UNIT-2

Prepared By:
Deepak Kumar Sharma
Asst. Professor
SoCS UPES Dehradun

UNIT-II

- Static keyword

• Extended Class, Constructors in Extended classes, Inheriting and Redefining Members
• Type Compatibility and Conversion, protected, final Methods and Classes
• Abstract methods and classes, Object Class
• Designing extended classes, Single Inheritance versus Multiple Inheritance
• Interface, Interface Declarations, Extending Interfaces, Working with Interfaces
• Marker Interfaces, When to Use Interfaces, Package naming, type imports
• Package access, package contents, package objects and specifications

Java static Keyword

- Mainly used for memory management.
- Can apply static keyword with variables, methods, blocks and nested classes.
- The static keyword belongs to the class than an instance of the class.

The static can be:

- Variable (also known as a class variable)
- Method (also known as a class method)
- Block
- Nested class

Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects
  - for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

- It makes your program **memory efficient** (i.e., it saves memory).

Java static variable

- Without static variable

```
class Student{
    int rollno;
    String name;
    String college="UPES";
}
```

- With static variable

```
class Student{
    int rollno;
    String name;
    static String college="UPES";
}
```

Assume 100 students in the class.
Which version of the program will you prefer?

Java static property is shared to all objects.

Java static variable

```
class Student{
   int rollno;String name; //instance variable
   static String college ="UPES";//static variable
   Student(int r,String n){
   rollno = r; name = n;
   }
void display ()
{System.out.println(rollno+" "+name+" "+college);}
}
public class TestStaticVariable{
 public static void main(String args[]){
 Student s1 = new Student(1,"Raj");
 Student s2 = new Student(2,"Rohan");
 s1.display();
 s2.display();
 Student.college="University of petroleum and Energy Studies";
 s1.display();
 s2.display();
 }
}
```

Output:

1 Raj UPES
2 Rohan UPES
1 Raj University of petroleum and Energy Studies
2 Rohan University of petroleum and Energy Studies

Java static variable

- **Count number of instances using static variable**

```
class Counter2{
        static int count=0;//will get memory only once and retain its value
        Counter2(){
                count++;//incrementing the value of static variable
                System.out.print(count+" ");
        }
        public static void main(String args[]){
        Counter2 c1=new Counter2();
        Counter2 c2=new Counter2();
        Counter2 c3=new Counter2();
        }
}
```

Output: 1 2 3

Java static method

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

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

Java static method

Example:

```
class Calculate{
  static int cube(int x){
  return x*x*x;
  }

  public static void main(String args[]){
  int result=Calculate.cube(5);  //or cube(5)
  System.out.println(result);
  }
}
```

Java static method

```
class Student{
     int rollno;String name;
     static String college = "UPES";
     static void change(){
     college = "UPES Dehradun";
     }
     Student(int r, String n){
     rollno = r; name = n;
     }
     void display(){System.out.println(rollno+" "+name+" "+college);}
}
public class TestStaticMethod{
    public static void main(String args[]){
    Student.change();//calling change method
    Student s1 = new Student(1,"Karan");
    Student s2 = new Student(2,"Aryan");
    //calling display method
    s1.display();
    s2.display();
    }
}
```

Output:
1 Karan UPES Dehradun
2 Aryan UPES Dehradun

Restrictions for the static method

There are two main restrictions for the static method.

- The static method can not use non static data member or call non-static method directly.
- 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

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

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

Output: 40

Why is the Java main method static?

Java **main()** method is always static, so that compiler can call it without the creation of an object or before the creation of an object of the class.

- In any Java program, the **main()** method is the starting point from where compiler starts program execution. So, the compiler needs to call the main() method.
- If the **main()** is allowed to be non-static, then while calling the **main()** method JVM has to instantiate its class.

