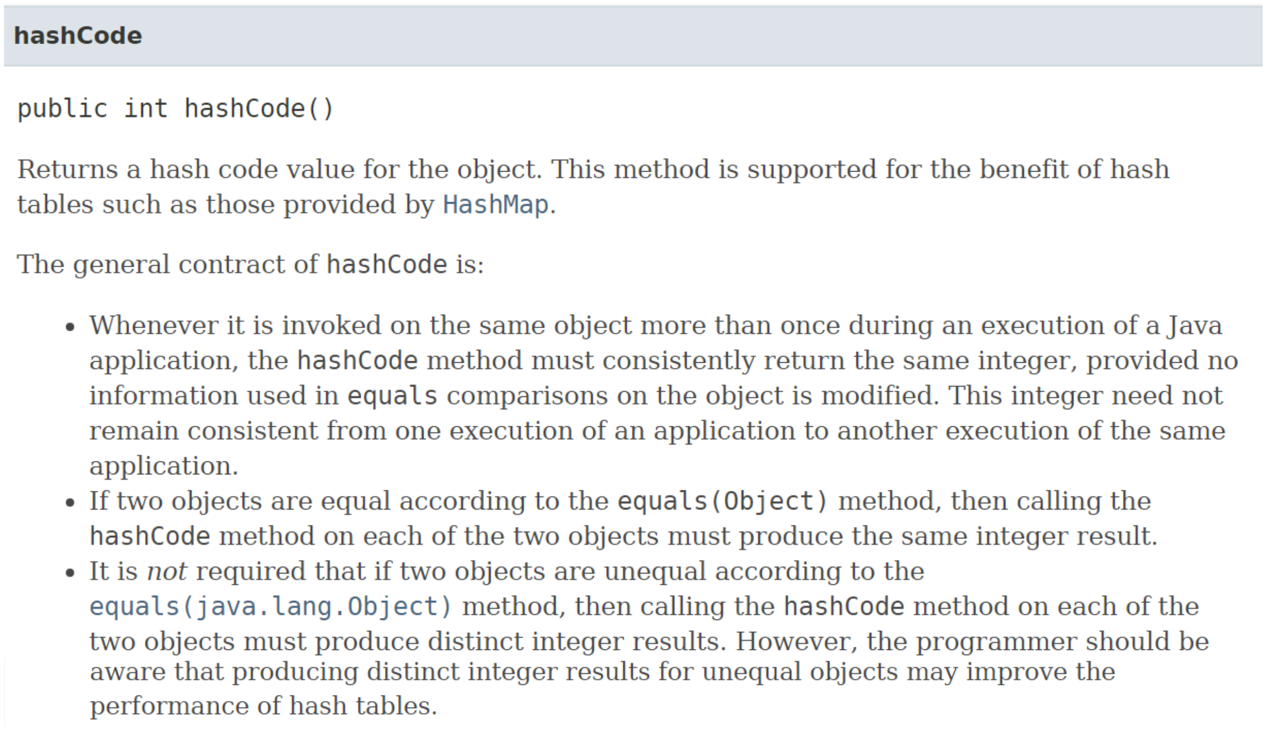
equals() and hashCode() methods

Let’s consider Point class below. Let’s say we have decided to override to equals() method that allows us to check whether two objects are equivalent. However, whenever we override equals() method, we need to override hashCode() method as well. If we do not do that, we will not be able to use any of the Collection structures that use hashing functions such as HashSet in order to store instances of these objects.

```
public class Point {

	private int x;
	private int y;


	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}

	@Override
	public boolean equals(Object obj) {
	    if (!(obj instanceof Point))
	      return false;
	    Point p = (Point) obj;
	    return (p.x == this.x) && (p.y == this.y);
	}


}
```

According to JavaDoc, hashCode() method has the following contract:

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#hashCode--



hashCode

`public int hashCode()`

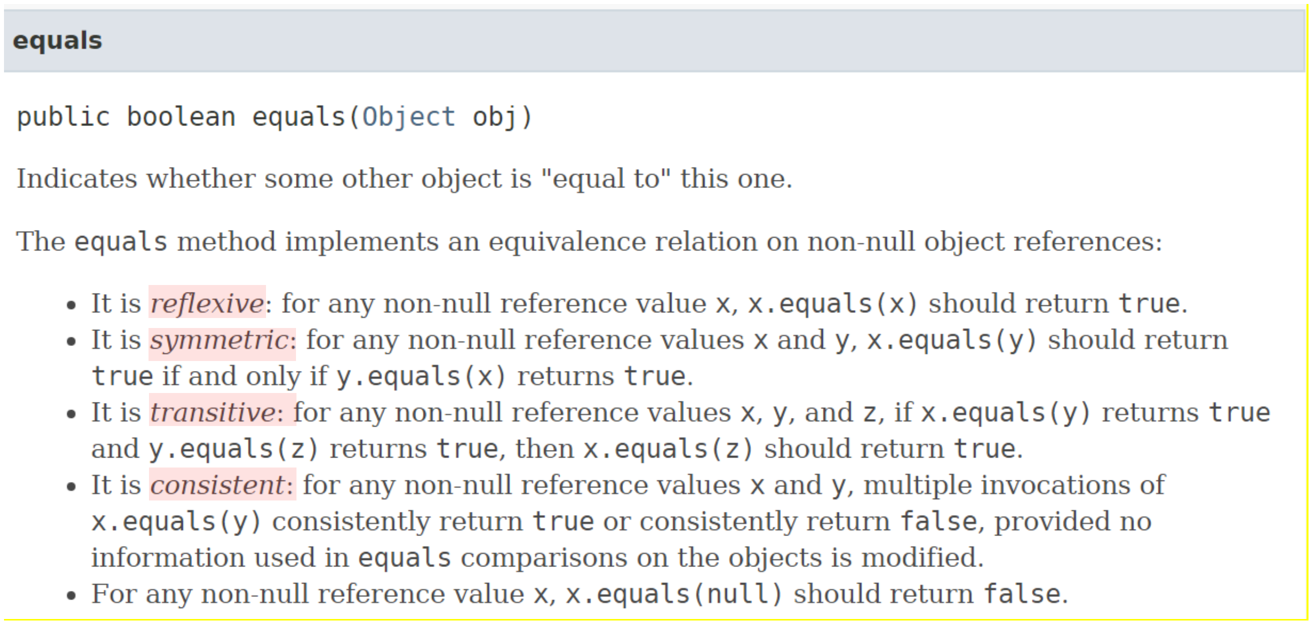
Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `HashMap`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the `equals(java.lang.Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

According to JavaDoc, equals() method has the following contract:

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-



equals

`public boolean equals(Object obj)`

Indicates whether some other object is "equal to" this one.

The `equals` method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any non-null reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

Summary:

If two objects are equal according to the equals() method, they must have the same hash code. However, that does not go the other way around. Specifically, if two objects have the same hash code, they do not have to be equal (they can share the same hash code).