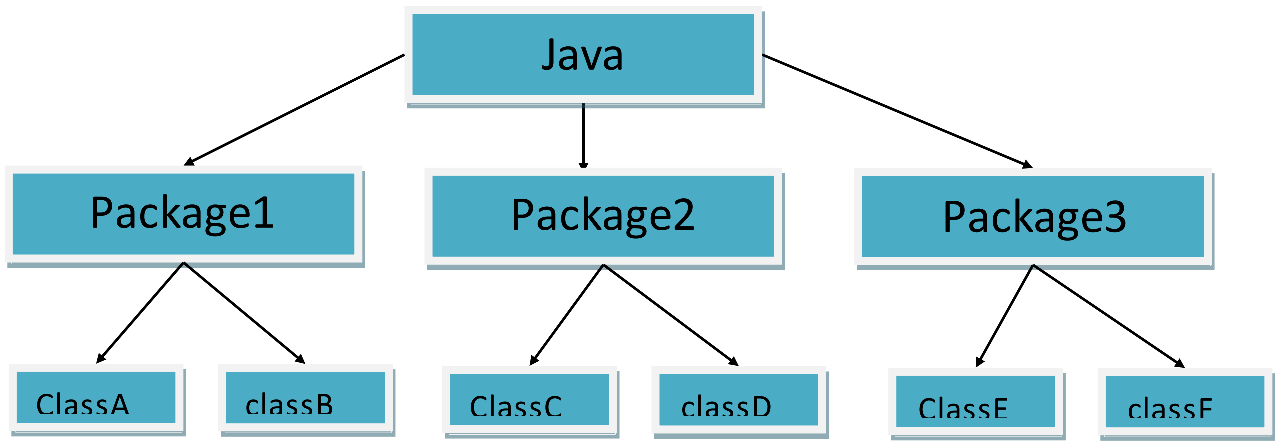
JAVA CONCEPTS

Agenda

Hierarchy



Java contains n number of packages and each package contains n number of classes. The above is the hierarchy diagram.

1. Packages in java

A Package is a group or collection of related classes which has related functionality packed into one category. For example,

1. **java.util.*;** **[1]link to java package**

All the utility things are clubbed into one group and put into one package

2. **java.io.*;**

All the input and output entities are grouped into one package.

Package are mainly used for three main reasons

1. To provide transparency between different classes.
2. To avoid same functionality to be overridden.
3. To provide a standard set of methods and functions to be used in classes.

1.1 How to Use Packages in Java Program?

Packages can be used in a program by using import statements.

Syntax

Import <space><package_name>.*;

This will import your package into your function or program.

2. Structure of a java program

//Import statement block

Import util.*;

………………,

//<access_specifiers><space><access_modifiers><space><return type><space><Main_Function><(Data_Type Parameter[])>

{

Public static void Main(String arg[])

{

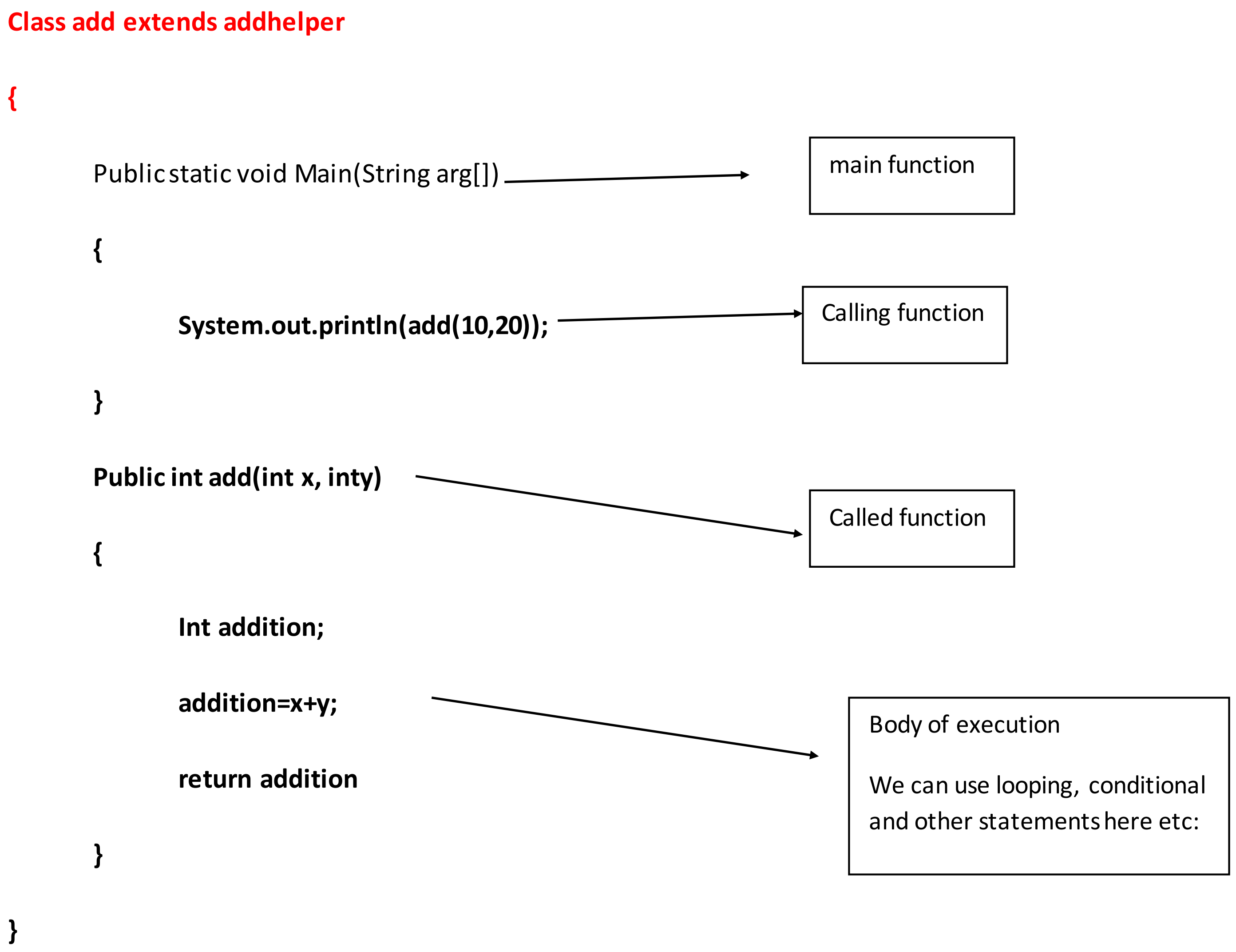
Function Body;

}

For Example.,

Consider the following program:

$$\left.\begin{array}{l}\textbf{Import java.util.*;}\\[1em]\textbf{Import java.io.*;}\\[1em]\textbf{Import java.lang.*;}\end{array}\right\}\quad\boxed{\text{Import statement}}$$



```
Class add extends addhelper

{

        Public static void Main(String arg[])

        {

                System.out.println(add(10,20));

        }

        Public int add(int x, inty)

        {

                Int addition;

                addition=x+y;

                return addition

        }

}
```

main function
Calling function
Called function
Body of execution
We can use looping, conditional and other statements here etc:

3. Class

Class means a blueprint for an object. It consists of the behaviour and property which is used to create an object. Constructor is also used to create an object. It is explained later in this document (refer to page no. 19).