Java static block

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

```
class A2{
  A2(){System.out.println("Constructor Called");}

static{
            System.out.println("static block is invoked");
        }

  public static void main(String args[]){
   System.out.println("Hello main");
   A2 a=new A2();
  }
}
```

Output:

```
static block is invoked
Hello main
Constructor Called
```

Example: Static block in two classes

```
class CheckStatic{
    static{
        System.out.println("I am static block of Checkstatic");
    }
       public static void main(String[] args) {
              System.out.println("Hello, World!");

    }
}
class HelloWorld {
    static{
        System.out.println("I am static block of HelloWorld");
    }

}
```

```
PS D:\java_pr> java CheckStatic
I am static block of Checkstatic
Hello, World!
```

Example: Static block in two classes

```
class CheckStatic{
    static{
        System.out.println("I am static block of Checkstatic");
    }
       public static void main(String[] args) {
              System.out.println("Hello, World!");
              HelloWorld hw=new HelloWorld();
    }
}
class HelloWorld {
    static{
        System.out.println("I am static block of HelloWorld");
    }

}
```

```
PS D:\java_pr> java CheckStatic
I am static block of Checkstatic
Hello, World!
I am static block of HelloWorld
```

Garbage Collection

- Garbage collection done automatically in java.
- When no reference to an object exist, that object is assumed to be no loner needed, and the memory occupied by the object can be reclaimed.
- Garbage collection only occurs at regular intervals during the execution of your program.
- We can run garbage collection on demand by calling the **gc()** method.
- **public static void gc():** Initiates the garbage collection.

System.gc();

```
public class GarbageCollector{
    public static void main(String[] args) {
        int SIZE = 200;
        StringBuffer s;
        for (int i = 0; i < SIZE; i++) {
        }
        System.out.println("Garbage Collection started explicitly.");
        System.gc();
    }
}
```

finalize( ) method

- Sometimes an object will need to perform some action when it is destroyed.

  Ex:

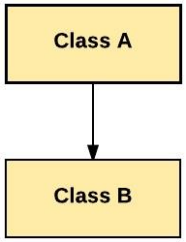
  If an object is holding some non-java resource such as a file, then you might want to make sure these resources are freed before an object is destroyed.

- To handle such situations, Java provides a mechanism called *finalization*.

- **The finalize( ) method has this general form:**

```
protected void finalize( ){
        // finalization code here
}
```

Inheritance



- **Inheritance** is a mechanism in which one class acquires the property of another class.
  - E.g. a child inherits the traits of his/her parents.
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

WHY INHERITANCE?

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

Key Terms:

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

Extends

- The **extends keyword** indicates that you are making a new class that derives from an existing class.

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

Example:

