CS 61BL Lab 4

Ryan Purpura

Method Overloading

- Two methods in a class can have the same name, as long as they have a different **signature**
- Different signature = different name or different # of parameter or different parameter types (order matters)
- If two methods have the same name but still has a different signature, this is called **method overloading**
- Return type does not contribute to the signature, and neither does the *name* of the parameters

Examples

Method 1	Method 2	Can coexist?
`int add(int x, int y)`	`int add(double x, double y)`	Yes
`int add(int x, int y)`	`int add(int x, int y, int z)`	Yes
`int add(int x, int y)`	`long add(int first, int second)`	No
`int add(int x, int y)`	`int add2(int x, int y)`	Yes

Inheritance

- If you have an existing class, you can add functionality to it by **extending** it.
- The new class will inherit all of the methods and instance variables of the class, but...
  - It cannot directly access private instance variables and private methods.
- The new class can also **override** methods, allowing the subclass to have a different implementation of the "same" method.

```java
public class Salmon extends Fish {
    private String home;
    public Salmon(int w, String h) {
        super(w);
        home = h;
    }

    public void migrate() {
       System.out.println(
           "Migrating to " + home;
    }

    @Override
    public void swim() {
        System.out.println("splish splash");
    }

    public void swim(int speed) {
         System.out.println("swimming at " + speed + " mph");
    }
}
```

subclass (the one that adds functionality)

```java
public class Fish {
    private int weight;
    public Fish(int w) {
        weight = w;
    }

    public void swim() {
        System.out.println("splash");
    }
}
```

superclass (the original class)

The constructor

- Constructors are NOT inherited!
- Additionally, you *must* have a call to one of the superclass's constructor as the first line of the new constructor.
    - Exception: if there is a no-arg constructor in the superclass, Java will insert it in for you on compilation if you don't make a call to **`super(<args>)`**
- To call a superclass' constructor, use **`super(<args>)`**

```
public Salmon(int w, String h) {
    super(w);
    home = h;
}
```

Polymorphism

```
Fish alice = new Fish(20);
Salmon bob = new Salmon(15, "Yukon River");
alice.swim();
bob.swim();
bob.migrate();

alice = bob;
alice.swim();

// alice.migrate();
// above doesn't compile... why?
```

```
splash
splish splash
Migrating to Yukon River
splish splash
```

Static vs. Dynamic Types

- Every variable has a static and dynamic type.
- **`Fish x = new Salmon(5, "Yukon River");`**
- The static type is the type that you declared it to be. (in this case, `Fish`)
- The dynamic type is which class actually gets called in the constructor (aka the class it actually is, in this case, `Salmon`)
- The dynamic type must be a subclass (or the same as) the static class. We describe this as an **is-a** relationship (E.g. Salmon **is a** Fish, but not all Fish are Salmon)

The compiler

- The compiler only knows about the static type!
- Because of this, you can only call methods that exist in the static type's class.
- But if a variable's dynamic type overrides a method, you will call the overridden version! We saw that earlier.
- To temporarily "upgrade" a static type, use casting, for example,

```
Fish x = new Salmon(5, "Yukon River");
((Salmon) x).migrate()
```

- See lab for more details on casting.