Method return type as a class :

* As we know while defining a method in java, return type is compulsory.

* As a return type of method, we have various options are available :

  1) void keyword [not to return any kind of value]
  2) All primitive data type
  3) We can also take class name as a method return type.

Note : Whenever we take class name as a return type of the method then internally we are depending upon the class constructor.

Different cases for Method return types :

Case 1 :

```
void accept()   //Method is not returning any kind of value [No return type]
{
}
```

Case 2 :

```
int accept()
{
   return 2.3;  //Invalid because the value 2.3 is not compaitable with int return type
}
```

Case 3 :

```
int accept()
{
   return 10;   /* Valid  The return value (10) of the method must be compaitable with return type (int) of
}                  the method OR in order words we can say we can assign the value 10 (return value of
                   the method) to return type of the method (int type) via a variable.*/

                              int x = 10;
```

Case 4 :

```
public class Test
{
   public Test accept()   //Test t1 = null;
   {
      return null;
   }
}
```

Case 4 :

```
public class Test
{
   public Test accept()   //Test t1 = new Test();
   {
      Test t1 = new Test();
       return t1;
   }
}
```

Case 5 :

```
public class Test
{
   public Test accept()   //Test t1 = new Test();
   {
       return new Test();
   }
}
```

Case 6:

```
public class Test
{
   public Test accept()
   {
       return  this;
   }
}
```

Small program on method return type as a class :

```
public class Test
{
     Test accept()
     {
          //return null;

          //return new Test();

          return this;
     }
}
```

```
public class Test
{
   public Test(int x){}

     Test accept()
     {
          return new Test(9); //depending on constructor
     }
}
```

What is a static factory Method :

* A static factory method is special static method which is used to **create and return** the object of the class instead of using constructor directly.

* It prevent us to create the object from outside of the BLC class that means outsider cannot create the object because constructor is private [Singleton design pattern]

  [Note : It is not Factory design pattern, Factory design pattern we can achieve using interface concept]

//Program on static factory method :

```
package com.nit.static_factory_method;

public class StaticFactoryMethodDemo1
{
    public static void main(String[] args)
    {
       Book book = Book.getBookObject();
       IO.print(book);
    }
}

class Book
{
    private String title;
    private String author;
    private double price;


    private Book(String title, String author, double price)
    {
        super();
        this.title = title;
        this.author = author;
        this.price = price;
    }


    @Override
    public String toString()
    {
        return "Book [title=" + title + ", author=" + author + ", price=" + price + "]";
    }

    //static factory method
    public static Book getBookObject()
    {
        Book book = new Book("Java", "James Gosling", 1200);
        return book;
    }
}
```

Note : In this program we are creating and returning Book object by using static factory method but we are not re-using the method.

```
package com.nit.static_factory_method;

public class StaticFactoryMethodDemo2
{
    public static void main(String[] args)
    {
        int noOfObject = Integer.parseInt(IO.readln("Enter the number of objects :"));

        for(int i=1; i<=noOfObject; i++)
        {
            int id = Integer.parseInt(IO.readln("Enter Product id :"));
            String name = IO.readln("Enter Product Name :");
            double price = Double.parseDouble(IO.readln("Enter Product price :"));
            Product product = Product.getProductObject(id, name, price);
            IO.println(product);
        }
    }
}
//BLC
class Product
{
    private int id;
    private String name;
    private double price;

    private Product(int id, String name, double price)
    {
        super();
        this.id = id;
        this.name = name;
        this.price = price;
    }

    @Override
    public String toString()
    {
        return "Product [id=" + id + ", name=" + name + ", price=" + price + "]";
    }


    //static factory method
    public static Product getProductObject(int id, String name, double price)
    {
        return new Product(id, name, price);
    }

}
```

What is Garbage Collector in Java ?

Java

C, C++

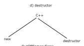
Java

a) Pointers

b) Multiple Inheritance

c) :: (No Global Variable)

d) destructor



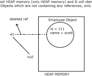
* In older language like C++, A programmer is responsible to **allocate and de-allocate** the memory otherwise there is a chance of getting OutOfMemoryError.

* Java languge provides **automatic memory management through Garbage Collector.**

* A Garbage Collector is a **deamon thread which is running in the background of java program to delete the un-used objects.**

* On the request of JVM, GC (Garbage Collector) will visit HEAP memory (only HEAP memory) and It will identify which objects are eligible for Garbage Collector, The Objects which are not containing any references, only those objects are eligible for Garbage Collector.
  Example :

```
Employee e1 = new Employee(111, "Scott");
e1 = null;
```



* Internally GC is using an algorithm "Mark and Sweep" and "Mark and Compact" to delete the un-used objects.

* Garbage Collector will visit the HEAP memory on the request of JVM but as a pogrammer, We can also explicitly call GC by using the gc() static method of System class.
  Example :
  System.gc();

How many ways we can make an object eligible for GC :