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

O/P:
Programmer salary is:40000.0
Bonus of programmer is:10000

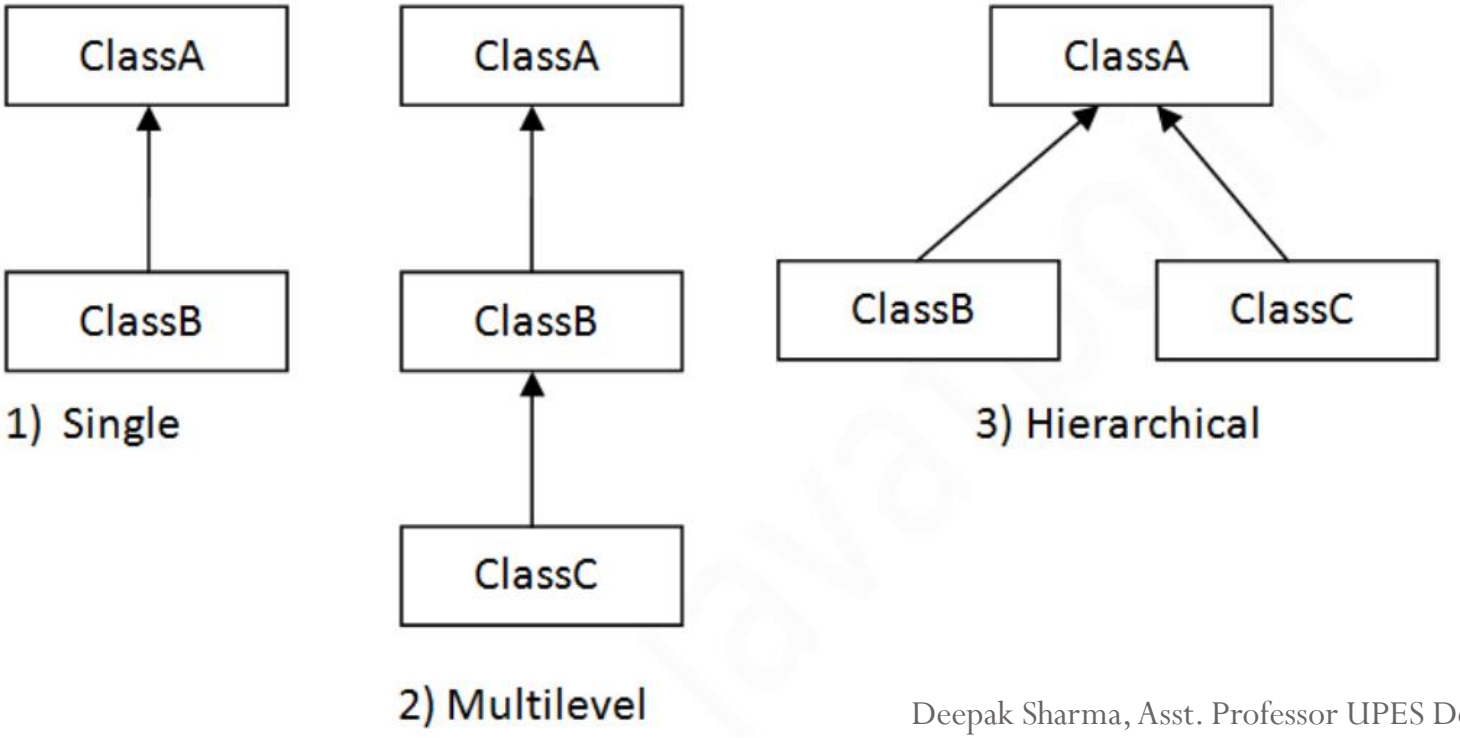
Example:

```
class Vehicle {
  protected String brand = "TATA";
  public void honk() {
    System.out.println("Horn PLz!");
  }
}
class Car extends Vehicle {
  private String modelName = "NEXON";
  public static void main(String[] args) {
    Car myFastCar = new Car();
    myFastCar.honk();
    System.out.println(myFastCar.brand + " " + myFastCar.modelName);
  }
}
```

O/P:
Horn PLz!
TATA NEXON

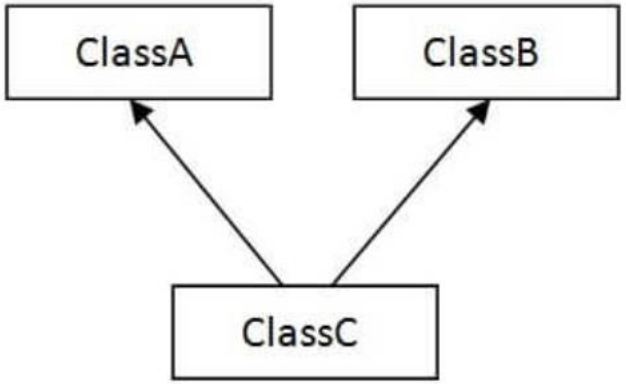
Types of inheritance in java

- On the basis of class, there can be three types of inheritance in java:
  - Single
  - multilevel
  - hierarchical

