

CS231 Fall 2013 Lecture 5

Stephanie R Taylor

September 13, 2013

1 Administrative Topics

- We take the quiz.

2 Inheritance and polymorphism

- One big advantage of OO programming over iterative programming is that it is conceptually easier to organize and understand large programs if they are divided into classes and objects
- But there is another advantage that we haven't seen yet: It promotes code reuse and avoids code duplication.
- Let's go back to our Die class and in the main method, add a line.

```
System.out.println(die1.toString());
```

- What do you think will happen? error? [run it]
- Somehow the Die class has a toString() method. Where did it come from?
- Answer: It inherited it from its parent or superclass. Every class has a parent class except for the Object class which is the granddaddy of them all. If you don't specify a parent class, then Object is used as the parent. [draw inheritance tree]

- A subclass inherits all the methods and fields from its parent class. Since that class inherited methods and fields from its parent, a grand-child inherits from both the parent and the grand parent.
- It turns out that the Object class has an toString method and that's where it came from in our example. [look at online docs for Object class]
- Overriding toString
 - What if we create our own toString method in the Die class and then call toString as we did above? Which toString method gets executed?
 - Answer: The most local one does. So the Die class' toString method gets executed.
 - This is called overriding. The Die toString method overrides the inherited toString method.
 - Why would you want to override the inherited toString method?
 - Answer: The inherited method just returns the string "Die@...", which is not very helpful. A more friendly version of toString() would be nice. [add it to the Die class in the quiz solution]
- Summary:
 - A class inherits all the methods of its ancestor classes, all the way up the Object class.
 - A subclass can override the inherited methods by defining a method with the same header in the subclass.
 - Fields are also inherited by subclasses. However, if the fields are private in the superclass, then the subclass inherits them but can't access them.
- Let's try another line of code to see what happens:


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

 - What do you think will happen when we try to execute it? Error? [run it]

- Since Die is a subclass of Object, every Die “is an” Object and so you can do such assignments. A variable of type A can be assigned any object of type A or of a type that is a subclass of A.
- How do you tell Java that you want your class to be a subclass of another class?
 - Use the **extends** keyword:

```
public class Die extends Object
{ ... }
```

- But you don’t have to explicitly extend any class and in that case the Java compiler automatically puts in **extends** Object for you.

3 Static fields and methods

- Static methods belong to a class, not to an object. That is, they are things you ask classes to do, not objects when main started, there was a class Die but there was not Die object created yet. We asked the class Die to execute the main method.
- Where else have we seen static methods? [Math.random]
- Can have static fields as well that belong to the class. All objects of that class have access to the static variable they can all modify the same variable; they don’t have their own copies of it like instance fields. Where have we seen static fields? [System.out]

4 Equals

- Note that Die also inherits an equals method. What is that all about?
- Consider the code:

```
int x = 3, y = 3;
Die d1 = new Die();
Die d2 = new Die();
d1.roll(); d2.roll();
```

```
if(x == y) x = 0;  
if(d1 == d2) y = 0;
```

- What happens if we run this?
- What if you want to test whether the data held by two objects are equal rather than whether the two objects are really the same object? Create a method that does so!