Marwadi University

UNIT : 3 Classes, Objects and Methods

KNOWLEDGE IS THE CURRENCY FOR THE 21st CENTURY

INDEX

- Class and Object,
- Object reference,
- What is Constructor? Constructor
- Method Overloading,
- Recursion,
- Passing and Returning object form Method,
- new operator,
- this and static keyword,
- finalize() method,
- Access control and modifiers,
- Nested class, Inner class, Anonymous inner class, Abstract class.

Class and Object

- A class can be defined as a **template/blueprint** that describes the **behavior/state** that the object of its type support.
- Objects have states and behaviors.
- Example: A dog has **states - color, name, breed** as well as **behaviors – wagging the tail, barking, eating**.
- **An object is an instance of a class**.

Object

- The **new keyword is used to allocate memory for Object at runtime.**
- All objects get memory in **Heap memory area**.
- Syntax:

```
Student s1=new Student();
//creating an object of Student Class
```

Object reference

```
class Student{

int id;//field or data member or instance variable
String name;

public static void main(String args[]){

Student s1=new Student();//creating an object of Student
System.out.println(s1.id);//accessing member through reference variable
System.out.println(s1.name);
 }
}
```

Constructor

- In Java, a **constructor is a block of codes similar to the method**.
- **It is called when an instance of the object is created, and memory is allocated for the object.**
- It is a special type of method which is used to initialize the object.
- Every time an object is created **using new() keyword, at least one constructor is called.**

Rules for creating Java constructor

Following are the rules to define a constructor.

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

Types of Java constructors

There are two types of constructors in Java:

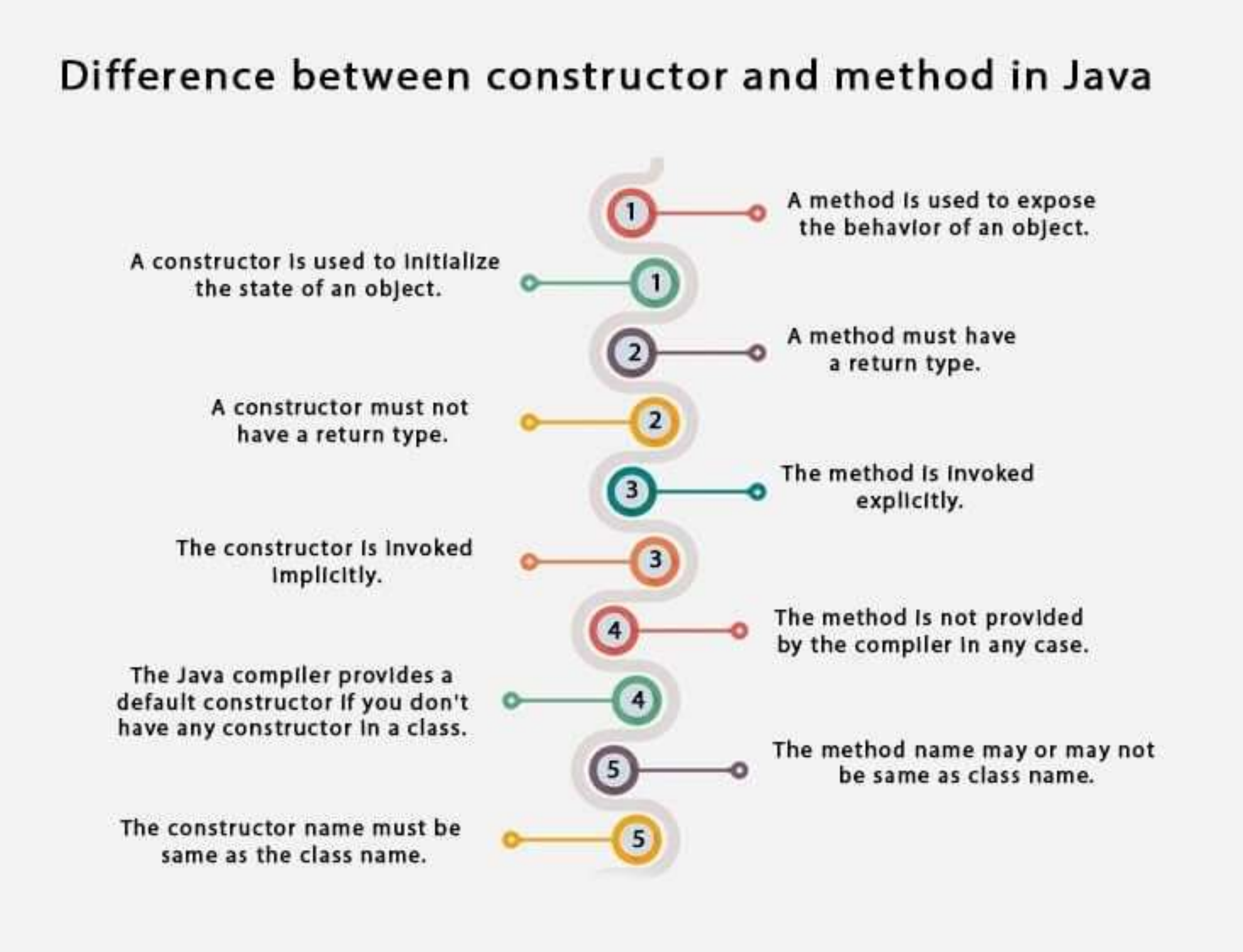
1. Default constructor (no-arg constructor)
2. Parameterized constructor

