The Coca-Cola Company has founded Vending University. VU has two kinds of students. The full time students pay $250.00 per credit in tuition. The associates students pay $150.00 per credit in tuition. There is a difference of an abstract subclass. Provide a test program that uses polymorphism to test your classes and methods.

**Practice Problem**

The Coca-Cola Company has founded Vending University. VU has two kinds of students. The full time students pay $250.00 per credit in tuition up to a maximum of $300.00 (12 credits), even if they enroll in more than 12 credits. Tuition for students in the executive program is computed differently; these students pay $300.00 “executive fee” plus $400.00 per credit, with no ceiling or cap on the total. Each student has a name and is enrolled for some integer number of credits.

Write an abstract superclass called Student, and write concrete subclasses called FullTimeStudent and ExecutiveStudent. The method for computing the tuition should be called computeTuition().

Now do it again, but with an interface called Student instead of an abstract superclass. Provide a test program that uses polymorphism to test your classes and methods.

---

**Midterm**

- **deadline for having TAs check corrected midterms is the Thu lab tomorrow**
- **then solutions released**
- **Vista currently has unscaled, difference mark as Assignment 2 Correction**
- **after it's finalized, we'll add two more columns**
  - scaled difference
  - scaled combined

---

**Weekly Questions**

- You'll get full credit if you handed in questions for 10 (out of the 12 possible) weeks.
- Last one due today.
- Reminder: weeklies all together count for 2% of your course grade.

---

**Final Exam**

- Final review session will be Mon Apr 24.
- 10am-12pm, room WOOD 4
- Given by grad TA Primal Wijesekera
- Final is Wed Apr 28, 3:30-6:30 pm, FSC 1005
- Exam will be 2.5 hours
- 3 hour slot reserved in case of fire alarms, etc.
- Closed book/notes/laptops/calculators
- Material covered:
  - Whole course, but significant emphasis on later topics not covered in previous exams
- Exception: GUIs will not be covered.

---

**Reading Summary**


---

**Alternate Book**

- If you're not getting it and want to try a different approach, run to the bookstore (or head to Amazon.ca or Indigo.ca) and get a copy of...
  
  **Head First Java** by Kathy Sierra and Bert Bates

  Read this book, work all the problems (there are zillions), and you should have a better grasp of what's going on with Java. (I have no financial interest in this book or any bookstore.)

---

**Practice Exams**

- One practice final (without solutions) up on WebCT/Vista
- Another practice exam available under Challenge link from course page
  

---

**Exam Philosophy**

- My exams tend to be hard and long. Thus, I almost always end up grading scales.
- Difficult exams can be scaled.
- Too-easy exams cannot distinguish those who know material from those who don’t.
- How to handle exams with deliberate time pressure:
  - Do not panic if you think you won’t finish.
  - Do be strategic about how to spend your time.
  - I recommend you look through entire exam before you jump into writing answers.
  - Spend a few minutes up front to plan best approach for your strengths.

---

**Reading Summary**


---

**How To Prepare**

- Read all the required reading.
- Review lecture notes and code written in class.
- Available from web.
  

- Practice, practice, practice -- write programs!!
  - Especially using inheritance and abstract classes.

---

**Programming Practice**

- Two kinds of practice, both are important!
  - Using computer, open book, Internet, discussing approach with friends, take as long as you need to fully understand.
  - Closed book, write on paper, don’t talk to anybody about the question, time pressure.

---

**Material Covered**

- Midterm 1:
  - Primitives, constants, strings, classes, objects
- Midterm 2:
  - All of the above plus especially:
    - Conditionals, loops, arrays, sorting
    - Final
    - All of the above plus especially:
      - Interfaces, inheritance
      - More on classes, objects
      - Scope, static fields/methods, control flow
      - Pass by reference vs. pass by value.

---

**Office Hours**

- Reminder: TA office hours at DLC end Thu afternoon.
- Labs end this week.
- My office hours for rest of term:
  - Monday 4/19 4pm
  - By appointment through 4/23:
    - Send email to book
    - Not Mon 4/26
    - I'm out of town 4/24-4/27
    - Will check email at least once/day, but not online all the time.

