6.178: Introduction to Software Engineering in Java

Lecture 5: Inheritance and Polymorphism

Quick review

classes with fields, constructors, and instance methods are Java objects

the object contract with equals(...) and hashCode()

primitives vs. objects

mutable data can be sneaky

Static variables/methods

one copy shared between ALL instances

called with ClassName.variable or ClassName.method()

```
public class MathThings {
    public static final double PI = 3.14;

    public static double circleArea(int radius) {
        return PI * radius * radius;
    }
}

MathThings.PI;                MathThings.circleArea(3);
```

Redundancy

take a look at R2D2.java and R3D2.java in the redundancy package

spot the differences!

add a method, flashLights(), to both:

```
public void flashLights() {
    System.out.println(lightColor);
}
```

update flyStarfighter() and saveGalaxy() to call flashLights() after calling playSound()

fine with 2 - what about 3, 4, 10, 20?

A solution

close all open Java programs, don’t quit Eclipse

look at R2D2.java in the **inheritance** package

now look at R3D2.java in the same package

run Main.java

what is going on?

Subclassing

```
public class R3D2 extends R2D2
```

R3D2 is a *subclass* or *child class* of R2D2, which is the *superclass* or *parent class*

R3D2 *inherits* properties from R2D2

public/protected fields/methods (& package-private if in same package)

ex: `r3.sound; r3.flyStarfighter();`

You can only directly extend one class

What can you do?

write a constructor that invokes the superclass constructor

use inherited fields/methods

declare new fields/methods

define new field with same name to hide the inherited one

override a method

super

```
public R3D2() {
    super();
    lightColor = "purple";
    galaxySaved = false;
}
```

can be used in two ways

- invoke the superclass constructor (shown above)
- access a hidden field/method

Calling the superclass constructor

can use super() or super(parameter list)

- constructor with appropriate parameters will be called

if not explicitly called, compiler automatically inserts a call to super()

- will give an error if there isn’t a constructor with no arguments

Accessing hidden fields/methods

super.field or super.method()

in R3D2, override the flyStarfighter() method:

```
@Override
public void flyStarfighter() {
    playSound();
    super.flyStarfighter();
}
```

now add the line `r3.flyStarfigher();` in Main.java and run it

Object.java

all classes that don’t explicitly extend a class extend Object

where we get methods like toString(), equals(...), hashCode(), etc.

so all classes are descendents of Object

Polymorphism

objects can have multiple types at the same time

close all open java files and open Main.java in the polymorphism package

run the program - what’s the output?

Polymorphism

```
Object test = new String("test");
```

the test variable is both an Object and a String

try to call a string function on test

ex: `test.length();`

what happens?

virtual method invocation

Cookies!

take a look at the three cookie classes in the polymorphism package, starting with Cookie.java

uncomment the section labelled “Cookie” in Main.java, and run it

do the results make sense?

Typecasting

saw it in equals(...) method template

```
Student thatStudent = (Student) thatObject;
```

upcast - casting to a supertype

```
ChocChipCookie chocChip = new ChocChipCookie(3);
Cookie upCasted = (Cookie) chocChip;
```

downcast - casting to a subtype

```
Cookie chocChip = new ChocChipCookie(3);
ChocChipCookie downCasted = (ChocChipCookie) chocChip;
```

Why typecast?

downcasting - get more specific methods/fields

ex: in an equals(...) method

```
Cookie thatCookie = (Cookie) thatObject;
return this.diameter == thatCookie.diameter;
```

upcasting - storing a more specific item in a general way

ex:
```
List<Cookie> cookies = new ArrayList<>();
cookies.add((Cookie) chocChipCookie);
```

instanceof

also saw this in equals(...) template

`variable instanceof Class` is true if:

- variable is exactly type Class (`student instanceOf Student`)
- variable is a subclass of Class (`dog instanceOf Mammal`)
- variable a implements interface Class (next class!)

add the following to Main.java and run it:

```
System.out.println(chocChip instanceof Cookie);
System.out.println(chocChip instanceof ChocChipCookie);
System.out.println(chocChip instanceof Snickerdoodle);
```

Old MacDonald

print 3 different verses of Old MacDonald in OldMacdonald.java

need a List of Animals

loop over the list and print out the song each time

you should only ever call Animal functions on each animal instance, but they should all give specific results

ex: `Animal cow = new Cow();`
`cow.name()` -> “cow”; `cow.sound();` -> “moo”

something that may come in handy: “\n” is a new line in a String