OBJECT ORIENTED PROGRAMMING WEEK 5

20 FEB-24 FEB, 2023

Instructor:

Abdul Aziz
Assistant Professor
(School of Computing)
National University- FAST (KHI Campus)

ACKNOWLEDGMENT

- Publish material by Virtual University of Pakistan.

- Publish material by Deitel & Deitel.

- Publish material by Robert Lafore.

INSTANCE INITIALIZER BLOCK

- The purpose of the instance initializer block is to initialize the instance data members.
- The instance initializer block looks just like the static initializer block, but without the static keyword
- The instance block runs at the time of instance creation.
- Static initializer blocks always execute before the instance initialization blocks

EXAMPLE

```
public class IBlock {
{
    System.out.println("Instance initializer block 1");
}
{
    System.out.println("Instance initializer block 2");
}
public IBlock () {
    System.out.println("Class constructor");
}
public static void main(String[] args) {
    IBlock ib = new IBlock();
    System.out.println("Main Method");
}
}
```

The `final` Keyword

- **Class**
  - You cannot subclass a `final` class.
- **Method**
  - You cannot override a `final` method.
- **Variable**
  - A `final` variable is a constant.
  - You can set a `final` variable only once.
  - Assignment can occur independently of the declaration (*blank final variable*).

Blank Final Variables

```
public class Employee{
    private final long ID;

    public Employee(){
      ID = createID();
    }

    private long createID(){
       //return the generated ID
    }
    …
 }
```

REMEMBER

CONSTANT INSTANCE DATA

`final`

```
public class Product{
        private final int ID;
}
```

REMEMBER

STATIC DATA

static

```
public class Product{
    private final int ID;
    private static counter;
    public Product(){
        ID = ++counter;
    }
}
```

REMEMBER

CONSTANT STATIC DATA

`static final`

```
public class Product{
   private final int ID;
   private static counter;
   private static final String name = "PRODUCT";
   public Product(){
      ID = ++counter;
   }

   public String getIDStr(){
     return name+ID;
   }
}
```

DESTRUCTOR

- It is a special method that automatically gets called when an object is no longer used.
- When an object completes its life-cycle the garbage collector deletes that object and deallocates or releases the memory occupied by the object.
- It releases the resources occupied by the object.
- No explicit call is required, it is automatically invoked at the end of the program execution.
- It does not accept any parameter and cannot have multiple destructors.

SYNTAX

```
protected void finalize throws Throwable()
{
//resources to be close
}
```

- It is a protected method of the Object class that is defined in the java.lang package.
- It can be called only once.
- We need to call the finalize() method explicitly if we want to override the method.
- The gc() is a method of JVM executed by the Garbage Collector. It invokes when the heap memory is full and requires more memory for new arriving objects.
- Except for the unchecked exceptions, the JVM ignores all the exceptions that occur by the finalize() method.

```
public class DestructorExample
{
          protected void finalize()
          {
          System.out.println("Object is destroyed by the Garbage Collector");
          }
public static void main(String[] args)
          {
          DestructorExample de = new DestructorExample ();
          de.finalize();
          de = null;
          System.gc();
          System.out.println("Inside the main() method");
          }

}
```

System.runFinalizersOnExit(true).

```
enum TrafficSignal
{
    RED("STOP"), GREEN("GO"), ORANGE("SLOW DOWN");

    private String action;

    private TrafficSignal(String action)
    {
        this.action = action;
    }

    public String getAction()
    {
        return this.action;
    }

 }
```

```java
public class EnumConstructorExample
{
    public static void main(String args[])
    {
        // let's print name of each enum and there action
        // - Enum values() examples
        TrafficSignal[] signals = TrafficSignal.values();

        for (TrafficSignal signal : signals)
        {
            // use getter method to get the value
            System.out.println("name : " + signal.name() +
                        " action: " + signal.getAction() );
        }
    }
}
```

Output:

```
name : RED action: STOP
name : GREEN action: GO
name : ORANGE action: SLOW DOWN
```

```
public enum GestureType {
    UP,
    RIGHT,
    DOWN,
    LEFT
}

-----------------------------------------------------------

for(GestureType type: GestureType.values()){
    System.out.println( type );
}
```

OUTPUT:

```
UP
RIGHT
DOWN
LEFT
```

```
public enum GestureType {
   UP (0, "fel"),
   RIGHT (1, "jobb"),
   DOWN (2, "le"),
   LEFT (3, "bal");

   GestureType( int value, String name ){
       this.value = value;
       this.name = name;
   }

   public int getValue(){
       return value;
   }

   public String getName(){
       return name;
   }

   private int value;
   private String name;
}
```

Enumerations

```
for(GestureType type: GestureType.values()){
    System.out.println(type.name()+", "+
                       type.getName()+", "+ type.getValue());
}
```

Output

```
UP, fel, 0
RIGHT, jobb, 1
DOWN, le, 2
LEFT, bal, 3
```