Instance initializer block

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

The initialization of the instance variable can be done directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.

Que) What is the use of instance initializer block while we can directly assign a value in instance data member? For example:

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

Why use instance initializer block?

Suppose I have 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.

Example of instance initializer block

Let's see the simple example of instance initializer block that performs initialization.

```
class Bike7{
  int speed;

  Bike7(){System.out.println("speed is "+speed);}

  {speed=100;}

  public static void main(String args[]){
  Bike7 b1=new Bike7();
  Bike7 b2=new Bike7();
  }
}
```

Test it Now

```
Output:speed is 100
       speed is 100
```

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

1. method
2. constructor
3. block

What is invoked first, instance initializer block or constructor?

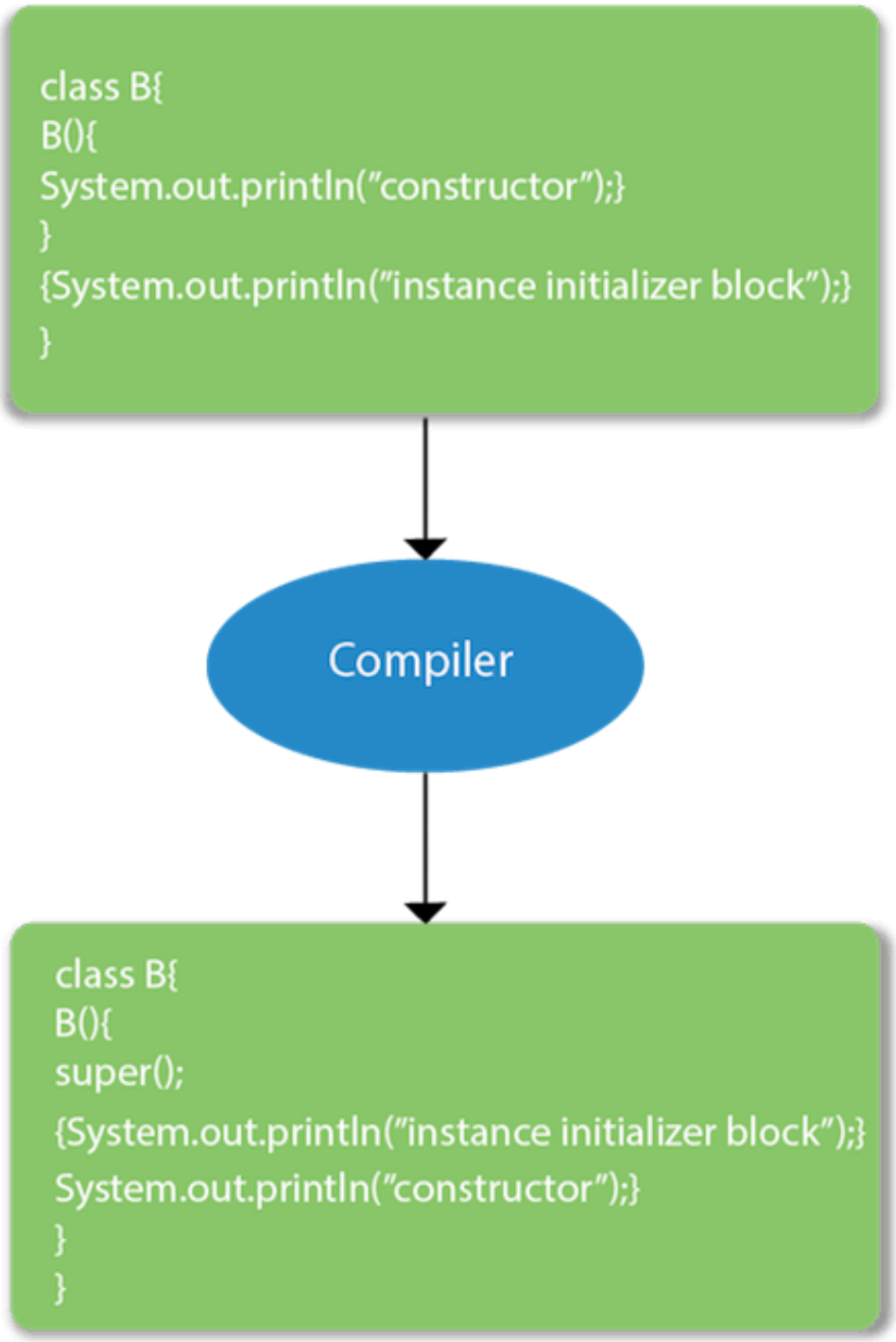
```
class Bike8{
  int speed;

  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();
  }
}
```

Test it Now

```
Output:instance initializer block invoked
       constructor is invoked
       instance initializer block invoked
       constructor is invoked
```

In the above example, it seems that instance initializer block is firstly invoked but NO. Instance intializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super(). So firstly, constructor is invoked. Let's understand it by the figure given below:

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



Rules for instance initializer block :

There are mainly three rules for the instance initializer block. They are as follows:

1. The instance initializer block is created when instance of the class is created.
2. The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
3. 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();
}
}
```

Test it Now

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

Another example of instance block

```
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);
}
}
```

Test it Now

```
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
```