3.1 How to create a class

The creation of class is as shown below in the example with methods and fields. It starts with keyword **class** followed by class_name. The class body contains methods and fields. In the following example a **bicycle class** is created.

```
class  bicycle

        {

                Int  i = 0;

                public static void str_printing(String str1)

                {

                        System.out.println("your name is"+str1);

                }

        }
```

The accessibility of a class is explained in access specifiers.

3.2 Types of classes

Different types of classes are

1. Super class
2. Subclass

3.2.1 Super Class

Super class is the head for all other sub classes which has behaviours and properties.

For example,

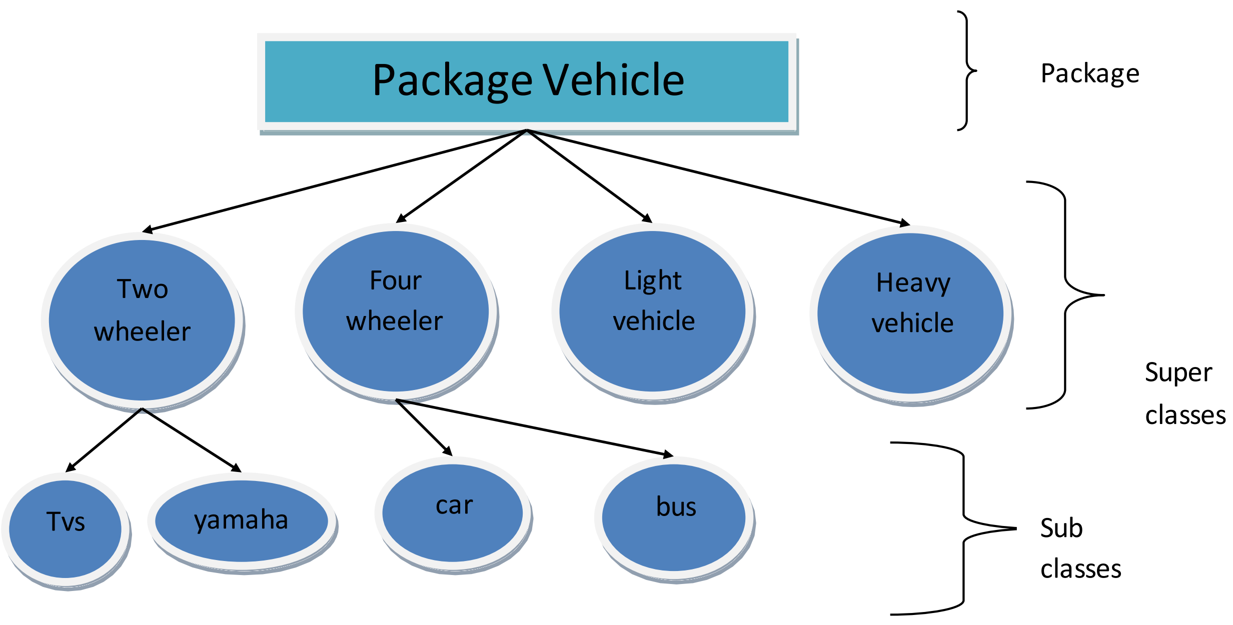
Consider two wheeler in the below example as the super class for Yamaha, TVS.

3.2.2 Sub Class

Sub Class is the child which inherits methods and behaviours of Super class. This inheritance is later explained in the concept of Inheritance.

For example,

In the Example below TVS, Yamaha are the subclass which inherits behaviour and methods of superclass Two wheeler.



4. Objects

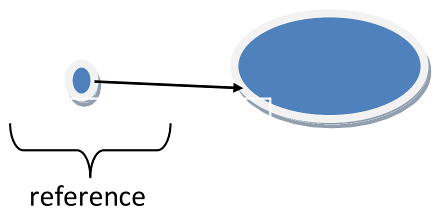
An Object is an instance of a class. An Object has only one class but a class can have many objects. Object is created from a class. Using object we can access methods related to it.

4.1 Creating instance of a class

Syntax

```
class_name  Obj_reference_name = new class_name ();
```

A reference to an object is created by the class name. We can create an instance of an object when a reference to an object is created.



From the above diagram it is shown that an object is created with a reference and the Object is referred to **Obj_reference_name.**

Now the object method is accessed by having that reference.

Obj_reference_name. Click();

The object reference accesses the click method from a class.

For example:

```
Class math

{

          public float area_circle(float radius)

          {

                    Float Area;

                    Area = 3.14* radius² ;

                    return Area;

          }

}
```

This class can be accessed by creating an instance which is shown below

```
Import  java.util.*;

Import   java.io.*;
Import  java.lang.*;

Class add extends addhelper

{

        Public static void Main(String arg[])

        {

                math area= new math(); // creating an instance of a class math

                System.out.println(area. area_circle(1.0));

        }

}
```

Output of the program is 3.14

Method of the class math is accessed by creating an instance in class add.

5. Conditional statements

There are two types of conditional statements.

1. if-else statement 2. Switch statement.

5.1 if statements

if statement is meant to control the flow of a program with a condition. The body of the statement is executed only when the condition is satisfied. If the condition is not satisfied it will skip the body of the **if** statement and will continue the remaining part of the program.

Syntax

```
if (condition)

{

        statements;
}
```

For example:

In the below condition If I want to print a particular employee pay bill statement then I pass a condition as the employee name should be equal to mulla .if the condition is satisfied then it will execute the block and prints the passed employee name pay bill else prints nothing.

```
if (State_Farm.employee_name == mulla)
{
        System.out.println ("pay bill statement of the employee"+mulla);
}
```

5.2 if-else statement

If-else statement is meant to have choice. If the condition pass then execute the statement inside if block else execute the statements inside else block. It is mainly used in Boolean data type.

