University of British Columbia
CPSC 111, Intro to Computation
2009W2: Jan-Apr 2010
Tamara Munzner

Inheritance III, Graphical User Interfaces

Lecture 35, Wed Apr 14 2010

borrowing from slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/111-10

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 me 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 handin 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

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

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

Reading Summary

- http://www.cs.ubc.ca/~tmm/courses/111-10/#reading

Practice Exams

- One practice final (without solutions) up on WebCT/Vista
- Another practice exam available under Challenge link from course page http://www.ugrad.cs.ubc.ca/~cs111/

Exam Philosophy

- my exams tend to be hard and long
- thus, I almost always end up scaling marks
  - 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

How To Prepare

- Read all the required reading
- Review lecture notes and code written in class
  - available from web http://www.cs.ubc.ca/~tmm/courses/111-10/
- Practice, practice, practice -- write programs!!
  - especially using inheritance and abstract classes

Programming Practice

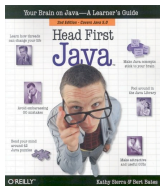
- Two kinds of practice, both are important!
  - Using computer, open book, Internet, discussing approach with friends, take as long as you need to fully understand
  - Closed book, write on paper, don't talk to anybody about the question, time pressure

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 bookseller.)



Practice Problem

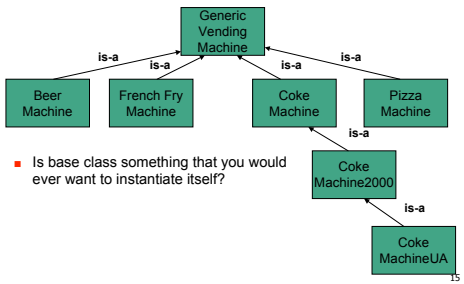
The Coca-Cola Company has founded Vending University. VU has two kinds of students. The full time students pay $250.00 per credit in tuition up to a maximum of $3000.00 (12 credits), even if they enroll in more than 12 credits. Tuition for students in the executive program is computed differently; these students pay a $3000.00 "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.

Recap: Inheritance Class Hierarchy



- Is base class something that you would ever want to instantiate itself?

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

A (Very) Last Look At Bunnies...

- interface and inheritance practice!
- let's make a SortableBunny class that both
  - extends `NamedBunny` class
  - implements `Comparable` interface
    - **`compareTo(Object o)`**
      - returns int < 0 if this object less than parameter
      - returns 0 if same
      - returns int > 0 if this object greater than parameter

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()
  );
 }
}
```


BunnySorter

```
public static void main(String[] args){
 SortableBunny[] bunnies = new SortableBunny[4];
 SortableBunny peter = new SortableBunny(3,6,1,"Peter");
 SortableBunny emily = new SortableBunny(3,4,5,"Emily");
 SortableBunny darlene = new SortableBunny(3,6,1,"Darlene");
 SortableBunny aaron = new SortableBunny(3,4,5,"Aaron");

 bunnies[0] = peter;
 bunnies[1] = emily;
 bunnies[2] = darlene;
 bunnies[3] = aaron;

 BunnySorter.sort(bunnies);
}
```


BunnySorter

```
public static void main(String[] args){
 SortableBunny[] bunnies = new SortableBunny[4];
 SortableBunny peter = new SortableBunny(3,6,1,"Peter");
 SortableBunny emily = new SortableBunny(3,4,5,"Emily");
 SortableBunny darlene = new SortableBunny(3,6,1,"Darlene");
 SortableBunny aaron = new SortableBunny(3,4,5,"Aaron");

 bunnies[0] = peter;
 bunnies[1] = emily;
 bunnies[2] = darlene;
 bunnies[3] = aaron;

 BunnySorter.sort(bunnies);

 darlene.compareTo("UhOhNotABunny");
}
```

- crashes when we pass in String!

SafeSorter

```
public static void main(String[] args){
 SafeBunny[] bunnies = new SafeBunny[4];
 SafeBunny peter = new SafeBunny(3,6,1,"Peter");
 SafeBunny emily = new SafeBunny(3,4,5,"Emily");
 SafeBunny darlene = new SafeBunny(3,6,1,"Darlene");
 SafeBunny aaron = new SafeBunny(3,4,5,"Aaron");

 bunnies[0] = peter;
 bunnies[1] = emily;
 bunnies[2] = darlene;
 bunnies[3] = aaron;

