Today …

• Go over quiz
• Assignment 2 notes
• Start on basic class inheritance
• Applets “lab”
• Reading assignments
  – Core: Ch. 4: 144-152, 162-169
  – Core: Ch. 5: 171-182
Quiz 2 ...

• Be sure to carefully read the instructions
  – For the quiz, you were asked to find the errors,
  – And state why these are errors
  – But not fix the errors (which many tried to do …)

Quiz 2

    public class Ball {
        private static boolean inflated = true;
        private int diameter = 4;
        private void Ball(int aDiameter) {
            diameter = aDiameter;
        }
        public Ball(boolean inflate) {
            inflated = inflate;
        }
        public void inflate() {
            inflated = true;
        }
        public static void stretch() {
            diameter += 1;
        }
        public static void main(String[] args) {
            Ball b1 = new Ball(6);
            Ball b2 = new Ball(true);
            Ball b3 = new Ball(6, true);
            b2.inflate();
            b2.stretch();
        }
    }
Quiz 2

• Here are the compile errors:

```java
[sbowers]$ javac Ball.java
Ball.java:19: non-static variable diameter cannot be referenced from a static context
diameter += 1;
   ^
Ball.java:23: cannot find symbol
symbol : constructor Ball(int)
location: class Ball
   Ball b1 = new Ball(6);
      ^
Ball.java:25: cannot find symbol
symbol : constructor Ball(int,boolean)
location: class Ball
   Ball b3 = new Ball(6, true);
      ^
3 errors
```

Quiz 2

```java
public class Ball {
    private static boolean inflated = true;
    private int diameter = 4;
    private void Ball(int aDiameter) {
        diameter = aDiameter;
    }
    public Ball(boolean inflate) {
        inflated = inflate;
    }
    public void inflate() {
        inflated = true;
    }
    public static void stretch() {
        diameter += 1;
    }
    public static void main(String[] args) {
        Ball b1 = new Ball(6);
        Ball b2 = new Ball(true);
        Ball b3 = new Ball(6, true);
        b2.inflate();
        b2.stretch();
    }
}
```
Quiz 2

public class Ball {
    private static boolean inflated = true;
    private int diameter = 4;
    private void Ball(int aDiameter) {
        diameter = aDiameter;
    }
    public Ball(boolean inflate) {
        inflated = inflate;
    }
    public void inflate() {
        inflated = true;
    }
    public static void stretch() {
        diameter += 1;
    }
    public static void main(String[] args) {
        Ball b1 = new Ball(6);
        Ball b2 = new Ball(true);
        Ball b3 = new Ball(6, true);
        b2.inflate();
        b2.stretch();
    }
}

Error 1:
Non-static variable
Static method

Error 2:
No constructor

Error 3:
Constructors don’t have return types (so not a constructor!)
Quiz 2

public class Ball {
    private static boolean inflated = true;
    private int diameter = 4;
    private void Ball(int aDiameter) {
        diameter = aDiameter;
    }
    public Ball(boolean inflate) {
        inflated = inflate;
    }
    public void inflate() {
        inflated = true;
    }
    public static void stretch() {
        diameter += 1;
    }
    public static void main(String[] args) {
        Ball b1 = new Ball(6);
        Ball b2 = new Ball(true);
        Ball b3 = new Ball(6, true);
        b2.inflate();
        b2.stretch();
    }
}

OK to access static vars in constructors and in non-static methods
(not typically done, but sometimes useful)

OK to call static methods on objects
(better to call from class)

b2 is a local variable
So, it is ok to call non-static methods on b2
Checking for integer types

- In assignment 2, you must verify correct menu choices
  - But how do we check if an integer was given?
  - There are different ways to do this in Java
  - Here is one way:

Within the Integer class:

```
public static int parseInt(String s) throws NumberFormatException
```

e.g., `int val = Integer.parseInt(inputString);`

Java Overloading

- In Java, methods can be “overloaded”
  - The same method name
  - Different parameter types
  - Different number of parameters
  - In some cases, different return types ...

```
public class Dog {
    public void bark() {…}
    public void bark(int numOfBarks) {…}
    public void stay() {…}
    public void stay(Treat treat) {…}
    public void stay(int duration) {…}
    public Item fetch(ItemType type) {…}
    public Newspaper fetch() {…}
}
```

When can you not overload?

When the compiler can’t tell the methods apart ...

e.g., `void fetch(ItemType)` and `Item fetch(ItemType)`

Subclassing is OK though ...

```
public class Dog {
    public void bark() {…}
    public void bark(int numOfBarks) {…}
    public void stay() {…}
    public void stay(Treat treat) {…}
    public void stay(int duration) {…}
    public Item fetch(ItemType type) {…}
    public Newspaper fetch() {…}
}
```
Simple Collections

Arrays assume a fixed number of items
- Java has a number of built-in collection types that relax this
- For assignment 2, you should use Lists (ArrayLists)

```java
import java.util.List;
import java.util.ArrayList;

List<Todo> aList = new ArrayList<Todo>();
aList.add(item);          // add a Todo object to the list
for(Todo item : aList) {       // iterate through the list
    // do something with the item
}
aList.remove(item);        // remove an item from the list
int size = aList.size();       // get the size of the list
Todo item2 = aList.get(2);  // get the 3rd item in the list
aList.remove(2);       // remove the 3rd item from the list
```

Uses Java generics (templates)
We’ll talk more about this later
For now, just syntax...

