# Immutable Objects:

“An object is considered *immutable* if its state cannot change after it is constructed” (Java Tutorials Docs). A number of advantages comes from using immutable objects.

## Advantages:

* **Thread Safety Because no methods can modify the state of immutable objects, they are thread safe no matter how many threads use them and do not require synchronization.**
* **Avoids Temporal Coupling and “Side Effects” Because state cannot be modified, we do not have to worry about the order our methods are called, as they will by definition not have any effect on our object. Also we do not have to worry about external methods modifying or tainting our objects that we pass to them, and can re use them as many times as needed.**
* **Reduces Needless Copying We can expose any immutable object as we don’t have to worry that they will be changed by the caller. This means we don’t have to defensively copy them when returning them causing unneeded object proliferation.**

## Disadvantages:

* **Protecting Mutable Members** Extra overhead is required when returning internal mutable properties that our immutable structure may contain, they must be defensively copied before being returned.
* **Changes Require New Objects** In order to make a “change” a completely new object is created, often a copy is created with the change in place. This causes extra overhead and object proliferation, however it can be mitigated somewhat with methods such as *path copying*. The impact of this over head is often over estimated, as Oracle says: “Programmers are often reluctant to employ immutable objects, because they worry about the cost of creating a new object as opposed to updating an object in place. The impact of object creation is often overestimated, and can be offset by some of the efficiencies associated with immutable objects. These include decreased overhead due to garbage collection, and the elimination of code needed to protect mutable objects from corruption.”