 SafeSorter.sort(bunnies);

 darlene.compareTo("UhOhNotABunny");
}
```

- no crashes. whew....

SafeBunny

```
/* compare by name alphabetical order */
/* check if it's the right type before call getName method! */
public int compareTo(Object other)
{
 if (other instanceof SafeBunny) {
   return this.getName().compareTo(
          ((SafeBunny)other).getName()
   );
 } else {
  return 0;
 }
}
```

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

Checking Type Dynamically

- **`A instanceof B`**
  - checks at execution time
  - `instanceof` is a reserved word
  - A is object
  - B is specific class name

A Final Reminder (for the Final)

- *designing* classes vs. *using* classes
  - all of our code starts with `public class Foo`
- designing classes
  - you fill in fields, method
- using classes
  - tester/driver, with main method
- examples
  - *design:* SortableBunny implements Comparable interface
  - *use:* Sorter.java code uses Integer, Double, String, SortableBunny classes and Comparable interface

Evaluations - Right Now

- official TA evaluations
  - still on paper, not online yet
- unofficial course evaluations
  - much more specific questions than the official ones
  - I do not look at these until after official ones returned, long after grades are out
- need volunteer to collect these
  - bring official ones to front desk
  - ask receptionist to put unofficial ones in my box
    - in closed envelope
- **please also** fill out official teaching surveys for instructor (me!) at the CoursEval website https://eval.olt.ubc.ca/science

Life After 111: What Next?

- two threads in CS coursework
  - continuing with programming
    - 211: Intro to Software Development
  - your first taste of theory
    - 111: Intro to Computation
- playing with computers on your own
  - fame, fortune, and joy!

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

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.

Simple Graphics

To begin with, we need a "canvas" or a "blank sheet of paper" on which to draw. In Java, this is called a frame window or just a frame. You don't put your graphics just anywhere you want...you draw them inside the frame.

It should come as no surprise that a specific frame that we draw in will be an object of some class that serves as a template for frames. Remember, nothing much happens in Java until we create objects.

Making a frame window

Step 1: Construct an object of the JFrame class.

Making a frame window

```
import javax.swing.JFrame;  //Swing is a user interface toolkit

public class FrameViewer
{
  public static void main(String[] args)
  {
    JFrame myframe = new JFrame();  // make a new JFrame object









  }
}
```


Making a frame window

Step 1: Construct an object of the JFrame class.

Step 2: Set the size of the frame.

Making a frame window

```
import javax.swing.JFrame;

public class FrameViewer
{
  public static void main(String[] args)
  {
    JFrame myframe = new JFrame();  // make a new JFrame object

    final int F_WIDTH = 300;        // 300 pixels wide
    final int F_HEIGHT = 400;       // 400 pixels high

    myframe.setSize(F_WIDTH, F_HEIGHT);









  }
}
```


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).

Making a frame window

```
import javax.swing.JFrame;

public class FrameViewer
{
  public static void main(String[] args)
  {
    JFrame myframe = new JFrame();  // make a new JFrame object

    final int F_WIDTH = 300;        // 300 pixels wide
    final int F_HEIGHT = 400;       // 400 pixels high

    myframe.setSize(F_WIDTH, F_HEIGHT);
    myframe.setTitle("My Frame");   // this is optional









  }
}
```


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)
  {
    JFrame myframe = new JFrame();  // make a new JFrame object

    final int F_WIDTH = 300;        // 300 pixels wide
    final int F_HEIGHT = 400;       // 400 pixels high

    myframe.setSize(F_WIDTH, F_HEIGHT);
    myframe.setTitle("My Frame");   // this is optional
    myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);









  }
}
```


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.

Step 5: Make the frame visible.

