Immutable class in Java

When an object of **immutable class** is created, the state or value of it cannot be changed , it means any modification on immutable object will result in a new immutable object. Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state.

Advantages of making an immutable class

- Immutable objects are thread safe and doesn't have synchronization issues.
- Does not need a copy constructor
- Less intermediate or local objects created.
- Immutability makes it easier to parallelize your program as there are no conflicts among objects.
- Act as keys for **Map** and values of **Set**
- final class cannot have sub class
- Another important benefit of Immutable objects is reusability, you can cache Immutable object and reuse them, much like String literals and Integers. You can use static factory methods to provide methods like valueOf(), which can return an existing Immutable object from cache, instead of creating a new one.

How to create an Immutable class ?

- Create a class with final keyword.
- Set the values of properties using constructor only.
- Make the properties of the class final and private
- Do not provide any setters for these properties.
- If the getters return an Object reference such as Collections, Arrays or other class object, clone the object before returning. Make sure the object is deep clone and not shallow.

Example

```
public final class Person {
    private final String name;
    private final int age;

    public FinalPersonClass(final String name, final int age) {
     this.name = name;
     this.age = age;
    }
    public int getAge() {
      return age;
    }
    public String getName() {
     return name;
    }
  }
```