Default constructor

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

Parameterized constructor

```
class Student{

    int id;
    String name;
    int age;

    //creating two arg constructor
    Student(int i,String n){
    id = i;
    name = n;
    }

    //creating three arg constructor
    Student(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }

void display(){System.out.println(id+" "+name+" "+
age);}

public static void main(String args[]){
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan",25);
    s1.display();
    s2.display();
  }
```



Difference between constructor and method in Java

1 A method is used to expose the behavior of an object.

1 A constructor is used to initialize the state of an object.

2 A method must have a return type.

2 A constructor must not have a return type.

3 The method is invoked explicitly.

3 The constructor is invoked implicitly.

4 The method is not provided by the compiler in any case.

4 The Java compiler provides a default constructor if you don't have any constructor in a class.

5 The method name may or may not be same as class name.

5 The constructor name must be same as the class name.

Method Overloading

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading.**

```
class Adder{
        static int add(int a,int b)
        {return a+b;}
        static int add(int a,int b,int c)
        {return a+b+c;}
        }

class Main{
        public static void main(String[] args)
        {
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(11,11,11));
        }
}
```

Passing and Returning object form Method

```
public class Student {
    String name;
    int rollnum, marks;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getRollnum() {
        return rollnum;
    }
    public void setRollnum(int rollnum) {
        this.rollnum = rollnum;
    }
    public int getMarks() {
        return marks;
    }
    public void setMarks(int marks) {
        this.marks = marks;
    }
    public static Student checkTopper(Student s1,Student s2){
        if(s1.marks < s2.marks){
            return s2;
        }
        else{
            return s1;
        }
    }
}
```

Passing and Returning object form Method

```
public class StudentTest {
    public static void main(String[] args) {
        Student s1 = new Student();
        Student s2 = new Student();
        Student s3 = new Student();
        s1.setRollnum(1);
        s1.setName("SAURABH");
        s1.setMarks(25);
        s2.setRollnum(2);
        s2.setName("PRATIBHA");
        s2.setMarks(30);
        s3 = Student.checkTopper(s1,s2);
        System.out.println("Name of Topper is:"+s3.name);
    }
}
```

Recursion in Java

- Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

```
public class RecursionExample3 {
  static int factorial(int n){
      if (n == 1)
        return 1;
      else
        return(n * factorial(n-1));
  }

public static void main(String[] args) {
System.out.println("Factorial of 5 is: "+factorial(5));
         }
}
```