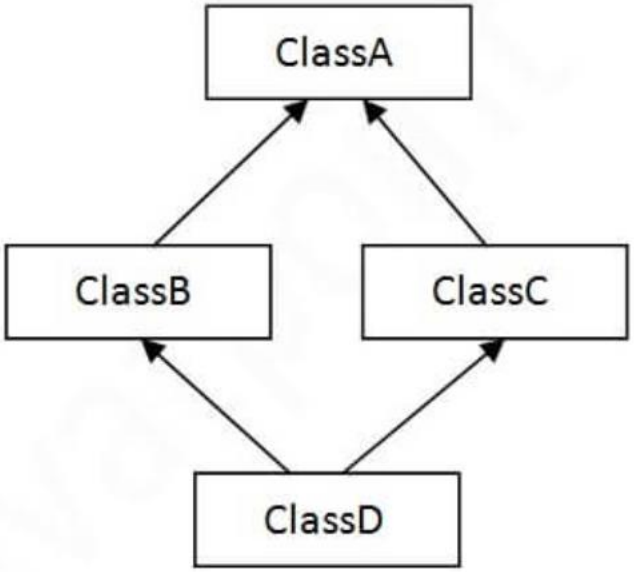


Types of Inheritance

- In java programming, *multiple and hybrid inheritance* is supported through interface only.



4) Multiple



5) Hybrid

Note: Multiple inheritance is not supported in Java through class.

Single Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}
}

class Dog extends Animal{
void bark(){System.out.println("barking...");}
}

class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

O/P:
barking...
eating...

Multilevel Inheritance Example

- When there is a chain of inheritance, it is known as *multilevel inheritance*.

```
class Animal{
void eat(){System.out.println("eating...");}
}

class Dog extends Animal{
void bark(){System.out.println("barking...");}
}

class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}

class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

O/P:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance Example

- When two or more classes inherits a single class.

```
class Animal{
void eat(){System.out.println("eating...");}
}

class Dog extends Animal{
void bark(){System.out.println("barking...");}
}

class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}

class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:
meowing...
eating…

Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- To avoid ambiguity while calling same method present in both parent classes.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Output: Compile time Error

Aggregation in Java

- If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship (part of relationship).

```
class Employee{
int id;
String name;
Address address;//Address is a class
…
}
```

- Employee has an entity reference address, so relationship is Employee HAS-A address.

Example of Aggregation

```
class Operation{
 int square(int n){
  return n*n;
 }
}
class Circle{
 Operation op;//aggregation
 double pi=3.14;
 double area(int radius){
   op=new Operation();
   int rsquare=op.square(radius);//code reusability (i.e. delegates the method call)
   return pi*rsquare;
 }

 public static void main(String args[]){
   Circle c=new Circle();
   double result=c.area(5);
   System.out.println(result);
 }
}
```

When use Aggregation?

- Code reuse is also best achieved by aggregation when there is no is-a relationship.
- Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Method Overloading and Overriding

Method Overloading

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- Method overloading *increases the readability of the program*.
- There are two ways to overload the method in java
  - By changing number of arguments
  - By changing the data type
- In Java, Method Overloading is not possible by changing the return type of the method only.

//method overloading

```
class OverloadDemo {
    void test() {
        System.out.println("No parameters");
    }
    void test(int a) {
        System.out.println("a: " + a);
    }
    void test(int a, int b) {
        System.out.println("a and b: " + a + " " + b);
    }
    double test(double a) {
        System.out.println("double a: " + a);
    return a*a;
    }
}
```

```
class Overload {
        public static void main(String args[]) {
                OverloadDemo ob = new OverloadDemo();
                double result;
                ob.test();
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(123.25);
                System.out.println("Result of ob.test(123.25): " + result);
}       }
```

Can we overload java main() method?

- Yes, by method overloading. You can have any number of main methods in a class by method overloading.
- But JVM calls main() method which receives string array as arguments only.

```
class TestOverloading{
public static void main(String[] args){System.out.println("main with String[]");}
public static void main(String args){System.out.println("main with String");}
public static void main(){System.out.println("main without args");}
}
```

Output: main with String[]

Method Overriding in Java

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Problem without method overriding

```
class Vehicle{
  void run(){System.out.println("Vehicle is running");}
}

//Creating a child class
class Bike extends Vehicle{
  public static void main(String args[]){
  //creating an instance of child class
  Bike obj = new Bike();
  //calling the method with child class instance
  obj.run();
  }
}
```

