Object Methods

zyBook Chap 10.4

`Object` Class (`public class Object`)

`Object` Class 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 override in our own classes that create instances of an object
  - `toString()`
  - `equals()`
  - Methods come from the `Object` class and must match the syntax EXACTLY

toString() in Object Class

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.

toString()

Overridden version

```
$ javac ToStringDemo.java
$ java ToStringDemo
java.awt.Point[x=1,y=2]
java.awt.Point[x=1,y=2]
[I@6d06d69c
[I@6d06d69c
```

Original version

```
import java.awt.*;

public class ToStringDemo {
    public static void main (String[] args) {
        Point p = new Point(1, 2);
        System.out.println(p.toString());
        System.out.println(p);

        int[] arr = new int[3];
        System.out.println(arr.toString());
        System.out.println(arr);
    }
}
```

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.

toString() can be **called automatically** by Java when concatenating an object with a String or when an object is printed

toString()

toString

```
public String toString()
```

The **header stays the same** when **overridden**

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.

Override toString() in Book Class

```
public class BookClient {
    public static void main(String[] args) {
        Book book = new Book("Building Java Programs, 5th Edition",
                             "Stuart Reges, Marty Stepp", 2020, 7);
        System.out.println(book.toString());
    }
}
```

```
$ javac BookClient.java
$ java BookClient
Building Java Programs, 5th Edition by Stuart Reges, Marty Stepp in 2020.
```

```
public class Book {
    // ... Instance Variables ...

    // ... Instance Methods ...
    public Book(String title, String author, int pubYear, int location) {
        this.title = title;
        this.author = author;
        this.pubYear = pubYear;
        this.location = location;
        this.checkedOutBy = null;
        this.dueDate = null;
    }

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

equals() in Object Class

equals

```
public boolean equals(Object obj)
```

The **header stays the same** when **overridden**

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`

The **default** equals() method compares the **referential equality**

Override equals()

Compares two objects for **equality of SOME or ALL fields**

- Implemented within the class definition of the object under comparison
  - Compares the implicit parameter (current object) with the object passed as a parameter
  - That is, we need **the object to be compared** to be the **same object type** as the current object via
    - `instanceof` operator
    - Object casting

Object Casting

- Widening Typecasting with Objects (Upcasting)
  - Cast a reference along the class hierarchy in a direction from the subclasses towards the root class
    - E.g., cast a Dog object into an Animal object
- Narrowing Typecasting with Objects (Downcasting)
  - Cast a reference along the class hierarchy in a direction from the root class towards the subclasses
    - E.g., cast an Animal object into a Dog object

Downcasting + `instanceof` Keyword

We often use **`instanceof`** operator before downcasting to check if the object belongs to the specific type:

```
<obj> instanceof <objType>
```

- also checks if the object is **`null`**
- If the real object doesn't match the type we downcast to, then **ClassCastException** will be thrown at runtime

Override equals() in Book Class

```
public class Book {

    // ... Some instance variables and methods here ...

    public boolean equals(Object obj) {
        // The parameter obj could be any type of Object, such as a Point
        // Hence, we need to check if obj is an instance of Book Class
        if (obj instanceof Book) {
            // Object casting
            // Tell the compiler to treat the obj as if it is a Book object
            Book b = (Book) obj;
            // title and author are Strings, and hence compared with equals()
            // pubYear is an int, and hence compared with ==
            return title.equals(b.getTitle()) && author.equals(b.getAuthor())
                    && pubYear == b.getPubYear();
        } else {
            return false;
        }
    }
}
```

Override equals() in Book Class

```
import java.awt.*;

public class BookClient {
    public static void main(String[] args) {
        Book b1 = new Book("Building Java Programs, 5th Edition", "Stuart Reges, Marty Stepp", 2020, 7);
        Book b2 = new Book("Building Java Programs, 5th Edition", "Stuart Reges, Marty Stepp", 2020, 6);
        Book b3 = new Book("Building Java Programs, 4th Edition", "Stuart Reges, Marty Stepp", 2016, 7);
        Book b4 = null;
        Point p = new Point(1, 2);

        System.out.println("b1.equals(b2) is " + b1.equals(b2));
        System.out.println("b1.equals(b3) is " + b1.equals(b3));
        System.out.println("b1.equals(b4) is " + b1.equals(b4));
        System.out.println("b1.equals(p) is " + b1.equals(p));
    }
}
```

```
$ javac BookClient.java
$ java BookClient
b1.equals(b2) is true
b1.equals(b3) is false
b1.equals(b4) is false
b1.equals(p) is false
```