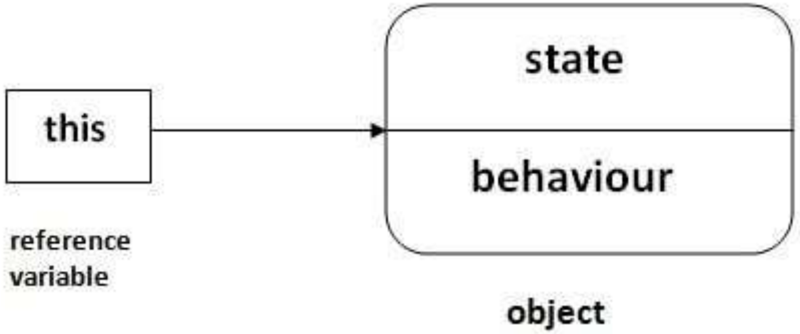
new operator

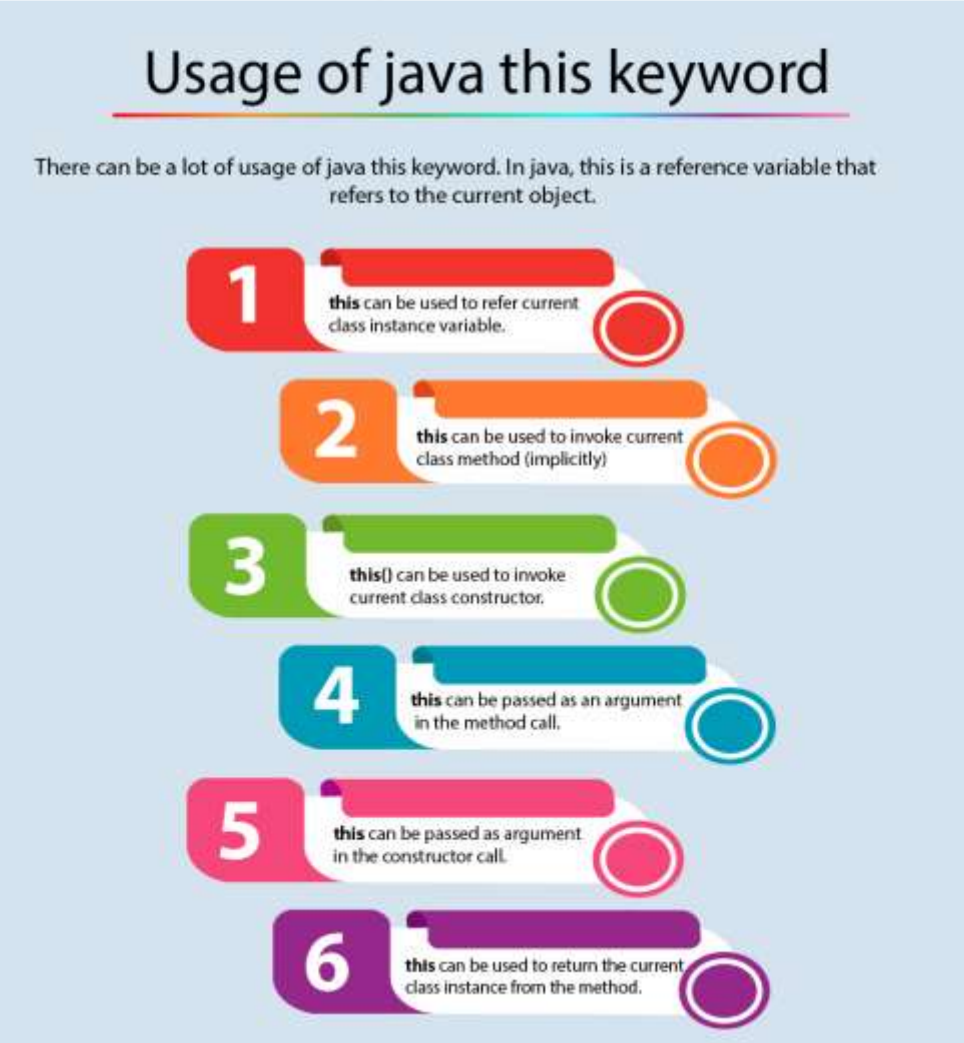
- The **'new'** operator in java is responsible for the creation of **new object** or we can say instance of a class.
- The new keyword is used to allocate memory for Object at runtime.
- All objects get memory in Heap memory area.
- Syntax:

```
Student s1=new Student();
//creating an object of Student Class
```

this keyword

- this is a **reference variable** that refers to the **current object**.