Syntax

```
if (condition)
{
        Body;
}
else
{
        Body;
}
```

For example:

Consider the same example mentioned in **if** statement. If the condition is not satisfied in if block then else block is executed.

```
If (State_Farm.employee_name == mulla)

{

        System.out.println (“pay bill statement of the employee”+mulla);

}

Else

{

        System.out.println (“pay bill statement of the employee not found”);

}
```

5.3 If-else-if-else statement

if-else-if-else statement is similar to **if else** statement but else-if has got a condition similar to if block. If condition & as well as else if condition is not satisfied then it will execute the else block is executed.

Syntax

```
If (condition)

{

        Body;

}

else if (condition)

{

        Body;

}

else

{

        Body;
}
```

For example:

Consider the same example mentioned in **if** statement. If the condition in if block is satisfied then prints the pay bill of the employee. If else if condition is satisfied then prints pay bill of the employee is found. If both the condition fails then prints the else block statement as pay bill of the employee is not found in both the list.

```
if (State_Farm.employee_name == mulla)

{

        System.out.println (“pay bill of the employee”+mulla);

}

else if(Keane.employee_name == mulla)

{

                    System.out.println (“pay bill of the employee is found”+mulla);

}

else

{

        System.out.println (“pay bill of the employee is not found in both the list”);

}
```

5.4 Nested if

Nested if is a combination of multiple if ,else if and else statement. When we have a multiple condition to be satisfied to execute one statement then we go for Nested **if**. If the first condition is satisfied then it checks for the second one condition and etc... If it is passed then it execute the body or else it will execute the else statement of the particular condition.

Syntax

```
if (condition)
{
    if(condition)
    {
        Body;
    }
    else
    {
        Body;
    }
}
else
{
    Body;
}
```

For example.,

Consider the same example mentioned in **if** statement. If we want an employee PF_details to be checked then first we have to check whether the person is in the company or not, second we have to check the PF_details for the same. In case we have to verify multiple condition to execute a block of statement then nested if statement is used.

```
if (State_Farm.employee_name == mulla)
{
    System.out.println ("mulla is found in State_Farm.employee_name account . checking
the PF details");
```

```
        if( mulla.details==PF_deatils)
        {
                System.out.println (“Pf of the employee  is”+PF_deatils);
        }
        else
        {
                System.out.println (“Pf of the employee is not there. Because he is on
        probation period”);
        }
}
else
{
        System.out.println (“Employee not found”);
}
```

5.5 Switch statement

Switch statement is used when we are having large number of condition in which we want to satisfy one condition to execute our statements.

Syntax

```
switch (control_expression)
{
        case  1:
                statement;
                break;
        case 2:
                statement;
                break;
                ...
                ...
        case n:
                statement;
                break;
        default:
                statement;
    }
```

For example

```
        switch (3)
    {
        case  1:
                System.out.println("monday");
                break;
        case 2:
                System.out.println("tuesday");
                break;
        case 3:
                System.out.println("wednesday");
                break;
        case 4:
                System.out.println("thursday");
                break;
        case 5:
                System.out.println("friday");
                break;
        case 6:
                System.out.println(“saturday”);
                break;
        default:
                System.out.println(“sunday”);
        }
```

Out put

```
    Wednesday
```

If we have large number of cases then we can use switch statement.

6. Looping or iteration statements

Looping statements are used for performing multiple iteration with a set of condition. There are three types of looping statements. They are

6.1 for-loop

for loop is used when a user knows how many times the same block of statements should be repeated to complete a task. for loop has a condition known as entry controlled loop . If this condition is satisfied then the block of statements is executed until the condition is true.

Syntax

```
For (initialization ; condition ; incrementation or decrementation)
{
    Body;
}
```

For example:

Consider the same example as used in if condition. If we want to print ‘n’ number of employee’s pay bill at a time we can use for loop as shown below.

```
    For (int i =0 ; i < total_no_emp; i++)

    {

        System.out.println (“pay bill of the employee”+total_no_emp(i));
```

```
}
```

This program will print the employee's pay bill for no. of employees passed in the condition of for loop.

We can use for loop without initializing, without condition and without incrementing or decrementing in the for condition.The way a for loop could be used is as shown

```
int  i =0

for ( ; i < total_no_emp; i++) or for (; i<total_no_emp;) or for (;;)

{

        System.out.println ("pay bill of the employee"+total_no_emp(i));

}
```

The example show that the i value is initialized before the for loop.

Note: Without condition we can use for loop but it will execute infinitely.

6.2 while-loop

while loop is used when a user does not know how many times the block of statements should be repeated to complete a task. while loop has a condition known as entry controlled loop . If this condition is satisfied then the block of statements is executed.

Syntax

```
while ( condition )
{
    Body;
}
```

For example:

Consider the same example that was used for if condition. If we want to print the entire employee pay bill at a time we can use this loop.

```
while ( i < total_no_emp)

{

        int i =0;

        System.out.println ("pay bill of the employee"+total_no_emp(i));

        i++;

}
```

This program will print the employee pay bill for the no of employee that was passed in the condition.

6.3 Do while loop

do while loop is similar to while loop. The difference is the statements inside the do block is executed atleast once even if the condition is not satisfied. The condition is also called as exit controlled loop because this execute the program in the do block and then checks the condition.

Syntax

```
do
{
    Body;
} while ( condition );
```

For example:

Consider the same example of while loop. If we want to print employee pay bill atlaest once or all at a time we can use this loop.

```
do

{
        int i =0;

        System.out.println ("pay bill of the employee"+total_no_emp(i));

        i++;

    } while ( i < total_no_emp);
```

This program will print the entire employee pay bill. Executes atleast once even the condition fails.

7. Branching statements

There are three types of branching statements

7.1 Return

It is used to return control to calling function with or without a value. It terminates the execution of called function.

Syntax

```
return variable; // returns a value
void return ; // returns nothing
```

For Example

```
public final int add(final int x)

{

    Int b;

    b= x+y;

    return b;

}
```

This will return the value of 'b' to the calling function.

7.2 Continue

This is used to give the control to the calling function to continue the looping statements by skipping the current iteration and helps to execute next iteration without breaking the control of the looping statement.

Syntax

```
continue;
```

For example:

```
public final void Print_even(final int x)

{

    for (int i=0 ; i<x ; i++)

    {

        If(b== Even number)

        {

        continue ;

        }

        Else

        {

        System.out.println("the odd numbers is"+b);

        }

    }

}
```

7.3 Break

It is used to terminate the execution of looping statements.

Syntax

```
break ;
```

Example for this break statement was previously explained in switch statement please refer section 5.5

8.Boxing and Unboxing

8.1 Boxing

Boxing is the **automatic conversion** of a primitive value to a corresponding wrapper instance.

For example:

Integer x = 9; // Here 9 is a primitive value and x is a wrapper class instance of Integer.