Output: Vehicle is running

Problem: Need to provide a specific implementation of run() method in subclass

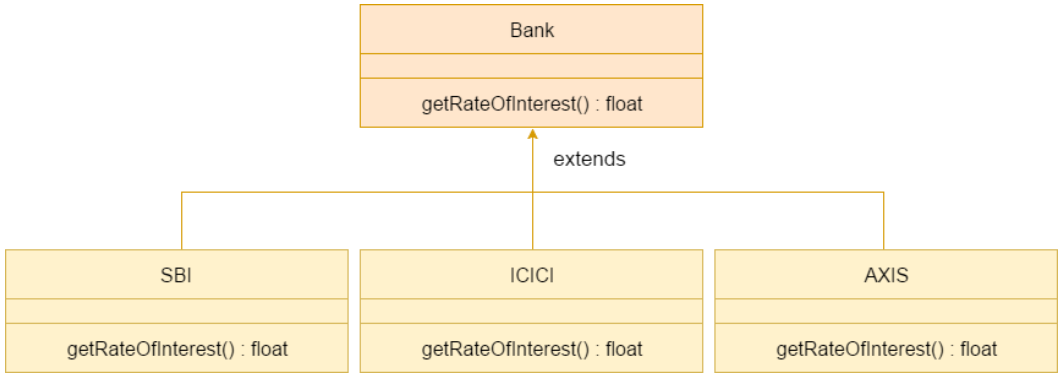
Example: method overriding

```
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");
}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Output: Bike is running safely

Example: Method Overriding

```
class Bank{
int getRateOfInterest(){return 0;}
}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}

class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
//Test class to create objects and call the methods
class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}
```



Output:

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Can we override static method?

- No, a static method cannot be overridden.
- the static method is bound with class whereas instance method is bound with an object.

Can we override java main method?

- No, because the main is a static method.

Super Keyword

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate **parent class instance variable**.
2. super can be used to invoke immediate **parent class method**.
3. super() can be used to invoke immediate **parent class constructor**.

Super- to refer immediate parent class instance variable

- to access the data member or field of parent class.
- if parent class and child class have same fields.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

Output:
Black
white

eating... barking...

Super- to invoke parent class method

- It should be used if subclass contains the same method as parent class (Methods are overridden).

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class TestSuper{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

Output:
eating…
barking…

Super() - to invoke parent class constructor

- To invoke parent class constructor.

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Output:
animal is created
dog is created

eating... barking...

Super() - to invoke parent class constructor

- Note: super() is added in each class constructor automatically as the first statement by compiler if there is no super() or this().

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
System.out.println("dog is created");
}
}
class TestSuper4{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Output:
animal is created
dog is created

Example- Super()

```
class Person{
int id;
String name;
Person(int id,String name){
this.id=id;
this.name=name;
}
}
class Emp extends Person{
float salary;
Emp(int id,String name,float salary){
super(id,name);//reusing parent constructor
this.salary=salary;
}
void display(){System.out.println(id+" "+name+" "+salary);}
}
class TestSuper5{
public static void main(String[] args){
Emp e1=new Emp(4,"aman",95000f);
e1.display();
}}
```

Output:
4 aman 95000

Instance initializer block- Example

- **Instance Initializer block** is used to initialize the instance data member.
- It run each time when object of the class is created.

```
class Bike{
    int speed;

    Bike(){System.out.println("speed is "+speed);}

    {speed=100;}

    public static void main(String args[]){
    Bike b1=new Bike();
    Bike b2=new Bike();
    }
}
```

Output:
speed is 100
speed is 100

Instance initializer block

- It run each time when object of the class is created.

We can directly assign a value in instance data member

```
class Bike{
   int speed=100;
}
```

Why initializer block?

- To perform some operations while assigning value to instance data member
  - e.g. a for loop to fill a complex array or error handling etc.

There are three places in java where you can perform operations:

- method
- constructor
- block

Instance initializer block

What is invoked first, instance initializer block or constructor?

