Java Bean

A Java Bean is a java class that should follow following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use Java Bean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance.

Simple example of java bean class

```
//Employee.java

package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;

public Employee(){}

public void setId(int id){this.id=id;}

public int getId(){return id;}

public void setName(String name){this.name=name;}

public String getName(){return name;}

}
```

How to access the java bean class?

To access the java bean class, we should use getter and setter methods.

```
package mypack;
public class Test{
public static void main(String args[]){

Employee e=new Employee();//object is created

e.setName("Arjun");//setting value to the object

System.out.println(e.getName());

}}
```

Proxy server is an intermediary server between client and the interner. Proxy servers offers the following basic functionalities:

- Firewall and network data filtering.
- Network connection sharing
- Data caching

Proxy servers allow to hide, conceal and make your network id anonymous by hiding your IP address.

Purpose of Proxy Servers

Following are the reasons to use proxy servers:

- Monitoring and Filtering
- Improving performance
- Translation
- Accessing services anonymously
- Security

Monitoring and Filtering

Proxy servers allow us to do several kind of filtering such as:

- Content Filtering
- Filttering encrypted data
- Bypass filters
- Logging and eavasdropping

Improving performance

It fasten the service by process of retrieving content from the cache which was saved when previous request was made by the client.

Transalation

It helps to customize the source site for local users by excluding source content or substituting source content with original local content. In this the traffic from the global users is routed to the source website through Translation proxy.

Accessing services anonymously

In this the destination server receives the request from the anonymzing proxy server and thus does not receive information about the end user.

Security

Since the proxy server hides the identity of the user hence it protects from spam and the hacker attacks.

"writing output on console" using PrintWriter() method

```
import java.io.*;

public class PrintWriterExample
{
   public static void main(String args[]) throws IOException
   {
      int [] id = {12456, 323, 345, 3423};
      String [] firstName = {"John", "Paul", "George", "Ringo"};
      String [] lastName = {"Lennon", "McCartney", "Harrison", "Star"};

      PrintWriter outFile = new PrintWriter(System.out, true);
      String format = "ID: %5d (%s, %s)\n";

         for (int i=0; i<id.length; i++)
         {
            outFile.printf(format, id[i], firstName[i], lastName[i]);
         }
   }
}
```

Here is the output:

```
ID: 12456 (John, Lennon)
ID:   323 (Paul, McCartney)
ID:   345 (George, Harrison)
ID:  3423 (Ringo, Star)
```

.

finalize method

It is a **method** that the **Garbage Collector** always calls just **before** the deletion/destroying the object which is eligible for Garbage Collection, so as to perform **clean-up activity**. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection or we can say resource de-allocation. Remember it is **not** a reserved keyword. Once finalize method completes immediately Garbage Collector destroy that object. finalize method is present in Object class and its syntax is:

```
protected void finalize throws Throwable{}
```

Since Object class contains finalize method hence finalize method is available for every java class since Object is superclass of all java classes. Since it is available for every java class hence Garbage Collector can call finalize method on **any java object**

Now, the finalize method which is present in Object class, has empty implementation, in our class clean-up activities are there, then we have to **override** this method to define our own clean-up activities.
Cases related to finalize method:

1. **Case 1 :** The object which is eligible for Garbage Collection, that object's corresponding class finalize method is going to be executed

```
class Hello {
    public static void main(String[] args)
    {
        String s = new String("RR");
        s = null;

        // Requesting JVM to call Garbage Collector method
        System.gc();
        System.out.println("Main Completes");
    }

    // Here overriding finalize method
    public void finalize()
    {
        System.out.println("finalize method overriden");
    }
}
```

2. Run on IDE
3. **Output**:

4. Main Completes

5. **Note** : Here above output came only **Main Completes** and **not** "finalize method overriden" because Garbage Collector call finalize method on that class object which is eligible for Garbage collection. Here above we have done->
   s = null and 's' is the object of String class, so String class finalize method is going to be called and not our class(i.e, Hello class). So we modify our code like->

6. **Hello s = new Hello()**;
7. **s = null**;

8. Now our class i.e, Hello class finalize method is called. **Output**:

9. finalize method overriden
10. Main Completes

11. So basically, Garbage Collector calls finalize method on that class object which is eligible for Garbage collection.So if String object is eligible for

Garbage Collection then **String** class finalize method is going to be called and **not the Hello class** finalize method.

12. **Case 2 :** We can call finalize method Explicitly then it will be executed just like normal method call but object won't be deleted/destroyed

```
class Bye {
    public static void main(String[] args)
    {
        Bye m = new Bye();

        // Calling finalize method Explicitly.
        m.finalize();
        m.finalize();
        m = null;

        // Requesting JVM to call Garbage Collector method
        System.gc();
        System.out.println("Main Completes");
    }

    // Here overriding finalize method
    public void finalize()
    {
        System.out.println("finalize method overriden");
    }
}
```

13. Run on IDE
14. **Output**:

```
finalize method overriden
//call by programmer but object won't gets destroyed.
finalize method overriden
//call by programmer but object won't gets destroyed.
Main Completes
finalize method overriden
//call by Garbage Collector just before destroying the object.
```

22. **Note** : As finalize is a method and not a reserved keyword, so we can call finalize method **Explicitly**, then it will be executed just like normal method call but object won't be deleted/destroyed.
23. **Case 3 :**
    - **Part a)** If programmer calls finalize method, while executing finalize method some unchecked exception rises.

```
class Hi {
```

```
    public static void main(String[] args)
    {
        Hi j = new Hi();

        // Calling finalize method Explicitly.
        j.finalize();

        j = null;

        // Requesting JVM to call Garbage Collector method
        System.gc();
        System.out.println("Main Completes");
    }

    // Here overriding finalize method
    public void finalize()
    {
        System.out.println("finalize method overriden");
        System.out.println(10 / 0);
    }
}
```

- Run on IDE
- **Output**:

```
exception in thread "main" java.lang.ArithmeticException:
/ by zero followed by stack trace.
```

- So **key point** is : If programmer calls finalize method, while executing finalize method some unchecked exception rises, then JVM terminates the program abnormally by rising exception. So in this case the program termination is **Abnormal**.
- **part b)** If garbage Collector calls finalize method, while executing finalize method some unchecked exception rises.

```
class RR {
    public static void main(String[] args)
    {
        RR q = new RR();
        q = null;

        // Requesting JVM to call Garbage Collector method
        System.gc();
        System.out.println("Main Completes");
    }

    // Here overriding finalize method
```

```
    public void finalize()
    {
        System.out.println("finalize method overriden");
        System.out.println(10 / 0);
    }
}
```

- Run on IDE
- **Output**:

```
Main Completes
finalize method overriden
```

- So **key point** is : If Garbage Collector calls finalize method, while executing finalize method some unchecked exception rises then JVM **ignores** that exception and rest of program will be continued normally. So in this case the program termination is **Normal** and not abnormal.

Important points:

- There is no guarantee about the time when finalize is called. It may be called any time after the object is not being referred anywhere (cab be garbage collected).
- JVM does not ignore all exceptions while executing finalize method, but it ignores **only Unchecked exceptions**. If the corresponding catch block is there then JVM won't ignore and corresponding catch block will be executed.
- System.gc() is just a request to JVM to execute the Garbage Collector. It's up-to JVM to call Garbage Collector or not.Usually JVM calls Garbage Collector when there is not enough space available in the Heap area or when the memory is low.