Lecture 4: The class hierarchy; static components
http://cs.cornell.edu/courses/cs2110
Announcements

- A0, HW1 due tonight
- Next week’s recitation: loop invariants

You do some stuff first

for ( … ) {
    …
}

You hope something is true

How do you know that your code is correct?
Hoare Triples

{P} C {Q}

Precondition (an assumption)

Code

Postcondition (property that is true when after code finishes)

{x = 5} x = x + 1 {x ≥ 5}

{x = 5} x = x - 1 {x ≥ 5}

There are videos to watch before recitation. Watch them before your recitation.
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)
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

```java
public class Object
```

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

**Since:**
JDK 1.0

**See Also:**
Class

## Constructor Summary

### Constructors

<table>
<thead>
<tr>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Object()</code></td>
</tr>
</tbody>
</table>

## Method Summary

### All Methods

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected <code>Object</code></td>
<td><code>clone()</code></td>
</tr>
<tr>
<td></td>
<td>Creates and returns a copy of this object.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>equals(Object obj)</code></td>
</tr>
<tr>
<td></td>
<td>Indicates whether some other object is &quot;equal to&quot; this one.</td>
</tr>
<tr>
<td>protected void</td>
<td><code>finalize()</code></td>
</tr>
<tr>
<td></td>
<td>Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.</td>
</tr>
<tr>
<td><code>Class&lt;?&gt;</code></td>
<td><code>getClass()</code></td>
</tr>
<tr>
<td></td>
<td>Returns the runtime class of this <code>Object</code>.</td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>hashCode()</code></td>
</tr>
<tr>
<td></td>
<td>Returns a hash code value for the object.</td>
</tr>
</tbody>
</table>
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;
    }

    // We often omit the Object partition to reduce clutter; we know that it is always there.
}

Java: Every class that does not extend another extends class Object.
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 2 partitions: one for JFrame methods, one for C methods

We saw this in L2!
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 accessed 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

```java
public class Apartment extends House {
    private int floor;
    private Apartment downstairs;

    //constructor
    public Apartment(int floor, Apartment downstairs) {
        floor= floor;
        downstairs = downstairs;
    }
}
```

```java
public class 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! **this** avoids overshadowed field name
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()`.
**Overriding methods**

```java
public class Apartment {
    // ...
    /** Return a representation of an Apartment */
    @Override
    public String toString() {
        return "" + (getBeds() + getBaths()) + " room apartment on " + floor + "th floor";
    }
}
```

```java
Apartment@af8
toString()  equals(Object)  hashCode()
House
bdrs 3
baths 1
House(...) getBeds() getBaths()
setBeds(...) setBaths(...)
Apartment
floor 2
upstairs
Apartment@f34
Apartment(...) isBelow(...) toString()
```

`a.toString()` calls this method
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?
  A. Triangle extends Shape { … }
  B. PHDTester extends PHD { … }
  C. BankAccount extends CheckingAccount { … }
When should you make a subclass?

Which of the following seem like reasonable designs?

A. Triangle extends Shape { ... }
   
   Yes! A triangle is a kind of shape.

B. PHDTester extends PHD { ... }
   
   No! A PHDTester “tests a” PHD, but itself is not a PHD.

C. BankAccount extends CheckingAccount { ... }
   
   No! A checking account is a kind of bank account; we likely would prefer:
   
   CheckingAccount extends BankAccount { ... }
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.*

You should make a method static if the body does not refer to any field or method in the object.
/** = “this object is below”.
  Pre: a is not null. */

public boolean isBelow(Apartment a) {
    return this == a.downstairs;
}

/** = “a is below b”.
  Pre: b and c are not null. */

public static boolean isBelow(Apartment b, Apartment a) {
    return b == a.downstairs;
}
Referencing a static method

Container for Apartment contains: objects, static components

```java
public static void main(String[] args) {
    Apartment.isBelow(a, b);
}
```
Good example of static methods

- `java.lang.Math`
  
  [http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html](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?
Use 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`
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 = ...;
...
```
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.