Usage of java this keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

1. **this** can be used to refer current class instance variable.
2. **this** can be used to invoke current class method (implicitly)
3. **this()** can be used to invoke current class constructor.
4. **this** can be passed as an argument in the method call.
5. **this** can be passed as argument in the constructor call.
6. **this** can be used to return the current class instance from the method.

refer current class instance variable

```
class Student{

int rollno;
String name;
float fee;

Student(int rollno,String name,float fee){
        this.rollno=rollno;
        this.name=name;
        this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

this() : to invoke current class constructor

```
class A{
        A()
        {
        System.out.println("hello a"); }

        A(int x){
        this();
        System.out.println(x);
        }
}
```

```
class TestThis5{
public static void main(String args[])
        {
        A a=new A(10);
        }
}
```

to pass as an argument in the method

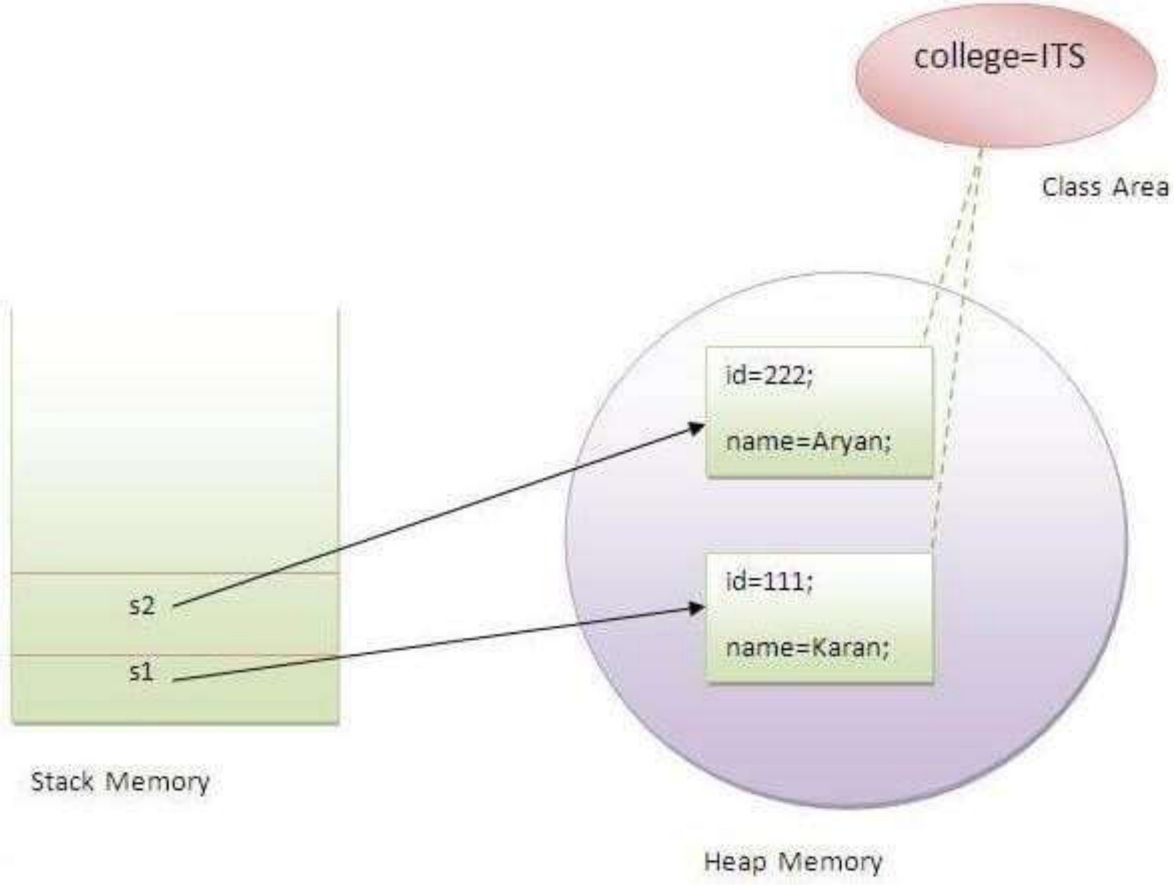
```
class S2{
                void m(S2 obj)
                {
                System.out.println("method is invoked");
                }
                void p()
                {
                m(this);
                }
public static void main(String args[])
                {
                 S2 s1 = new S2();
                 s1.p();
                }
}
```

static keyword

- The **static keyword** in Java is used for **memory management** mainly.
- The static can be:
  1. Variable (also known as a class variable)
  2. Method (also known as a class method)
  3. Block
  4. Nested class

1. static variable

- The static variable can be used to refer to the common property of all objects
- For example: college name of students, the company name of employees



Example:

