**Equals**

```java
class MyClass {
    public boolean equals(Object o) {
        return this == o;
    }
}
```

**Guidelines:**
- `x.equals(x)` should return true
- `x.equals(y)` should return true if and only if `y.equals(x)` returns true
- if `x.equals(y)` and `y.equals(z)` return true, then `x.equals(z)` should return true
- multiple invocations of `x.equals(y)` should consistently return the same answer if no state used in the equals method changes
- `x.equals(null)` should return false
- generally necessary to override `hashCode()` whenever `equals()` is overridden

**Special notes:**
- must take an Object as the parameter
- should be legal to compare this object to *any* other object, including objects of different type (return false in that case)
- use `getClass()` to compare the type of this object and the parameter object

**Equals() in the wild:**
- `contains()` method of `Collection` uses `equals()` to determine equality
- two different implementations of `Set` can be equal if they have the same contents

*Effective Java Tip #8: Obey the general contract when overriding equals.*

**Implementing Comparable<T>**

```java
public interface Comparable<T> {
    public int compareTo(T o);
}
```

**Semantics of** `a.compareTo(b)`:

<table>
<thead>
<tr>
<th>Returned int</th>
<th>Relationship between <code>a</code> and <code>b</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td><code>a &lt; b</code>; `a&quot; comes before&quot; b in the natural ordering</td>
</tr>
<tr>
<td>0</td>
<td><code>a = b</code></td>
</tr>
<tr>
<td>+</td>
<td><code>a &gt; b</code>; `a&quot; comes after&quot; b in the natural ordering</td>
</tr>
</tbody>
</table>

**Guidelines:**
- used to describe a "natural ordering" of a class of objects
- `x.compareTo(null)` should throw a `NullPointerException`
- recommended that `compareTo()` be consistent with `equals()`

**Implementation hints:**
- use the subtraction trick (return `this.int - other.int`)
- call the `compareTo()` method of fields that are objects (return `this.string.compareTo(other.string)`)
• for doubles, use either Math.signum() (return (int)Math.signum(this.double - other.double)), or if/else chains

CompareTo in the wild:
• Every collection or method in the Java library that uses the "natural ordering" of elements calls compareTo(), including:
  o TreeMap
  o TreeSet
  o PriorityQueue
  o Collections.sort()

*Effective Java Tip #12: Consider implementing Comparable.*

**Implementing Comparator<T>**

```java
public interface Comparator<T> {
    public int compare(T o1, T o2);
    public boolean equals(Object o);
}
```

Semantics of compare(a, b):
Same as a.compareTo(b)

Guidelines:
• used to describe an "artificial ordering" of a class of objects, even if there is no "natural ordering"
• can be passed to Java library objects and methods that use sorting instead of compareTo()

**Clone**

```java
protected Object clone() throws CloneNotSupportedException { ... }
```

General intent:
• that all of the following are true:
  o x.clone() != x
  o x.clone().getClass() == x.getClass()
  o x.clone().equals(x)

Special notes:
• when overriding, change to a public method and change return type
• must implement Cloneable interface
• use super.clone() as the initial copying operation (performs a shallow copy), then add in modifications of fields, deep copying, etc

Shallow copy:
• copies the values of all primitive fields and the references to all object fields
• objects used by this object are now shared by the original and the clone
• this is what Object.clone() does

Deep copy:
• copies the values of all primitive fields and clones all object fields
• objects used by the clone are separate copies

*Effective Java Tip #11: Override clone judiciously.*