Inheritance Review

1. Inheritance allows us to reuse existing code.
2. Allows us to define a hierarchy of classes.
3. *Base class* has the most general behavior
4. *Derived classes* have more specific behavior.
public class Person
{
    public String lastName;
    public String id;
    public void eat() {}
    public void sleep() {}
}
public class Professor extends Person
{
    public String officeNum;
    public void teach() { }
    public void grade() { }
    public void forget() { }
}
What if we wanted to make a Student class that was a subclass of Person?
   ▷ methods attendClass, doHomework()

What if we also wanted to make a GradStudent that was a subclass of Student?
   ▷ New instance variable: public Professor advisor.

Note that Student is an *is-a* relationship, and advisor is a *has-a* relationship.
The Java I/O package provides some nice examples of inheritance.

**InputStream** is a base class
- Provides read(), skip(), close() methods.
- Very basic functionality.

**FileInputStream** and **FilterInputStream** subclass **InputStream**
- They *override* the behavior of InputStream for particular data sources.
On Monday, you did a lab in which you created a Shape class.
   ▲ This had an area() method.

You then subclassed it with Circle and Rectangle classes.

One problem with this: you may not want users to ever create Shapes.
   ▲ You just want anything that inherits from Shape to have an area() method.
6 Solution: define Shape as an *abstract class*

```java
public abstract class Shape {
    public int locX;
    public int locY;
    public abstract double area(); // note: semi-colon, no method body
}
```
An abstract class is one that has one or more abstract methods.

- Can also have concrete methods.

Classes that subclass from an abstract class must override all abstract methods.

An abstract class therefore provides a common *interface* for a set of subclasses.
How would we rewrite Shape as an abstract class?
What if we have this situation:

```java
public class A {
    public void m1() { }
    public void m2() { }
}
public class B extends A {
    public void m2() { }
}
B bex = new B();
A aex = new B();
bex.m1();
aex.m1();
bex.m2()
```

Which methods are called?
**15-10: Dynamic binding**

- Java resolves this via *dynamic binding*
  - The actual type of an object is determined at runtime.
  - That object's class is searched for the corresponding method.
  - If the method doesn’t exist in that class, the parent class is checked.
- What is the advantage of dynamic binding?
- What is the disadvantage?
15-11: Next time

- Dealing with constructors
- Interfaces vs subclasses
- Polymorphism