```
class Bike8{
   int speed;

   Bike8(){System.out.println("constructor is invoked");}

   {System.out.println("instance initializer block invoked");}

   public static void main(String args[]){
   Bike8 b1=new Bike8();
   Bike8 b2=new Bike8();
   }
}
```

Output:

```
instance initializer block invoked
constructor is invoked
instance initializer block invoked
constructor is invoked
```

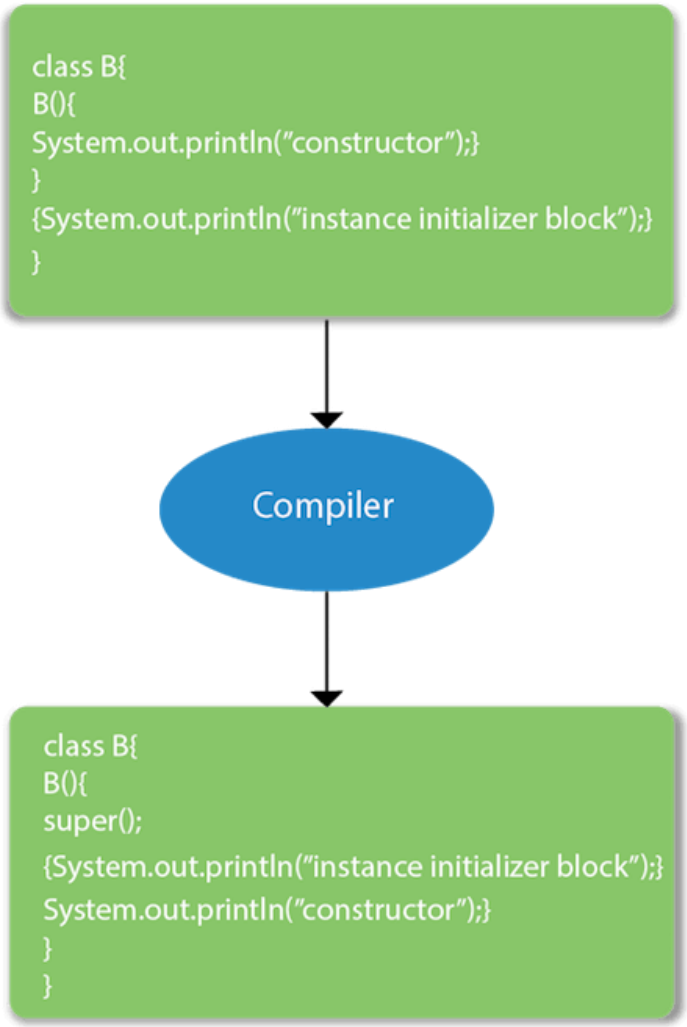
- It seems that instance initializer block is firstly invoked but NO.
- Instance intializer block is invoked at the time of object creation.

Note: The java compiler copies the code of instance initializer block in every constructor.

Rules for instance initializer block :

Three rules:

- The instance initializer block is created when instance of the class is created.
- The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
- The instance initializer block comes in the order in which they appear.



Program of instance initializer block that is invoked after super()

```
class A{
A(){
System.out.println("parent class constructor invoked");
}
}
class B2 extends A{
B2(){
super();
System.out.println("child class constructor invoked");
}

{System.out.println("instance initializer block is invoked");}

public static void main(String args[]){
B2 b=new B2();
}
}
```

Output:

```
parent class constructor invoked
instance initializer block is invoked
child class constructor invoked
```

Checkpoint...

```
class A{
A(){
System.out.println("parent class constructor invoked");
} }


class B3 extends A{
B3(){
super();
System.out.println("child class constructor invoked");
}

B3(int a){
super();
System.out.println("child class constructor invoked "+a);
}

{System.out.println("instance initializer block is invoked");}
public static void main(String args[]){
B3 b1=new B3();
B3 b2=new B3(10);
} }
```

```
parent class constructor invoked
instance initializer block is invoked
child class constructor invoked
parent class constructor invoked
instance initializer block is invoked
child class constructor invoked 10
```