8.2 **Unboxing**

Unboxing is the automatic conversion of a wrapper instance to a primitive type.

For example:

Integer x = 9;

int y = x; // Here x is the wrapper class instance and y is the primitive data type.

9. Constructors[8]

Constructors are used for creating an object. The rule for creating a constructor is it should have the **same class name** for which an instance is created. Constructor is also used in final fields to initialize the value. When we declare a function without constructors in a class, if we want to create a constructor for any object it uses default constructor.

Syntax

```
class class_name
{
        class_name()
        {
                Body;
```

```
          }
}
```

For example:

```
          Class math
          {
                    math ()
                    {
                              Int x;
                              Int y;
                    }
          }
```

This will create a math object when a constructor is called.

You can call a constructor the following ways

```
          new math();
          math m = new math();
```

We can also overload constructors like

```
Class math
          {
                    Void math()
                    {
                              Int x;
                              Int y;
                    }
```

```
        Void math(int x)

        {

                Int x;

                Int y;

        }

    }
```

10.Over loading

Overloading is done for a methods and constructors. Overloading is having the same method name with different functionality. Overloading can be done in two ways:

1. With same set of different parameters.

- With different set of parameters.

10.1 Overloading with same set of different parameters

For example, consider a program that a mathematician created a formula to find area of a circle. If I want to use the same method with same set but different parameters we can do this type of overloading the following way:

For Example.,

A class math has area method. Consider sub class is inheriting the super class for change in parameters.

```
Class math

{

    public float area_circle(float radius)

    {
```

```
                Float  Area;

                Area = 3.14*  radius$^2$ ;

                return  Area;

        }

}

Class new_method extends math       // sub class

{

        public  float  area_circle(int  radius)   // overloading with different parameter with same method name

        {

                Float  Area;

                  Area = 3.14*  radius$^2$;

                return  Area;

        }

}
```

In the above example a user has overloaded a float parameter to a int parameter as he requires int value for his calculation.

10.2 Overloading with different number of parameters

For example, consider a program that a mathematician created a formula to find area of a circle. If I want to use the same method name with different number of parameters we can do this type of overloading the following way:

For Example:

A class math has area method. Consider a sub class is inheriting the super class for changing the parameters. User has altered the formula as shown in the example:

```
Class math

{

        public float area_circle(float radius)

        {

                Float Area;

                Area = 3.14* radius2 ;

                return Area;

        }

}

Class new_method extends math      // sub class

{

        public float area_circle(float radius, float diameter)        // overloading method

        {

                Float Area;

                Area = 3.14* radius2*diameter;

                return Area;

        }

}
```

In the above example a user has changed the formula according to his requirement. So he has overloaded the method for his convenience with increase in number of parameter.

11. Overriding

Overriding concept is used to override methods with same method name and same set of parameter. The only thing that is editable is the body of the method.

For Example:

A class math has area method. Consider sub class is inheriting the super class with out change in method name and parameter.

```
Class math

{

    public float  area_circle(float radius)

    {

         Area = 3.14* radius² ;

        return  Area;

    }

}

Class new_method extends math      // sub class

{

    public  float  area_circle(float radius)          // overriding method

    {

        Float  Area;

        Area = 3.14* radius²*8;

        return  Area;
        }
}
```

In the above example a user has changed the formula according to his application. So he has overridden the method for his convenience without change in parameters.

12. Inheritance

Inheritance is defined as "is" a relation concept. Inheritance is nothing but inheriting the properties and behaviour of an object from its super class. Multiple inheritance is not supported in java.

Syntax

First we should have a super class

```
class class_name
{
        Body;
}
class class_name1 extends class_name
        {
                Inheriting property and methods;
        }
```

For example:

```
Class math
{
        public float area_circle(float radius)
        {
               Area = 3.14* radius$^2$ ;
             return  Area;
        }
}
Class new_method extends math      // sub class
{
        public float area_circle(float radius)        // overriding method
        {
             Float  Area;
             Area = 3.14* radius$^2$*8;
             return  Area;
        }
}
```

This is inheriting area method from a super class.

13. Access Modifiers or Access Specifiers[7]

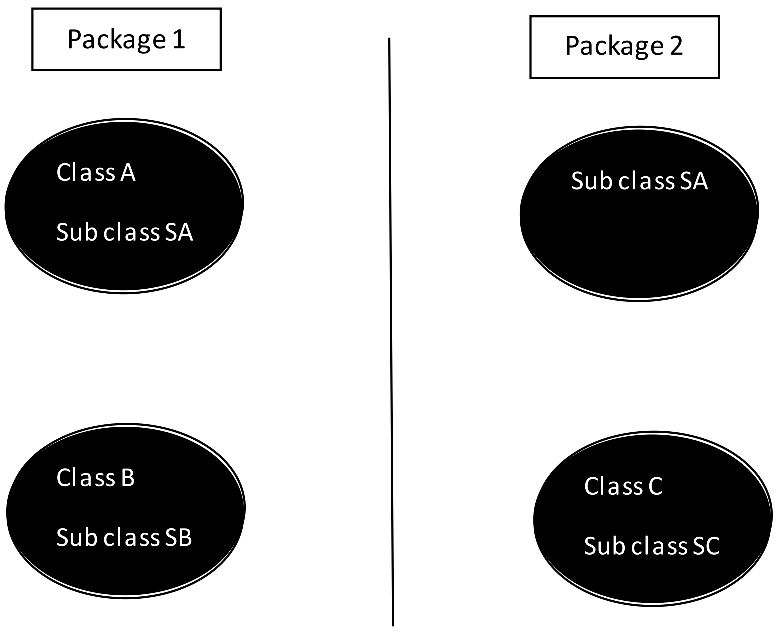
The access to the class, methods and variables are controlled by access modifiers.

Accessibility of different types of access modifiers are as shown below:

	public	protected	default	Private
Same package 13.1.1.Same class	✓	✓	✓	✓

13.1.2. other class and sub class	✓	✓	✓	☒
Other package 13.1.3. Its sub class	✓	✓	☒	☒
13.1.4. other class	✓	☒	☒	☒

Consider following diagram for the above table:



Note : A class can only be public or default.

13.1 Public

If you declare as public you can use that field or method or class in any class of any package.

Consider the previous example, if I declare class A as public

Package 1

```
public class A

{

        body

}
```

You can use this in any class of any package like below

Package 2

```
Class SA extends class A

{

Body

}
```

For variable

If we use

```
public int i=20;
```

We can use this field in any package.

For methods

```
public int add(int X, int Y){

 int Z=X+Y;

return z;

}
```

This can be used in any package.