```
class Student{
    int rollno;//instance variable
    String name;
    static String college =“RK
University";//static variable
    //constructor
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display the values
    void display (){System.out.println(rollno+" "+
name+" "+college);}
```

```
}
//Test class to show the values of objects
public class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by t
he single line of code
 //Student.college="BBDIT";
 s1.display();
 s2.display();
 }
}
```

2. static Method

If you apply static keyword with any method, it is known as static method.

1. A static method belongs to the class rather than the object of a class.
2. A static method can be invoked without the need for creating an instance of a class.
3. A static method can access static data member and can change the value of it.

```
class Student{
    int rollno;
    String name;
    static String college = "ITS";
     //static method to change the value of static variable
    static void change(){
    college = "BBDIT";
    }
    //constructor to initialize the variable
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object
public class TestStaticMethod{
   public static void main(String args[]){
   Student.change();//calling change method
   //creating objects
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   Student s3 = new Student(333,"Sonoo");
   //calling display method
   s1.display();
   s2.display();
   s3.display();
   }
}
```

- **Restrictions for the static method**
  1. The static method can not use non static data member or call non-static method directly.
  2. this and super cannot be used in static context.

```
class A{
 int a=40;//non static

 public static void main(String args[]){
  System.out.println(a);
 }
}
```

Output: Compile Time Error

static block

- Is used to initialize **the static data member**.
- It is executed before the main method at the time of classloading.

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
  System.out.println("Hello main");
  }
}
```

Output:static block is invoked Hello main

Access Modifiers

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

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

private access modifier

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

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

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

default access modifier

- In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.
- The scope of class A and its method msg() is default so it cannot be accessed from outside the package

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

```
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

protected access modifier

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

Example

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