---

**Assignments**

- Assignment 3 due Fri Apr 16, 5pm.
  - Electronic hand in only.
  - Writeup hardcopy handed out mentioned hardcopy, ignore that! (fixed in online version).
  - Assignment 2 grading reports should arrive by email very soon.
  - Ugrad account email: check it or forward it to your real account.
  - A3 grading report target is Apr 26, so you have a few days to look through before final.

---

**Recap: Inheritance Class Hierarchy**

!!![](image)

- Is base class something that you would expect to be a superclass?
- Possible to use polymorphism to test your classes and methods.

---

**Recap: Abstract Classes**

- Abstract class:
  - Not completely implemented.
  - Serve as place holders in class hierarchy.
  - Partial description inherited by all descendants.
- Usually contains one or more abstract methods.
- Has no definition: specifies method that should be implemented by subclasses.
- Just has header, does not provide actual implementation for that method.
- Abstract class uses abstract methods to specify what interface to descendant classes must look like.
- Without providing implementation details for methods that make up interface.
- Descendent classes supply additional information so that instantiation is meaningful.
Recap: Interfaces vs. Abstract Classes

- Use abstract class with inheritance to initiate a hierarchy of more specialized classes
- Use interface to say, "I need to be able to call methods with these signatures in your class."
- Use an interface for some semblance of multiple inheritance

from Just Java 2 by Peter van der Linden

Comparing Bunnies

- how to compare?
- number of carrots? location...
- names - alphabetical order!

- do we have to implement this from scratch?
- no! use String compareTo method

SortableBunny

public class SortableBunny extends NamedBunny implements Comparable {
    public SortableBunny(){
        super();
    }
    public SortableBunny(int x, int y, int carrots, String name)
    {super(x,y,carrots,name);
    }
    /* compare by name alphabetical order */
    public int compareTo(Object other){
        return this.getName().compareTo((SortableBunny)other).getName();
    }
}

SafeBunny

public class SafeBunny extends NamedBunny {
    public void getFame() {
        System.out.println("Your fame and fortune: ");
        System.out.println(getFame()+" "+getFame() + " and joy!";)
    }
}

SafeBunny crashes when we pass in String!

if (other instanceof NamedBunny) {
    if (compareTo(other) > 0) {
        return 1;
    }
    else {
        return 0;
    }
}

String solution: check type of object dynamically
- before we call bunny-specific method

Graphical User Interfaces (as much as we have time for)

This week reading: 2.11-2.12, 9.5-9.8, 10.9-10.10
- 5.1-5.2, 11.5, 12.2-12.3 (2nd edition)
- we will only get through some of this material in lecture today
  - not covered on final
  - but weekly reading question due today

reading for GUIs

- This week reading: 2.11-2.12, 9.5-9.8, 10.9-10.10
- 5.1-5.2, 11.5, 12.2-12.3 (2nd edition)
- we will only get through some of this material in lecture today
  - not covered on final
  - but weekly reading question due today

Objectives

- Taste of what's under the hood with graphical programming
  - note: taste, not mastery!

Simple Graphics

This week is all about very simple graphics in Java. What we'll talk about aren't necessarily fundamental computing concepts like loops, arrays, inheritance, and polymorphism, which surface in all sorts of different computing contexts.

This stuff will be Java-specific and may not translate well to other programming languages.
Simple Graphics
The good news is that you might find graphics more fascinating than Coke Machines.
The bad news is that Java graphics can become tedious very quickly.

Making a frame window
Step 1: Construct an object of the JFrame class.
Step 2: Set the size of the frame.
Step 3: Set the title of the frame to appear in the title bar (title bar will be blank if no title is set).
Step 4: Set the default close operation. When the user clicks the close button, the program stops running.

Making a frame window
import javax.swing.JFrame;
public class FrameViewer{
  public static void main(String[] args)
  {
    // when it's time to draw something in the frame,
    // we'll do it here
    myframe.setVisible(true);
  }
}

Making a frame window
Step 1: Construct an object of the JFrame class.
Step 2: Set the size of the frame.
Step 3: Set the title of the frame to appear in the title bar (title bar will be blank if no title is set).
Step 4: Set the default close operation. When the user clicks the close button, the program stops running.

