Clone and Equals

In Java, all variable names are references (pointers) to a space in memory where the value in the object is stored. If we compare two objects with == we see if the two references point to the same physical address in memory (a.k.a. the heap in RAM, which is a big array). The only time new space is declared in Java is when the new keyword is used. Therefore, if we want to modify an object and then have access to the object in the unmodified form, we have to clone the object. This involves making a new object and setting all fields inside to the same values (if they are objects, to clones of the objects inside this object). In order to check objects stored in different places in memory for equality, we have to write a method called equals(). Both of these properties are derived from java.lang.Object, which every object extends by default.

```java
public class MyObject{
    public int myNumber;
    public OtherObject obj;
    public String str;
    public MyObject(int myNumber, OtherObject obj, String str){...}
    public boolean equals(Object other){
        if(!(other instanceof MyObject)) return false;
        MyObject casted=(MyObject) other;
        //if your object is huge, you can split this up, ie:
        //if(!obj.equals(casted.obj)) return false;
        //and return true at the end.
        return obj.equals(casted.obj) && myNumber==casted.myNumber &&
            casted.str.equals(str);
    }
    public MyObject clone(){
        return new MyObject(myNumber, obj.clone(), str);
        //we don't need to clone str, because strings are immutable.
    }
}
```

Protip: You can use the IDE to create Getters, Setters, and Constructors for you!

Why do we need to document?

It is very rare to work on code that nobody else will have to work with. Very few modern applications are built and maintained by one person, and even if they are, their codebases are usually so vast that it would be impossible to effectively remember exactly what each part does without good documentation.

What is documentation?

- Good variable naming - variable names should be descriptive, it should be possible to understand good code without too much additional commenting for individual lines
- Comment confusing lines whose meaning is not immediately apparent
- Document the API Meticulously

What is an API?
API is short for Application Programmer Interface. It details to someone using your code what resources they have available. Java was designed to be very conducive to interfacing with someone else’s code. Therefore, it has a very precise method of API commenting called Javadoc. If you Javadoc your code properly, there is a way to generate a pretty webpage like you find in the built in Java API’s documentation (https://docs.oracle.com/javase/7/docs/api/).

Here is a reference to how to generate the javadoc in Eclipse or Netbeans: http://www.itcsolutions.eu/2010/12/23/tutorial-java-6-2-1-how-to-write-or-generate-javadoc-comments/.

You should javadoc your classes (with a brief description of what it is and what it does for each class). You should also javadoc all publicly available data fields with a brief description. Additionally, all methods should have a javadoc. In most IDEs, if you type /** on the line before a method declaration and hit enter, the javadoc comment block should pop up. Here is a sample javadoc comment block with a description of each field

/**
 *This is a one line description of the method.
 *
 *This is a more detailed description of the method, it can span multiple *lines.
 *
 *@param o1 here we describe what the first parameter should be/contain
 *@param q, here we describe what the second parameter should represent
 *@throws AnException here we describe why AnException might be thrown
 *@return here we describe what the method returns (ie: sum of params)
 */

public SomeObject myMethod(AnotherObject o1, int q) throws AnException {
  //some code happens here
}

<table>
<thead>
<tr>
<th>JavaDoc Tag</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@see</td>
<td>Name of associated class</td>
<td>Class, method</td>
</tr>
<tr>
<td>@author</td>
<td>Author</td>
<td>Class</td>
</tr>
<tr>
<td>@version</td>
<td>Version</td>
<td>Class</td>
</tr>
<tr>
<td>@param</td>
<td>Input parameters</td>
<td>Method</td>
</tr>
<tr>
<td>@return</td>
<td>Return value</td>
<td>Method</td>
</tr>
<tr>
<td>@exception</td>
<td>Generated exception</td>
<td>Method</td>
</tr>
<tr>
<td>@throws</td>
<td>Generated exception</td>
<td>Method</td>
</tr>
<tr>
<td>@deprecated</td>
<td>Defines the element as deprecated</td>
<td>Class, method</td>
</tr>
<tr>
<td>@since</td>
<td>The API version in which this element was included</td>
<td>Class, method</td>
</tr>
</tbody>
</table>