Class Diagrams

It is often useful to use diagrams when designing your classes

```java
public class Animal {
    private int hunger;
    private Point location;

    public void eat(Food theFood) {
        ...
    }

    public void move(Point newLoc) {
        ...
    }
}
```

This is "UML - like"
Class inheritance

- Object-oriented programming languages allow you to define relationships between classes.
- Inheritance implies an “is-a” relationship.
- For example:
  - Dog “is-a” (kind of) Animal
  - Recliner “is-a” (kind of) Chair
  - SportsCar “is-a” (kind of) Vehicle
  - Manager “is-a” (kind of) Employee

Class inheritance

- What does this all mean?
  - When we say that Dog “is-a” (kind of) Animal
  - It means every Dog object is also considered an Animal object.
  - Everything an Animal object can do (or has), a Dog object can also do (or has).
**Class inheritance**

- **Think “subset” and “superset”**
- **Classes really define sets of objects**

- In this example
  - We say Dog is a “**subclass**” of Animal
  - And Animal is a “**superclass**” of Dog
  - Subclasses usually “**extend**” their superclasses
  - That is, they add fields or methods
  - They might also define different behaviors for the same methods
  - We also say Dog is a “**derived**” class
  - And Animal is a “**base**” class

---

**Class inheritance**

- **Animal**
  - int hunger
  - Point location
  - void eat(Food)
  - void move(Point)

- **Dog**
  - int barkVolume
  - void bark()
  - void rollOver()
Class inheritance

Animal
- int hunger
- Point location
- void eat(Food)
- void move(Point)

Dog
- int hunger
- Point location
- int barkVolume
- void eat(Food)
- void move(Point)
- void bark()
- void rollOver()

Conceptually, the superclass fields and methods are "inherited" by the subclass.

The subclass can only access the public and protected fields and methods though.

This denotes an Animal object.
(Notice it has specific values for the fields)
Objects include superclass "object parts"

A Dog object

: Dog

hunger = 5
Location = p1
eat(Food)
move(Point)

barkVolume = 2
bark()
rollOver()

The Animal part

Class inheritance

• This means that …
• We can treat a Dog object as if it were an Animal object
• For instance

Dog d = new Dog();
d.eat(milkBone);
Class inheritance

- The opposite does not work …
- We cannot treat an Animal object as a Dog object
- This won’t work:
  Animal a = new Animal();
  a.bark();

Objects include superclass "object parts"

A Dog object

<table>
<thead>
<tr>
<th>: Dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>hunger = 5</td>
</tr>
<tr>
<td>Location = p1</td>
</tr>
<tr>
<td>eat(Food)</td>
</tr>
<tr>
<td>move(Point)</td>
</tr>
<tr>
<td>barkVolume = 2</td>
</tr>
<tr>
<td>bark()</td>
</tr>
<tr>
<td>rollOver()</td>
</tr>
</tbody>
</table>

The Animal part

Class inheritance hierarchies

- The “is-a” relationships induce class hierarchies
  - In Java, a class can only have one superclass
  - This means the hierarchies are trees
  - We often speak of a “parent” class (immediate superclass)
  - Or of a “child” class (immediate subclass)
**Why inheritance?**

- To avoid *code duplication*
  - If you have multiple classes with the same fields and methods
  - Put the duplicated code in a common superclass
  - Avoids mistakes, makes modification easier

- You can take advantage of **polymorphism**
  - Any derived class can be substituted for the base class
  - You can do stuff like
    - Pass in a Dog object to a method that expects an Animal object
    - Or even things like this:
      ```java
      Animal[] animals = new Animals[3];
      animals[0] = new Dog();
      animals[1] = new Cat();
      animals[2] = new Wolf();
      for (Animal a : animals) // feed the animals
        a.eat(new Food());
      ```

**More on Polymorphism**

- Let's say we have this:

```
Food
  ▲
DogFood
```

- We can now do this:

```
Food milkBone = new DogTreat();
Dog d = new Dog();
d.eat(milkBone);
```

- We didn’t have to change `eat(Food)` to handle this!!!
  - Although in this case, we might want to … (more later)
Inheritance in Java

- Use the “extends” keyword

```java
public class Dog extends Animal {
    private int barkVolume;
    public void bark() {
        ... 
    }
}
```

- Java only supports “public” inheritance
  - Unlike C++, which supports public, protected, private

The Applet Lab

- Warning: This is an experiment!
Using ada in the lab

• Download putty:

• Install and run it:
  – This just means download and copy it somewhere useful
  – For example, onto the desktop

• Login to ada using putty

• Under your ~/public_html directory, create a subdirectory: cpsc324

• Create a simple HelloWorld.java program there
  – E.g., using emacs, vi, pico, or joe

Applets

• Run Java programs in a web browser

• Convert your HelloWorld program to a Java Applet!
  – Make our HelloWorld class extend JApplet
  – Modify the code to work as an Applet
  – Create an .html file and embed your Applet

• Note you may have to restart the browser after you compile again…
Your HTML File

```html
<html>
<body>
<h1>The Hello World Applet</h1>
<applet code="HelloWorld.class" width="100" height="50">
</applet>
</body>
</html>
```

Save this in your cpsc324 directory and name it "HelloWorld.html"

Your Class File

```java
import javax.swing.JApplet;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;

public class HelloWorld extends JApplet {
    static final long serialVersionUID = 13L; // for serializable
    public void init() {
        Runnable r = new Runnable() {
            public void run() {
                add(new JLabel("Hello World!", SwingConstants.CENTER));
            }
        };
        try {
            SwingUtilities.invokeLater(r);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Compile this class
Start a web browser and open
http://www.cs.gonzaga.edu/~yourlogin/cpsc324/HelloWorld.html