For example:

Consider two packages pack 1 and pack2:

//super classA in pack1

```
package pack1;
public class ClassA {
        public int a;
        public int b;
        String str1;
        public int addition(int x, int y)
        {
                int add;
                add=x+y;
                return add;
        }
}
```

Subclass of super class in pack1

```
package pack1;
//sub class from the same package
public class Sub_classA extends ClassA {
        // overidding from super class
        public int addtion(int x, int y) {
                int add;
                add=x+y;
                System.out.println("\noveridding can be done by sub class in same package
and the value is  :"+add);
                return add;
        }
        //overloading
        public int addtion(int x, int y, int z) {
                int add;
                add=x+y+z;
                System.out.print("\noverloading can be done by sub class in same package and
the value is :");
                return add;
```

```
          }
// usage of super in subclass
          public void usage_of_super(int x, int y) {

                    System.out.print("\nusage of methods can be done by sub class using super in
the same package and the value is :");
                    System.out.println(super.addtion(x, y));
          }
          // variable usage
          public void Variable_usage() {
                    a=20;
                    System.out.println("\nvariable usage can be done by sub class in same package
and the value is :"+a);
          }
}
```

Different class from pack1

```
package pack1;
//separate class from the same package
public class ClassB {
          ClassA objectA = new ClassA();
// variable usage
  public void variable_usage()
  {
            objectA.a=10;
            System.out.println("the variable can be accessed by separate class in same package
and it value is"+(objectA.a=10));
  }
// method usage
  public void method_usage(int x, int y) {

            objectA.addtion(x, y);
            System.out.println("method usage can be done in same package");
}

}
```

Sub class in pack2

```
package pack2;

import pack1.ClassA;
//subclass from different package
public class Sub_classA extends ClassA {
//          overidding from super class
        public int addtion(int x, int y) {
                int add;
                add=x+y;
                System.out.println("overidding can be done by sub class in other package");
                return add;
        }
        //overloading
        public int addtion(int x, int y, int z) {
                int add;
                add=x+y+z;
                System.out.println("overidding can be done by sub class in other package");
                return add;
        }
// usage of super in subclass
        public void usage_of_super(int x, int y) {
                System.out.println(super.addtion(x, y));
                System.out.println("usage of methods can be done by sub class using super in 
the other package");
        }
// variable usage
        public void Variable_usage() {
                a=20;
                System.out.println("variable usage can be done by sub class in other package 
and the value is \n"+a);
        }
}
```

Other class in pack2

```
package pack2;
import pack1.*;
//separate class from other package
public class ClassC {
```

```
            ClassA objectA = new ClassA();
              public void variable_usage()
              {
                        objectA.a=10;
                        System.out.println("the variable can be accessed by separate class in
different package and it value is"+(objectA.a=10));
              }
              public void method_usage(int x, int y) {

                        objectA.addtion(x, y);
                        System.out.println("method usage can be done by separate class in different
package");
            }

}
```

----
output

```
Script_for_public.testMain()

Usage of public in same package

Same sub class

            usage of methods can be done by sub class using super in the same package and the
value is :3

            variable usage can be done by sub class in same package and the value is :20

            overidding can be done by sub class in same package and the value is  :43

            overloading can be done by sub class in same package and the value is :49

Different class

            method usage can be done in same package

            the variable can be accessed by separate class in same package and it value is10
```

Usage of public in different package

Same sub class

usage of methods can be done by sub class using super in the other package and the value is :3

variable usage can be done by sub class in other package and the value is :20

overidding can be done by sub class in other package and the value is :43

overloading can be done by sub class in other package and the value is :49

Different class

method usage can be done by separate class in different package5

the variable can be accessed by separate class in different package and it value is10

13.2 Protected

Protected can be used in all classes in same package and its subclasses in other package.

Consider the diagrammatic example,

If we declare classA as public, its methods and fields as protected then we can use classA in subclassA, classB and subclass B in package1 and only subclassesSA in package2.

For Example

Consider two packages pack 1 and pack2:

```
---------------------------------------------------------------------------------------------------------
//super classA in pack1
---------------------------------------------------------------------------------------------------------
package pack1;
public class ClassA {
        protected int a;
        protected int b;
        String str1;
```

```
		protected int addition(int x, int y)
		{
			int add;
			add=x+y;
			return add;
		}
}
```

Subclass of super class in pack1

```
package pack1;
//sub class from the same package
public class Sub_classA extends ClassA {
		// overidding from super class
		public int addition(int x, int y) {
			int add;
			add=x+y;
			System.out.println("\noveridding can be done by sub class in same package
and the value is  :"+add);
			return add;
		}
		//overloading
		public int addtion(int x, int y, int z) {
			int add;
			add=x+y+z;
			System.out.print("\noverloading can be done by sub class in same package and
the value is :");
			return add;
		}
// usage of super in subclass
		public void usage_of_super(int x, int y) {

			System.out.print("\nusage of methods can be done by sub class using super in
the same package and the value is :");
			System.out.println(super.addtion(x, y));
		}
		// variable usage
		public void Variable_usage() {
			a=20;
```

```
                System.out.println("\nvariable usage can be done by sub class in same package
and the value is :"+a);
        }
}
```

Different class from pack1

```
package pack1;
//separate class from the same package
public class ClassB {
        ClassA objectA = new ClassA();
// variable usage
  public void variable_usage()
  {
          objectA.a=10;
          System.out.println("the variable can be accessed by separate class in same package
and it value is"+(objectA.a=10));
  }
// method usage
  public void method_usage(int x, int y) {

          objectA.addtion(x, y);
          System.out.println("method usage can be done in same package");
}
}
```

Sub class in pack2

```
package pack2;

import pack1.ClassA;
//subclass from different package
public class Sub_classA extends ClassA {
//        overidding from super class
        public int addtion(int x, int y) {
                int add;
                add=x+y;
                System.out.println("overidding can be done by sub class in other package");
                return add;
        }
        //overloading
        public int addtion(int x, int y, int z) {
                int add;
                add=x+y+z;
                System.out.println("overidding can be done by sub class in other package");
                return add;
        }
// usage of super in subclass
        public void usage_of_super(int x, int y) {
                System.out.println(super.addtion(x, y));
                System.out.println("usage of methods can be done by sub class using super in
the other package");
        }
// variable usage
        public void Variable_usage() {
                a=20;
                System.out.println("variable usage can be done by sub class in other package
and the value is \n"+a);
        }
}
```

Other class in pack2

```
package pack2;
import pack1.*;
//separate class from other package
public class ClassC {
        ClassA objectA = new ClassA();
          public void variable_usage()
          {
                  objectA.a=10;
                  System.out.println("the variable can be accessed by separate class in
different package and it value is"+(objectA.a=10));
          }
          public void method_usage(int x, int y) {

                  objectA.addtion(x, y);
                  System.out.println("method usage can be done by separate class in different
package");
```

```
          }
}
```