Making a frame window
import javax.swing.JFrame;
public class FrameViewer{
  public static void main(String[] args)
  {
    // 300 pixels wide
    final int F_WIDTH = 300;
    // 400 pixels high
    final int F_HEIGHT = 400;
    myframe.setSize(F_WIDTH, F_HEIGHT);
  }
}

Now let's draw something
Wait, hold on. We don't draw anything. We create component objects (of course) and add them to the frame we've created.
We make our own component in the Swing user interface toolkit by extending the blank component called JComponent to make a RectangleComponent.
The paintComponent() method is inherited from JComponent, then we override the method with our own definition that makes a couple of rectangles.
Now let's draw something

The paintComponent() method of an object is called automatically when the frame that contains it is displayed for the first time, resized, or redisplayed after being hidden.

Here's what we drew

Now let's draw something

> java FrameViewer

Now we can draw the second and final box.

Now let's draw something

Now let's draw something

Now let's draw something

Now let's draw something

Now let's draw something

Now let's draw something

Now let's draw something

Now let's draw something

Here's what we drew

Questions?

Graphical user interfaces (GUIs)

The graphical user interface allows us to interact with our programs through mouse movements, button clicks, key presses, and so on.

Your Windows or Macintosh operating system provides you with a GUI so you don't have to remember all sorts of instructions to type at the command line.

Here's a GUI you've seen me use many times.
How do we make a GUI in Java? We install event listeners. An event listener is an object that belongs to a class which you define. The methods in your event listener contain the instructions to be executed when the events occur.

Any event listener is specific to an event source. For example, you’d have one kind of event listener to respond to the click of a button on your mouse, and another to respond to the press of a key on your keyboard.

When an event occurs, the event source calls the appropriate methods of all associated event listeners.

Here comes an example, straight from your book. This example is a simple program that prints a message when a button is clicked.

```java
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.event.ActionListener;
public class ButtonTester {
    public static void main(String[] args) {
        JFrame myframe = new JFrame();
        final int F_WIDTH = 100;
        final int F_HEIGHT = 60;
        myframe.setTitle("Button Tester");
        myframe.setSize(F_WIDTH, F_HEIGHT);
        myframe.setVisible(true);
        JButton button = new JButton("Click me!");
        myframe.add(button);
        myframe.setVisible(true);
    }
}
```

Java uses the event parameter to pass details about the event. We don’t need to worry about it.

Here’s what our example class that implements the ActionListener interface looks like:

```java
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.event.ActionListener;
public class ButtonTester {
    public static void main(String[] args) {
        JFrame myframe = new JFrame();
        JButton button = new JButton("Click me!");
        myframe.add(button);
        myframe.setVisible(true);  
    }
}
```

Next we’ll see a program that tests our ClickListener class. It looks very much like the program we wrote earlier.

First we create a frame window object so we have a place to put the button that we want to click.

Then we create a button object and add it to the frame, just like the rectangles before.

```java
public class ButtonTester2 {
    public static void main(String[] args) {
        JFrame myframe = new JFrame();
        final int F_WIDTH = 100;
        final int F_HEIGHT = 60;
        myframe.setTitle("Button Tester");
        myframe.setSize(F_WIDTH, F_HEIGHT);
        JButton button = new JButton("Click me!");
        myframe.add(button);
        myframe.setVisible(true);
    }
}
```

Finally we create an event listener object called clickListener and attach it to the button we just made.

```java
import java.awt.event.ActionListener;
public class ClickListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        System.out.println("I was clicked.");
    }
}
```

Next we’ll see a program that tests our ClickListener class. It looks very much like the program we wrote earlier.

First we create a frame window object so we have a place to put the button that we want to click.

Then we create a button object and add it to the frame, just like the rectangles before.

```java
import java.awt.event.ActionListener;
public class ClickListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        System.out.println("I was clicked.");
    }
}
```

Next we’ll see a program that tests our ClickListener class. It looks very much like the program we wrote earlier.

First we create a frame window object so we have a place to put the button that we want to click.

Then we create a button object and add it to the frame, just like the rectangles before.

```java
import java.awt.event.ActionListener;
public class ClickListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        System.out.println("I was clicked.");
    }
}
```

A button listener class like ClickListener is likely to be specific to a particular button, so we don’t really need it to be widely accessible. We can put the class definition inside the method or class that needs it. So we can put this class:

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class ClickListener extends ActionListener {
    public void actionPerformed(ActionEvent event) {
        System.out.println("I was clicked.");
    }
}
```

inside the main method of the ButtonTester class as an inner class.