**Create immutable class in Java:**

An immutable class is one whose state cannot be changed once created.

Rules to create immutable classes

* Don’t provide “setter” methods — Setter methods are meant to change the state of an object
* Make all fields final and private - Fields declared private will not be accessible outside the class and making them final will ensure the even accidentally you cannot change them.
* Don’t allow subclasses to override methods - declare the class as final. Final classes in java cannot be overridden.
* Always instance variables will be either mutable or immutable. Identify them and return new objects with copied content for all mutable objects. Immutable variables can be returned safely without extra effort.
* A more sophisticated approach is to make the constructor private and construct instances in factory methods.

**Immutable classes in JDK:**

Apart from your written classes, JDK itself has lots of immutable classes. Given is such a list of immutable classes in Java.

* String
* Wrapper classes such as Integer, Long, Double etc.
* Immutable collection classes such as Collections.singletonMap() etc.
* java.lang.StackTraceElement
* Java enums (ideally they should be)
* java.util.Locale
* java.util.UUID

**Benefits of making a class immutable:**

* are simple to construct, test, and use
* are automatically thread-safe and have no synchronization issues
* do not need a copy constructor
* do not need an implementation of clone
* allow hashCode() to use lazy initialization, and to cache its return value
* do not need to be copied defensively when used as a field
* make good Map keys and Set elements (these objects must not change state while in the collection)
* have their class invariant established once upon construction, and it never needs to be checked again.
* always have “failure atomicity”, if an immutable object throws an exception, it’s never left in an undesirable or indeterminate state.
* 2. Java immutable class example
* Lets apply all above rules for immutable classes and make a concrete class implementation for immutable class in Java.

ImmutableClass.java

import java.util.Date;

/\*\*

\* Always remember that your instance variables will be either mutable or immutable.

\* Identify them and return new objects with copied content for all mutable objects.

\* Immutable variables can be returned safely without extra effort.

\* \*/

public final class ImmutableClass

{

/\*\*

\* Integer class is immutable as it does not provide any setter to change its content

\* \*/

private final Integer immutableField1;

/\*\*

\* String class is immutable as it also does not provide setter to change its content

\* \*/

private final String immutableField2;

/\*\*

\* Date class is mutable as it provide setters to change various date/time parts

\* \*/

private final Date mutableField;

//Default private constructor will ensure no unplanned construction of class

private ImmutableClass(Integer fld1, String fld2, Date date)

{

this.immutableField1 = fld1;

this.immutableField2 = fld2;

this.mutableField = new Date(date.getTime());

}

//Factory method to store object creation logic in single place

public static ImmutableClass createNewInstance(Integer fld1, String fld2, Date date)

{

return new ImmutableClass(fld1, fld2, date);

}

//Provide no setter methods

/\*\*

\* Integer class is immutable so we can return the instance variable as it is

\* \*/

public Integer getImmutableField1() {

return immutableField1;

}

/\*\*

\* String class is also immutable so we can return the instance variable as it is

\* \*/

public String getImmutableField2() {

return immutableField2;

}

/\*\*

\* Date class is mutable so we need a little care here.

\* We should not return the reference of original instance variable.

\* Instead a new Date object, with content copied to it, should be returned.

\* \*/

public Date getMutableField() {

return new Date(mutableField.getTime());

}

@Override

public String toString() {

return immutableField1 +" - "+ immutableField2 +" - "+ mutableField;

}

}

Now its time to test our class:

TestMain.java

class TestMain

{

public static void main(String[] args)

{

ImmutableClass im = ImmutableClass.createNewInstance(100,"test", new Date());

System.out.println(im);

tryModification(im.getImmutableField1(),im.getImmutableField2(),im.getMutableField());

System.out.println(im);

}

private static void tryModification(Integer immutableField1, String immutableField2, Date mutableField)

{

immutableField1 = 10000;

immutableField2 = "test changed";

mutableField.setDate(10);

}

}

Program output:

Console

100 - test - Tue Oct 30 21:34:08 IST 2012

100 - test - Tue Oct 30 21:34:08 IST 2012

As it can be seen that even changing the instance variables using their references does not change their value, so the class is immutable.