----

output

Script_for_public.testMain()

Usage of public in same package

Same sub class

```
          usage of methods can be done by sub class using super in the same package and the
          value is :3

          variable usage can be done by sub class in same package and the value is :20

          overidding can be done by sub class in same package and the value is  :43

          overloading can be done by sub class in same package and the value is :49
```

Different class

```
          method usage can be done in same package

          the variable can be accessed by separate class in same package and it value is 10
```

Usage of public in different package

Same sub class

```
          usage of methods can be done by sub class using super in the other package and the
          value is :3

          variable usage can be done by sub class in other package and the value is :20

          overidding can be done by sub class in other package and the value is  :43

          overloading can be done by sub class in other package and the value is :49
```

Different class

Exception occurred during playback of script [Script1_protected] [RationalTestScriptError on line 57 of script Script1_protected: [java.lang.Error] - Unresolved compilation problems:

The method addtion(int, int) from the type ClassA is not visible

The method addtion(int, int) from the type ClassA is not visible

.].

13.3 Default

If we use access modifiers as default then we can use it in the same package not in other packages.

Consider the diagrammatic example,

If we declare classA as default then we can use classA in subclassA, classB and subclass B in package1 and we cannot use it in subclassesSA,classC and subclassC in package two.

For example:

Consider two packages pack 1 and pack2:

```
-----------------------------------------------------------------------------------------------------------------
//super classA in pack1
-----------------------------------------------------------------------------------------------------------------
package pack1;
class ClassA {
        int a;
        int b;
        String str1;
         int addtion(int x, int y)
        {
                int add;
                add=x+y;
                return add;
        }
}
-----------------------------------------------------------------------------------------------------------------
```

Subclass of super class in pack1

```
package pack1;
//sub class from the same package
public class Sub_classA extends ClassA {
          // overidding from super class
          public int addtion(int x, int y) {
                    int add;
                    add=x+y;
                    System.out.println("\noveridding can be done by sub class in same package
and the value is  :"+add);
                    return add;
          }
          //overloading
          public int addtion(int x, int y, int z) {
                    int add;
                    add=x+y+z;
                    System.out.print("\noverloading can be done by sub class in same package and
the value is :");
                    return add;
          }
// usage of super in subclass
          public void usage_of_super(int x, int y) {

                    System.out.print("\nusage of methods can be done by sub class using super in
the same package and the value is :");
                    System.out.println(super.addtion(x, y));
          }
          // variable usage
          public void Variable_usage() {
                    a=20;
                    System.out.println("\nvariable usage can be done by sub class in same package
and the value is :"+a);
          }
}
```

Different class from pack1

```
package pack1;
//separate class from the same package
```

```
public class ClassB {
        ClassA objectA = new ClassA();
// variable usage
  public void variable_usage()
  {
          objectA.a=10;
          System.out.println("the variable can be accessed by separate class in same package
and it value is"+(objectA.a=10));
  }
// method usage
  public void method_usage(int x, int y) {

          objectA.addtion(x, y);
          System.out.println("method usage can be done in same package");
}
}
```

Sub class in pack2

```
package pack2;

import pack1.ClassA;
//subclass from different package
public class Sub_classA extends ClassA {
//        overidding from super class
        public int addtion(int x, int y) {
                int add;
                add=x+y;
                System.out.println("overidding can be done by sub class in other package");
                return add;
        }
        //overloading
        public int addtion(int x, int y, int z) {
                int add;
                add=x+y+z;
                System.out.println("overidding can be done by sub class in other package");
                return add;
        }
// usage of super in subclass
        public void usage_of_super(int x, int y) {
```

```
                    System.out.println(super.addtion(x, y));
                    System.out.println("usage of methods can be done by sub class using super in 
the other package");
          }
```

<u>// variable usage</u>

```
          public void Variable_usage() {
                    a=20;
                    System.out.println("variable usage can be done by sub class in other package 
and the value is \n"+a);
          }
}
```

Other class in pack2

```
package pack2;
import pack1.*;
```

//separate class from other package

```
public class ClassC {
          ClassA objectA = new ClassA();
            public void variable_usage()
            {
                      objectA.a=10;
                      System.out.println("the variable can be accessed by separate class in 
different package and it value is"+(objectA.a=10));
            }
            public void method_usage(int x, int y) {

                      objectA.addtion(x, y);
                      System.out.println("method usage can be done by separate class in different 
package");
          }

}
```

----

output

Script_for_default.testMain()

Usage of default in same package

Same sub class

usage of methods can be done by sub class using super in the same package and the value is :3

variable usage can be done by sub class in same package and the value is :20

overidding can be done by sub class in same package and the value is :43

overloading can be done by sub class in same package and the value is :49

Different class

method usage can be done in same package

the variable can be accessed by separate class in same package and it value is 10

Usage of public in different package

Same sub class

Exception occurred during playback of script [Script1_default] [RationalTestScriptError on line 50 of script Script1_default: [java.lang.Error] - Unresolved compilation problems:

ClassA cannot be resolved to a type

The method addtion(int, int) is undefined for the type Object

a cannot be resolved

a cannot be resolved

.].

13.4 Private

If we use access modifier as private then we can use it only in the class but not in its subclass of the same package and other packages.

Consider the diagrammatic example,

If we declare member of classA as private then we can use it only in classA, not in subclassA, classB and subclass B in package1 and we cannot use it in package two.

}

For example

Consider two packages pack 1 and pack2:

//super classA in pack1

```
package pack1;
class ClassA {
        private int a;
        private int b;
        String str1;
        private  int addtion(int x, int y)
        {
                int add;
                add=x+y;
                return add;
        }
}
```

Subclass of super class in pack1

```
package pack1;
//sub class from the same package
public class Sub_classA extends ClassA {
        // overidding from super class
        public int addtion(int x, int y) {
                int add;
                add=x+y;
                System.out.println("\noveridding can be done by sub class in same package
and the value is  :"+add);
                return add;
        }
        //overloading
        public int addtion(int x, int y, int z) {
                int add;
                add=x+y+z;
                System.out.print("\noverloading can be done by sub class in same package and
the value is :");
                return add;
        }
// usage of super in subclass
        public void usage_of_super(int x, int y) {

                System.out.print("\nusage of methods can be done by sub class using super in
the same package and the value is :");
                System.out.println(super.addtion(x, y));
        }
        // variable usage
        public void Variable_usage() {
                a=20;
                System.out.println("\nvariable usage can be done by sub class in same package
and the value is :"+a);
        }
}
```

