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) and define the contents of each object of the class.
  - Methods: are the operations on objects of that class.
  - Fields: allow objects to contain data (L3)

Class House

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

    /** Constructor: bed is number of bedrooms,
     * bath is number of bathrooms
     * Prec: bed >= 0, 0 < bath <= 5 */
    public House(int bed, int bath) {
        nBed = bed;
        nBath = bath;
    }

    /** Return no. of bedrooms */
    public int getNumBed() {
        return nBed;
    }
}
```

Class Object: the superest class of all

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

    /** Constructor: bed is number of bedrooms,
     * bath is number of bathrooms
     * Prec: bed >= 0, 0 < bath <= 5 */
    public House(int bed, int bath) {
        nBed = bed; nBath = bath;
    }

    /** Return no. of bedrooms */
    public int getNumBed() {
        return nBed;
    }
}
```

Announcements

- A1 Due Friday
- A2 Out Today
public class House extends Object {  
    private int nBed; // number of bedrooms, >= 0.  
    private int nBath; // number of bathrooms, in 1..5  
    public House(int bed, int bath) {  
        nBed = bed;  
        nBath = bath;  
    }  
    public int getNumBed() {  
        return nBed;  
    }  
}

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  
    object has 3 partitions:  
    for Object components,  
    for JFrame components,  
    for C components  
    C@6667134e  
    equals() toString() …  
    Object  
    hide() show() setTitle(String) getTitle()  
    getWidth() getHeight() …  
    getXX() getYY() setLocation(int, int)  
    C

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

public class House {  
    private int nBed; // number of bedrooms, >= 0.  
    private int nBath; // number of bathrooms, in 1..5  
    /** Constructor: */  
    public House(int nBed, int nBath) {  
        nBed = nBed;  
        nBath = nBath;  
    }  
    /** This evaluates to the name of the object in which it occurs */  
    this.nBed = nBed;  
    this.nBath = nBath;  
    }  
    /** Example: Referencing a shadowed class field */  
    this.nBed = nBed;  
    this.nBath = nBath;  
    }  
    this avoids overshadowed field names
### A Subclass Example

```java
public class House {
    private int nBed; // num bedrooms, >= 0
    private int nBath; // num bathrooms, 1..5

    public House(int bed, int bath) {
        nBed = bed;
        nBath = bath;
    }

    public int getNumBed() {
        return nBed;
    }
}
```

### Overriding methods

```java
public class Apt extends House {
    private int floor;
    private Apt downstairsApt;

    public Apt(int floor, Apt downstairs) {
        this.floor = floor;
        downstairsApt = downstairs;
    }

    public String toString() {
        return "+" +
                (getNumBed() + getNumBath()) +
                " room apartment on " +
                floor + "th floor";
    }
}
```

### When should you make a subclass?

Which of the following seem like reasonable designs?

A. Triangle extends Shape {
   ... }
B. PHDTester extends PHD {
   ... }
C. BankAccount extends CheckingAccount {
   ... }

### 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!

- Which of the following seem like reasonable designs?
  A. Triangle extends Shape {
     ... }
  B. PHDTester extends PHD {
     ... }
  C. 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.

Make a method static if the body does not refer to any field or method in the object.

An Example

```java
/** returns true if this object is below Apt a */
public Boolean isBelow(Apt a)
{
    return this == a.downstairsApt;
}

/** returns true if Apt b is below Apt a */
public static boolean isBelow(Apt b, Apt a){
    return b == a.downstairsApt;
}
```

Referencing a static method

- `Apt@af
  House
  numBed
  2
  floor
  4
dstrs
  Apt@af
  nBath
  1
  Apt
  Apt@af
  nBath
  1
  Apt
  Apt@af
  Apt
  Apt@b4
  House
  numBed
  2
  floor
  4
dstrs
  Apt@af
  nBath
  1
  Apt
  Apt@af
  Apt
  Apt@b4

Container for Apartment contains: objects, static components

```java
{ 
    a = new Apt(...);
    b = new Apt(...);
    if (a.isBelow(b)) ... 
    if (Apt.isBelow(a, b)) ...
}
```

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?

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

Use of static variables:
Maintain info about created objects

```java
public class Apt extends House {
    public static int numApt; // number of Apartments created

    /** Constructor */
    public Apt(...) {
        ...
        numApt++; // increment numApt in constructors.
    }
}
```

To have numApt contain the number of objects of class Apartment that have been created, simply increment it in constructors.

numApt stored in the Container for Apartment
To access: Apartment.numApt
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:

```java
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 = ...;
...
```

Class `java.awt.Color` uses static variables:

```java
public class WhiteHouse extends House {
    private static final WhiteHouse instance = new WhiteHouse();

    private WhiteHouse() {} // ... constructor

    public static WhiteHouse getInstance() {
        return instance;
    }
    // ... methods
}
```

Uses of static variables:
Implement the singleton pattern

Only one WhiteHouse can ever exist.

```java
public class WhiteHouse extends House {
    private static final WhiteHouse instance = new WhiteHouse();

    private WhiteHouse() {} // ... constructor

    public static WhiteHouse getInstance() {
        return instance;
    }
    // ... methods
}
```

Container for WhiteHouse