Announcements

- A1 Due Thursday
- A2 Out Today

Where am I? Big ideas so far.

- Java variables have types (L1)
  - A type is a set of values and operations on them
    (int: +, -, *, /, %, etc.)
- Classes define new types (L2)
  - Methods are the operations on objects of that class.
- Fields allow objects to contain data (L3)

Class House

```java
public class House {
    private int bdrs; // number of bedrooms, >= 0.
    private int baths; // number of bathrooms, in 1..5

    /** Constructor: number of bedrooms b1, number of bathrooms b2
    *     Prec: b1 >= 0, 0 < b2 <= 5 */
    public House(int b1, int b2);

    /** Return number of bedrooms */
    public int getBeds() {
        return bdrs;
    }

    /** Return number of bathrooms */
    public int getBaths() {
        return baths;
    }

    Contains other methods!
}
```

Class Object: the superest class of all

```java
public class House extends Object {
    private int bdrs; // number of bedrooms, >= 0
    private int baths; // number of bathrooms, in 1..5

    /** Constructor: number of bedrooms b1, number of bathrooms b2
    *     Prec: b1 >= 0, 0 < b2 <= 5 */
    public House(int b1, int b2);

    /** Return number of bedrooms */
    public int getBeds() {
        return bdrs;
    }

    /** Return number of bathrooms */
    public int getBaths() {
        return baths;
    }

    Contains other methods!
}
```

We often omit the Object partition to reduce clutter; we know that it is always there.
Classes can extend other classes

/** An instance is a subclass of JFrame */
public class C extends javax.swing.JFrame {
    C: subclass of JFrame
    JFrame: superclass of C
    C inherits all methods that are in a JFrame
}

C@6667f34e
C subclass of JFrame
JFrame: superclass of C
C inherits all methods that are in a JFrame

JFrame Object has 2 partitions:
    one for JFrame methods,
    one for C methods

Accessing superclass things

- Subclasses are different classes
  - Public fields and methods can be accessed
  - Private fields and methods cannot be accessed
  - Protected fields can be access by subclasses

Keywords: this

- this keyword: this evaluates to the name of the object in which it occurs
- Makes it possible for an object to access its own name (or pointer)
- Example: Referencing a shadowed class field

Apartment extends House
    private int floor;
    private Apartment downstairs;
    //constructor
    public Apartment(int floor, Apartment downstairs) {
        this.floor= floor;
        this.downstairs = downstairs;
    }

Inside-out rule shows that field x is inaccessible!

Overriding methods

Object defines a method toString() that returns the name of the object
Apartment@af8

Java Convention: Define toString() in any class to return a representation of an object, giving info about the values in its fields.
New definitions of toString() override the definition in Object.toString()
When should you make a subclass?

- The inheritance hierarchy should reflect modeling semantics, not implementation shortcuts.
- A should extend B if and only if A “is a” B
  - An elephant is an animal, so Elephant extends Animal
  - A car is a vehicle, so Car extends Vehicle
  - An instance of any class is an object, so AnyClass extends java.lang.Object
- Don’t use extends just to get access to protected fields!

When should you make a subclass?

- Which of the following seem like reasonable designs?
  - Triangle extends Shape { … }
  - PHDTester extends PHD { … }
  - BankAccount extends CheckingAccount { … }

Static Methods

- Most methods are instance methods: every instance of the class has a copy of the method.
- There is only one copy of a static method. There is not a copy in each object.

An Example

```java
/** = “this object is below”. */
public boolean isBelow(Apartment a){
    return this == a.downstairs;
}

/** = “a is below b”. */
public static boolean isBelow(Apartment b, Apartment a){
    return b == a.downstairs;
}
```

Referencing a static method

- static: there is only one copy of the method. It is not in each object

Good example of static methods

- java.lang.Math
  - http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
- Or find it by googling Java 8 Math
Static Fields

- There is only one copy of a static method. There is not a copy in each object.
- There is only one copy of a static field. There is not a copy in each object.

What are static fields good for?

Class java.awt.Color uses static variables

An instance of class Color describes a color in the RGB (Red-Green-Blue) color space. The class contains about 20 static variables, each of which is (i.e. contains a pointer to) a non-changeable Color object for a given color:

public static final Color black = …;
public static final Color blue = …;
public static final Color cyan = new Color(0, 255, 255);
public static final Color darkGray = …;
public static final Color gray = …;
public static final Color green = …;
...

Uses of static variables: Maintain info about created objects

```java
public class Apartment extends House {
    public static int numAps; // number of Apartments created
    /** Constructor: */
    public Apartment(...) {
        numAps = numAps + 1;
    }
    // To have numAps contain the number of objects of class Apartment that have been created, simply increment it in constructors.
    numAps stored in the Container for Apartment
    To access: Apartment.numAps
}
```

Uses of static variables: Implement the singleton pattern

```java
public class WhiteHouse extends House{
    private static final WhiteHouse instance = new WhiteHouse();
    private WhiteHouse() { } // ... constructor
    public static WhiteHouse getInstance() {
        return instance;
    }
    // ... methods
}
```

Only one WhiteHouse can ever exist.