```
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String arg
s[]){
   B obj = new B();
   obj.msg();
  }
}
```

public access modifier

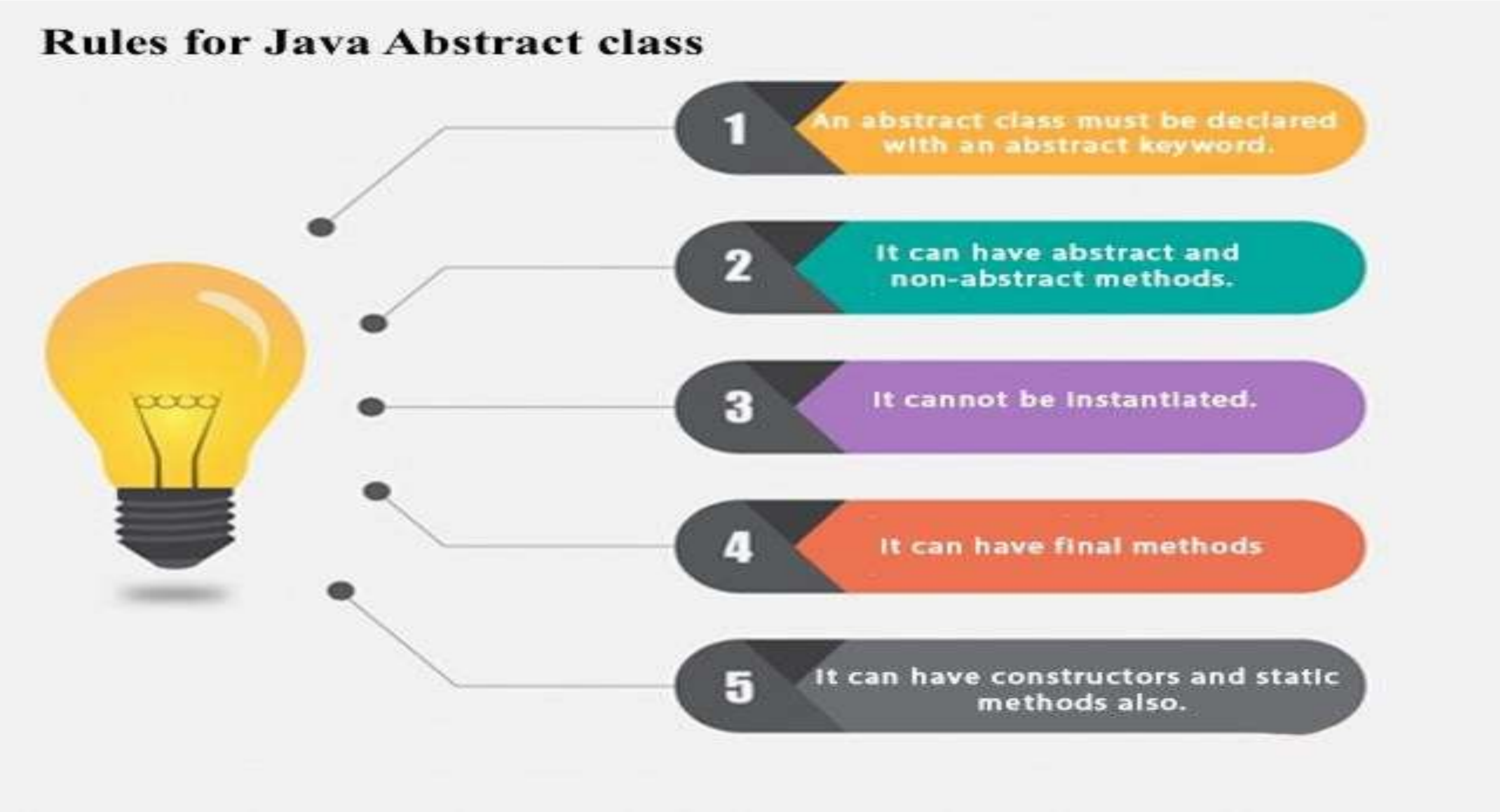
- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
//save by A.java

package pack;
public class A{
public void msg(){System.out.print
ln("Hello");}
}
```

```
//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String arg
s[]){
   A obj = new A();
   obj.msg();
  }
}
```

Abstract class

- A class which is declared with the **abstract keyword** is known as an abstract class in Java.
- It can have **abstract and non-abstract methods** (method with the body).
- There are two ways to achieve abstraction in java
  1. Abstract class (0 to 100%)
  2. Interface (100%)



Rules for Java Abstract class

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

Example 1: Abstract class

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
@Overriden
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

```
OUTPUT:
running safely
```

Example 2:

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+
b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+
b.getRateOfInterest()+" %");
}}
```

Inner class

- **Java inner class** or nested class is a class which is declared inside the class or interface.
- We use inner classes to logically group classes and interfaces
- it can access all the members of outer class including private data members and methods.

```
class Java_Outer_class{
   //code
   class Java_Inner_class{
    //code
   }
  }
```

Types of inner class

1. Nested Inner class
2. Method Local inner classes

Nested Inner

- **Nested Inner class** can access any private instance variable of outer class.

```
class Outer {
   // Simple nested inner class
   class Inner {
      public voidshow() {
           System.out.println("In a nested class method");
      }
   }
}
class Main {
   public static void main(String[] args) {
       Outer.Inner in = new Outer().new Inner();
       in.show();
   }
}
```

OUTPUT:

```
In a nested class method
```

Method Local inner classes

- Inner class can be declared within a method of an outer class. In the following example, Inner is an inner class in outerMethod().

```
class Outer {
    void outerMethod() {
        System.out.println("inside outerMethod");
        // Inner class is local to outerMethod()
        class Inner {
            void innerMethod() {
                System.out.println("inside innerMethod");
            }
        }
        Inner y = new Inner();
        y.innerMethod();
    }
}
class MethodDemo {
    public static void main(String[] args) {
        Outer x = new Outer();
        x.outerMethod();
    }
}
```

```
OUTPUT:
Inside outerMethod
Inside innerMethod
```

THANK YOU