Java – Mutable and Immutable Objects

1. **Mutable object** – You can change the states and fields after the object is created. For examples: StringBuilder, java.util.Date and etc.

2. **Immutable object** – You cannot change anything after the object is created. For examples: String, boxed primitive objects like Integer, Long and etc.

## Java Mutable Example

Normally, it provides a method to modify the field value, and the object can be exten

public class MutableExample {

private String name;

MutableExample(String name) {

this.name = name;

}

public String getName() {

return name;

}

// this setter can modify the name

public void setName(String name) {

this.name = name;

}

public static void main(String[] args) {

MutableExample obj = new MutableExample("3Edge");

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

// update the name, this object is mutable

obj.setName("Solutions ");

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

}

}

## Immutable

To create an Immutable object, make the class final, and don’t provide any methods to modify the fields.

An object is immutable if its state cannot change after construction. Immutable objects don’t expose any way for other objects to modify their state; the object’s fields are initialized only once inside the constructor and never change again.

With immutable classes, states are never modified; every modification of a state results in a new instance, hence each thread would use a different instance and developers wouldn’t worry about concurrent modifications.

**How Do We Create an Immutable Class**

In order to create an immutable class, you should follow the below steps:

1. Make your class ***final,***so that no other classes can extend it.
2. Make all your fields ***final,***so that they’re initialized only once inside the constructor and never modified afterward.
3. Don’t expose setter methods.
4. When exposing methods which modify the state of the class, you must always return a new instance of the class.
5. If the class holds a mutable object:
   * Inside the constructor, make sure to use a clone copy of the passed argument and never set your mutable field to the real instance passed through constructor, this is to prevent the clients who pass the object from modifying it afterwards.
   * Make sure to always return a clone copy of the field and never return the real object instance.

public final class ImmutableExample {

private String name;

ImmutableExample (String name) {

this.name = name;

}

public String getName() {

return name;

}

//no setter

public static void main(String[] args) {

ImmutableExample obj = new ImmutableExample("3Edge");

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

// there is no way to update the name after the object is created.

// obj.setName("new 3Edge");

// System.out.println(obj.getName());

}

}