More on Inheritance
Ch 5.3 & 5.5
Highlights

- Object class

```java
Object o = new Object();
```

- Built in functions

```java
o.toString();
o.equals(o);
```

- Up-casting/down-casting

```java
String x = "hi";
Object xo = (Object)x; // Up-casting
String same = (String)xo; // Down-casting
```
Early/late binding

**Early binding** chooses methods at **compile time** (runs based on object type in code)
Late binding chooses methods at run time (can pick most appropriate method)

What determines early/late binding?
- final and static modifiers use early binding (and private methods)
- Everything else uses late binding

(See: Binding.java and BindingParent.java)
Inheritance and static methods

Classes cannot “override” their parent's static methods

static means this is a property of the class, (the blue print stage) and thus not inherited

However you can “replace” them
(See: StaticMethodChild.java and StaticMethodParent.java)
Upcasting/downcasting

Upcasting is when a child is type-cast to a parent class:

```
Child c = new Child();
Parent p = (Parent)c;
```

Downcasting is when a parent is type-cast to a child class:

```
Parent p = new Parent();
Child c = (Child)p;
```

This will crash
Upcasting/downcasting

When can you upcast (without error)?
- Anytime, no problem!

When can you downcast (without error)?
- You can only correctly downcast objects if they were upcast at some point
  - Use `instanceof` to check if downcast possible

(See: UpDownCasting.java)
When you make a class, there seem to be methods defined already...

(See: Blank.java)

This is because every class is actually a descendant of the Object class

(See: Object.java)
Object Class

Why is having Object as a parent useful?
- Allows a generalizable “class”
- Defines useful methods that should exist (for example: toString() with println() )

This means we can use an array of Objects to store any type of classes
(See: ObjectArray.java)
A very useful method that all classes get from object is the `toString()` method. This method returns a String, and it controls what is shown when you `println()` an object. However, you have to override it to change its default behavior:

```
Child c = new Child();
System.out.println(c);
```

(See: OverrideToString.java)