Different class from pack1

```
package pack1;
//separate class from the same package
public class ClassB {
        ClassA objectA = new ClassA();
// variable usage
  public void variable_usage()
  {
          objectA.a=10;
          System.out.println("the variable can be accessed by separate class in same package
and it value is"+(objectA.a=10));
  }
// method usage
  public void method_usage(int x, int y) {
```

```
            objectA.addtion(x, y);
            System.out.println("method usage can be done in same package");
}

}
```

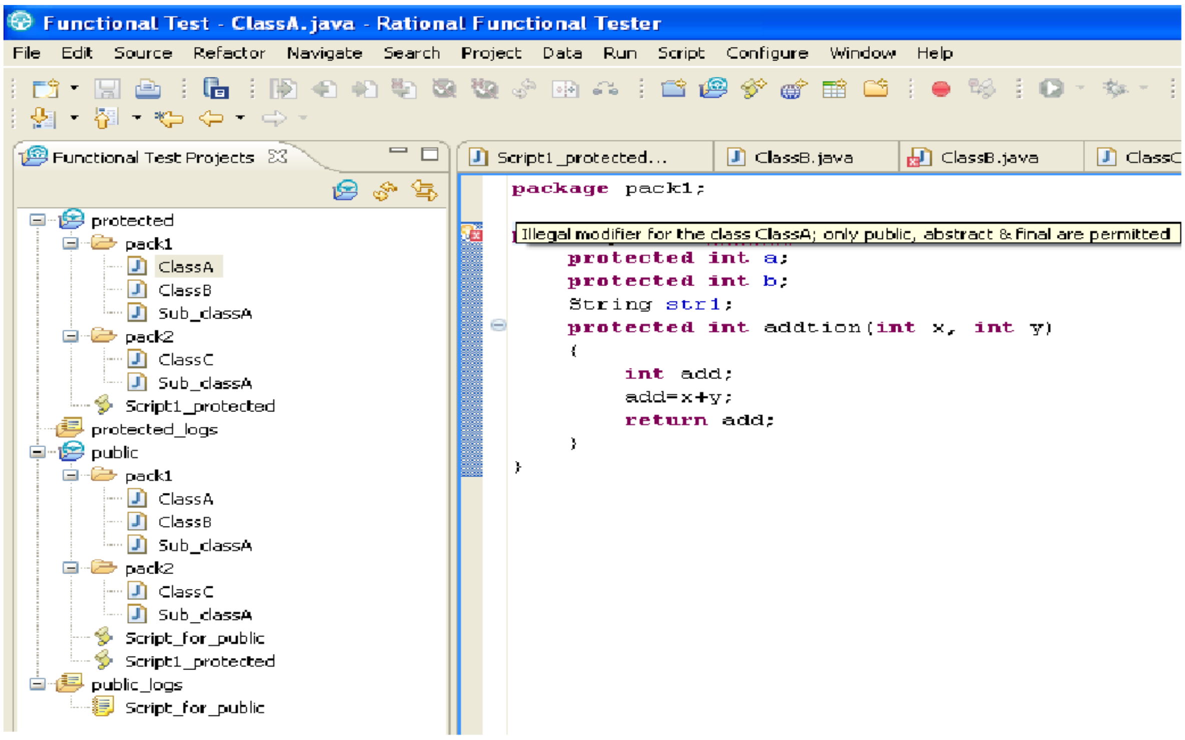
Sub class in pack2

```
package pack2;

import pack1.ClassA;
//subclass from different package
public class Sub_classA extends ClassA {
//      overidding from super class
        public int addtion(int x, int y) {
                int add;
                add=x+y;
                System.out.println("overidding can be done by sub class in other package");
                return add;
        }
        //overloading
        public int addtion(int x, int y, int z) {
                int add;
                add=x+y+z;
                System.out.println("overidding can be done by sub class in other package");
                return add;
        }
// usage of super in subclass
        public void usage_of_super(int x, int y) {
                System.out.println(super.addtion(x, y));
                System.out.println("usage of methods can be done by sub class using super in
the other package");
        }
// variable usage
        public void Variable_usage() {
                a=20;
                System.out.println("variable usage can be done by sub class in other package
and the value is \n"+a);
        }
}
```

Other class in pack2

We can't use this private fields and methods in sub class of its declared class of the same package and other packages.



```
package pack1;

Illegal modifier for the class ClassA; only public, abstract & final are permitted
    protected int a;
    protected int b;
    String str1;
    protected int addtion(int x, int y)
    {
        int add;
        add=x+y;
        return add;
    }
}
```

14. Escape sequence

Escape sequence is meant to have some special meaning.

\n	Newline
\t	Tab
\b	Backspace

\f	form feed
\r	Carriage return
\"	" (double quote)
\'	' (single quote)
\\	\ (back slash)

15. Scope of a variable

Scope of a variable is nothing but the accessibility of a variable in a function.

For example:

Consider the following example

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

 {

        Int inner=1;

        {

                Int outer =2;

                System.out.println(“inner=”+inner);

                System.out.println(“outer =”+ outer);

        }

        Int outer=3

        System.out.println(“inner=”+inner);

        System.out.println(“outer =”+ outer);

}
```

Output

```
inner=1

outer=2

inner=1

outer=3
```

Here scope of the variable outer = 2 is only inside the loop.

16.Keywords

Keyword in java means that it has predefined function which can be directly used in a function. It has got a specific meaning.

16.1 List of keywords used in java [4]

abstract	Case	continue	enum	for	instanceof	new	transient
Assert	Catch	Default	extends	goto	int	package	try
boolean	Char	Do	final	if	interface	private	void
Break	Class	Double	finally	implements	long	protected	volatile
Byte	Const	Else	float	import	native	public	while
Return	Short	Static	strictfp	super	switch	synchronized	this

17. Type casting

Type casting is nothing but type conversion from one type to another type. It can be done in two ways implicit and explicit.

17.1 Implicit Type conversion

It is nothing but converting a one data type to another data type by the following rules :

An implicit cast is done when an Object reference is assigned (cast) to.

Note: Converted type should have more memory than converting type

1. Two types should be compatible.

For Example,

Let ‘c’ be a variable of type Car class and ‘f’ be of class Ford and ‘v’ be a vehicle interface reference. We can assign the Ford reference to the Car variable:

I.e. we can do the following

Example

c = f; //Ok Compiles fine'"

Where c = new Car();

And, f = new Ford();

The compiler automatically handles the conversion (assignment) since the types are compatible (sub class - super class relationship), i.e., the type Car can hold the type Ford since a Ford is a Car.

17.2 Explicit Type conversion

When Data and Object conversion cannot be done through implicit conversion we go for explicit type. So we cast the data or object to convert that to our required data or object type.

Syntax

