DAILY ASSIGNMENT – 11/08/2021

Topics:-

1.Thread

Threads allows a program to operate more efficiently by doing multiple things at the same time. Threads can be used to perform complicated tasks in the background without interrupting the main program.

Java Thread Example by extending Thread class

```
Class Test extends Thread{
 public void run()
{
  System.out.println("thread is running...");
}
  public static void main(String args[]){
  Test obj = new Test();
  obj. start();
 }
}
```

Java Thread Example by implementing Runnable interface

```
Class Test implements Runnable{
public void run(){
System.out.println("thread is running...");
}
   public static void main(String args[]){
   Test m1=new Test();
```

```
 Thread obj =new Thread(m1);
 obj.start();
 }
}
```

-:Runnable Class:-

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

public void run(): is used to perform action for a thread.

Java inner class

- Java inner class or nested class is a class that is declared inside the class or interface.
- We use inner classes to logically group classes and interfaces in one place to be more
- readable and maintainable.
- Additionally, it can access all the members of the outer class, including private data
- members and methods.

Need of Java Inner class

- Sometimes users need to program a class in such a way so that no other class can access,

  Therefore, it would be better if you include it within other classes.

- If all the class objects are a part of the outer object, then it is easier to nest that

  class inside the outer

- class. That way all the outer class can access all the objects of the inner class.

2.Serialization

Serialization in Java is a mechanism of writing the state of an object into a byte-stream.

It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

- For serializing the object, we call the writeObject() method *of* ObjectOutputStream class

- We must have to implement the *Serializable* interface for serializing the object.
  E.g.- EmployeeDetails

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

public class Test2 {

  public static void main(String[] args) {

        ObjectOutputStream oos =
        new ObjectOutputStream(new FileOutputStream(new File("employee.txt") ));

        Employee emp = new Employee();
        emp.setId(100);
        emp.setName("java");

        oos.close();
        System.out.println("Completed");
```

```
        ObjectInputStream ois = new ObjectInputStream
         (new FileInputStream(new File("employee.txt")));
        Employee obj = (Employee) ois.readObject();
        System.out.println(obj.equals(emp));

}


Class Employee implements Serializable {

   private int id;
   private String name;

   public Employee() {
   }

   public Employee(int id, String name) {
      super();
      this.id = id;
      this.name = name;
   }

  public int getId() {
      return id;
   }

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

   public String getName() {
      return name;
   }
```

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