OBJECT METHODS

Oracle - Java Tutorial: Object as a Superclass
Object API

@ Dr. Jessica Young Schmidt and NCSU Computer Science Faculty

CLASS OBJECT

public class Object

- Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects implement the methods of this class.
  - There are special methods that we want to include in our classes that create instances of an object
    - toString
    - equals
  - Methods come from Object class and must match the syntax EXACTLY!!!

toString

A special method that returns a String representation of an object

- Called automatically by Java when concatenating an object with a String or when an object is printed
- Writing your own toString method overrides the default method, which prints the class name followed by @ and some letters and numbers (for example, Day@397d812b)

```
public String toString() {
    return <code to produce string>;
}
```

toString

```
public String toString()
```

Returns a string representation of the object. In general, the `toString` method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

The `toString` method for class `Object` returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object. In other words, this method returns a string equal to the value of:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

Returns:

a string representation of the object.

Book.java:

```
public String toString() {
    return this.title + " by " + this.author;
}
```

equals

Method that compares two objects for equality of some or all state

- Default method compares to see if objects are the same object (location or identity: same as ==)
- Implemented within the class definition of the object under comparison
  - Compares the implicit parameter (current object) with the object passed as a parameter

```
public boolean equals(Object o) {
    <statement>;
    ...
    <statement>;
    return <boolean>;
}
```

Key: The equals method takes an Object parameter. The type of the parameter will always be an Object no matter which class the method is within.

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`.

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x == y` has the value `true`).

Note that it is generally necessary to override the `hashCode` method whenever this method is overridden, so as to maintain the general contract for the `hashCode` method, which states that equal objects must have equal hash codes.

Parameters:

`obj` - the reference object with which to compare.

Returns:

`true` if this object is the same as the obj argument; `false` otherwise.

See Also:

`hashCode()`, `HashMap`

OBJECT CASTING

- To compare the Object parameter to our implicit parameter, we need the Object to be an object of our class type
- Object Casting: promise to the compiler that the Object reference actually refers to a different type and the compiler can treat it as such

```
Book b = (Book) o;
```

instanceof Keyword

<expression > instanceof <type >

- Passing non-Book objects will cause ClassCastException
- Instead, want equals method to return false if Object parameter is not a Book
- Use instanceof operator to test if the Object parameter is a Book
- Takes care of a null check

EQUAL BOOKS

What criteria do we want to use to determine if two books are equal?

- Same instance?
- Same author, title, and publication year?

```
public boolean equals(Object o) {
    if (o instanceof Book) {
        Book b = (Book) o;
        return title.equals(b.getTitle())
                && author.equals(b.getAuthor())
                && pubYear == b.getPubYear();
    } else {
        return false;
    }
}
```