```
Data_type variable = (Data_type) Variable_other_type;
```

For Example

```
float  f=21.003;

Int  i  = (Int) f;

System.out.println (i);
```

Output

```
21
```

So it will truncate the data and it will give the integer value of our needed type.

18. Primitive data types

Primitive data types are predefined in java language. It has got some predefined meaning. The eight primitive data types are: **byte, short, int, long, char, float, double,** and **Boolean**. These eight data types can be combined into four different types:

Integer : returns whole number : **byte, short, int,** and **long.**

Floating type variables: numbers with fractional precision - **float** and **double**.

Character: Alphanumeric Values - **char**.

Boolean : returns true or false - **boolean**.

Data Type	Default Value
byte, short, int, long	0
float, double	0.0f, 0.0d
char	'\u0000'
boolean	false

19. Regular expression[5]

A Regular Expression defines a pattern that matches a certain set of strings.

Regular expressions are used on dynamic data which keeps on changing in an application.

^	Matches beginning of line.
$	Matches end of line.
.	Matches any single character except newline. Using m option allows it to match newline as well.
[...]	Matches any single character in brackets.
[^...]	Matches any single character not in brackets
\A	Beginning of entire string
\z	End of entire string
\Z	End of entire string except allowable final line terminator.
re*	Matches 0 or more occurrences of preceding expression.

re+	Matches 1 or more of the previous thing
re?	Matches 0 or 1 occurrence of preceding expression.
re{ n}	Matches exactly n number of occurrences of preceding expression.
re{ n,}	Matches n or more occurrences of preceding expression.
re{ n, m}	Matches at least n and at most m occurrences of preceding expression.
a| b	Matches either a or b.
(re)	Groups regular expressions and remembers matched text.
(?: re)	Groups regular expressions without remembering matched text.
(?> re)	Matches independent pattern without backtracking.
\w	Matches word characters.
\W	Matches nonword characters.
\s	Matches whitespace. Equivalent to [\t\n\r\f].
\S	Matches nonwhitespace.
\d	Matches digits. Equivalent to [0-9].
\D	Matches nondigits.
\A	Matches beginning of string.
\Z	Matches end of string. If a newline exists, it matches just before newline.
\z	Matches end of string.
\G	Matches point where last match finished.
\n	Back-reference to capture group number "n"
\b	Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets.
\B	Matches nonword boundaries.
\n, \t, etc.	Matches newlines, carriage returns, tabs, etc.
\Q	Escape (quote) all characters up to \E
\E	Ends quoting begin with \Q

20. Non Access modifiers or Non Access Specifiers

The difference between access specifiers and Non access specifiers is

Access specifiers **- where to use**

Non Access specifiers -when to use

Different types of access modifiers are listed below

20.1 Static

Static is a type of non access modifier which is used to declare a variable or method as static.

If we declare a method as static then it can be accessed without creating any reference to the class in which the method is declared. In a program always static block is executed first if present by allocating memory to the static variable. It can be used for three types:

1. Variables
2. Methods
3. Static blocks

20.1.1 Variable

A variable can be declared with **static** keyword. If we use static then all the objects can use the same reference to access the variable.

Syntax

```
static data_type variable;
```

20.1.2 Method

A method can be declared static. If we use static then without creating the reference for that class we can access the method.

Syntax

```
static data_type method()
{
        Body;
}
```

20.1.3 Static block

Static block is always executed first when the class is loaded. So without creating the object we can execute this loop. Inside a static block we can write statements and manipulate the fields as required. We should always declare static block in a class but not inside a method.

Syntax

```
static
{
        Body;
}
```

For example:

Consider the following program

```
class maths
{
        Static int a =3;
        Static int b;
        Public static int variables_print()
        {
                System.out.println("the value of a is"+a);
                System.out.println("the value of b is"+b);
        }
        Public static int add(int x)
        {
                Int z = x+a+b;

                return z;

        }

        static

        {

                System.out.println(“static block is executed first”);

                b= a*2;

        }

}
```

Output

```
Static block is executed first

The value of a is 3

The value of a is 6
```

In that program static block gets executed first. We can use static method without creating instances like below

```
Public static void main (string arg [])

{

        System.out.println (maths. add (1));

}
```

Output

```
10
```

20.2 Final

Final is a type of non access modifier which is used to finalise a field or method or class. This keyword can be used in four ways

1. Variable
2. Parameters
3. Methods
4. Classes

20.2.1 Variable

If we use a variable as final we have to initialize at creation or else it can only be initialized through a constructor. These variable values cannot be changed.

Syntax

final data_type variable;

For example

1. Consider two cases
   a. if we declare like

      final int i = 0;

      This value cannot be changed at any place in any package.

      b. if we declare like

      final int i;

      This value can only be changed in a constructor as shown below

```
class maths

{

        math()

        {

                Int i =23;
        }

    }
```

20.2.2 Parameters

We can extend the usage of final to parameters also. If once we set the parameters as final then the value cannot be changed inside the body of the method. If we try to change the value it will throw an error.

Syntax

```
        static data_type method(final int x)

    {

        Body;

    }
```

20.2.3 Methods

We can extend the usage of final to methods. If we set the method as final then it cannot be overridden or overloaded. If we try to overload it will throw you an error message.

Syntax

```
        final data_type method()

        {

            Body;

        }
```

20.2.4 Classes

If we declare a class as final then the class cannot be inherited by any other class.

Syntax

```
final class maths

{

        Body;

}
```

For example:

```
final class maths

{

        final int i=3;

        public final int add(final int x)

        {

                Int b;

                b= x+y;

                return b;

        }

}
```

In this we have seen different ways to use a final keyword.

20.3 Abstract[6]

Abstract is used when we have to write our own implementation of a predefined method.

Note: If a method is declared as Abstract then a class should also be an abstract class.

Syntax

```
abstract class class_name
{
        abstract return type method();
}
```

For Example:

```
abstract class maths
{
        abstract int add(y);
}
Class maths1  extends maths
{
        public int add(string y)
        {
                System.out.println(y);
        }
}
```

21. Bibliography

1) http://publib.boulder.ibm.com/infocenter/wsadhelp/v5r1m2/index.jsp?topic=/com.sun.api.doc/java/util/Arrays.html
2) Head first java by Kathy Sierra & Bert Bates 2nd Edition Pg.No.178
3) http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.ii.doc/ad/rwrjwrap.htm
4) http://en.wikipedia.org/wiki/List_of_Java_keywords
5) http://download.oracle.com/javase/tutorial/essential/regex/intro.html //regular expression
6) http://xahlee.org/java-a-day/abstract_class.html
7) http://xahlee.org/java-a-day/access_specifiers.html
8) http://www.javabeginner.com/learn-java/java-constructors