

Java – Mutable and Immutable Objects

By mkyong (<http://www.mkyong.com/author/mkyong/>) | February 5, 2016 | Updated : December 5, 2016 | Viewed : 354 times +37 pv/w

This article shows you the difference between Mutable and Immutable objects in Java

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.

1. Java Mutable Example

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

MutableExample.java

```
package com.mkyong;

public class MutableExample {

    private String name;

    MutableClass(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("mkyong");
        System.out.println(obj.getName());

        // update the name, this object is mutable
        obj.setName("new mkyong");
        System.out.println(obj.getName());

    }
}
```

Output

```
mkyong
new mkyong
```

2. Java Immutable Example

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

ImmutableExample.java

```

package com.mkyong;

// make this class final, no one can extend this class
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("mkyong");
        System.out.println(obj.getName());

        // there is no way to update the name after the object is created.
        // obj.setName("new mkyong");
        // System.out.println(obj.getName());

    }
}

```

Output

mkyong

Note

Immutable object is simple, thread-safe (no need synchronization), less prone to error and more secure. If possible, make all objects immutable.

P.S Please refer to the Effective Java Book – Item 15: Minimize mutability.

References

1. Mutable vs immutable objects (<http://stackoverflow.com/questions/214714/mutable-vs-immutable-objects>)
2. Immutable objects as per java docs (<https://docs.oracle.com/javase/tutorial/essential/concurrency/immutable.html>)

Tags : [immutable \(http://www.mkyong.com/tag/immutable/\)](http://www.mkyong.com/tag/immutable/) [java \(http://www.mkyong.com/tag/java/\)](http://www.mkyong.com/tag/java/) [mutable \(http://www.mkyong.com/tag/mutable/\)](http://www.mkyong.com/tag/mutable/)

Share this article on

Twitter (<https://twitter.com/intent/tweet?text=Java - Mutable and Immutable Objects&url=http://www.mkyong.com/java/java-mutable-and-immutable-objects/&via=mkyong>) Facebook (<https://www.facebook.com/sharer/sharer.php?u=http://www.mkyong.com/java/java-mutable-and-immutable-objects/>) Google+ (<https://plus.google.com/share?url=http://www.mkyong.com/java/java-mutable-and-immutable-objects/>)

TEN TRIVIA QUESTIONS TO TEST YOUR ANDROID FANDOM



What was Android originally developed as a platform for?

15



- A Laptops
- B In-flight services
- C Digital cameras
- D Mobile Phones

1 2 3 4 5 6 |

About the Author