Making a frame window

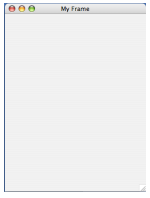
```
import javax.swing.JFrame;

public class FrameViewer
{
  public static void main(String[] args)
  {
    JFrame myframe = new JFrame();  // make a new JFrame object

    final int F_WIDTH = 300;        // 300 pixels wide
    final int F_HEIGHT = 400;       // 400 pixels high

    myframe.setSize(F_WIDTH, F_HEIGHT);
    myframe.setTitle("My Frame");   // this is optional
    myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);



    myframe.setVisible(true);
  }
}
```


Making a frame window

```
import javax.swing.JFrame;

public class FrameViewer
{
  public static void main(String[] args)
  {
    JFrame myframe = new JFrame();  // make a new JFrame object

    final int F_WIDTH = 300;        // 300 pixels wide
    final int F_HEIGHT = 400;       // 400 pixels high

    myframe.setSize(F_WIDTH, F_HEIGHT);
    myframe.setTitle("My Frame");   // this is optional
    myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // when it's time to draw something in the frame,
    // we'll do it here

    myframe.setVisible(true);
  }
}
```


Making a frame window

```
> java FrameViewer
```



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

```
import java.awt.Graphics;   // AWT is the Abstract Windowing Toolkit,
import java.awt.Graphics2D; // an older graphical user interface
import java.awt.Rectangle;  // toolkit
import javax.swing.JPanel;
import javax.swing.JComponent;

public class RectangleComponent extends JComponent
{
  public void paintComponent(Graphics g)
  {









  }
}
```

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.

Now let's draw something

```
import java.awt.Graphics;   // AWT is the Abstract Windowing Toolkit,
import java.awt.Graphics2D; // an older graphical user interface
import java.awt.Rectangle;  // toolkit
import javax.swing.JPanel;
import javax.swing.JComponent;

public class RectangleComponent extends JComponent
{
  public void paintComponent(Graphics g)
  {




  }
}
```


Now let's draw something

The paintComponent() method is passed an object of type Graphics2D, which extends the Graphics type, that contains useful information about colour and font to be used, among other things. Graphics2D provides more sophisticated methods for drawing too. But the paintComponent() method expects a parameter of the older Graphics type, so we use a cast to convert the object to Graphics2D type to recover the methods that come with the Graphics2D class.

Now let's draw something

```
import java.awt.Graphics;   // AWT is the Abstract Windowing Toolkit,
import java.awt.Graphics2D; // an older graphical user interface
import java.awt.Rectangle;  // toolkit
import javax.swing.JPanel;
import javax.swing.JComponent;

public class RectangleComponent extends JComponent
{
  public void paintComponent(Graphics g)
  {
    Graphics2D g2 = (Graphics2D) g;




  }
}
```


Now let's draw something

Now we draw a box. We give the X- and Y-coordinates of the upper left hand corner of the box, along with its width and height in pixels (i.e. picture elements).

Now let's draw something

```
import java.awt.Graphics;   // AWT is the Abstract Windowing Toolkit,
import java.awt.Graphics2D; // an older graphical user interface
import java.awt.Rectangle;  // toolkit
import javax.swing.JPanel;
import javax.swing.JComponent;

public class RectangleComponent extends JComponent
{
  public void paintComponent(Graphics g)
  {
    Graphics2D g2 = (Graphics2D) g;

    Rectangle box = new Rectangle(5, 10, 50, 75);
    g2.draw(box);




  }
}
```


Now let's draw something

The translate() method allows the programmer to start the drawing of the next box at different X- and Y-coordinates.

Now let's draw something

```
import java.awt.Graphics;   // AWT is the Abstract Windowing Toolkit,
import java.awt.Graphics2D; // an older graphical user interface
import java.awt.Rectangle;  // toolkit
import javax.swing.JPanel;
import javax.swing.JComponent;

public class RectangleComponent extends JComponent
{
  public void paintComponent(Graphics g)
  {
    Graphics2D g2 = (Graphics2D) g;

    Rectangle box = new Rectangle(5, 10, 50, 75);
    g2.draw(box);

    box.translate(80,100);



  }
}
```


Now let's draw something

Now we can draw the second and final box.

Now let's draw something

```
import java.awt.Graphics;   // AWT is the Abstract Windowing Toolkit,
import java.awt.Graphics2D; // an older graphical user interface
import java.awt.Rectangle;  // toolkit
import javax.swing.JPanel;
import javax.swing.JComponent;

public class RectangleComponent extends JComponent
{
  public void paintComponent(Graphics g)
  {
    Graphics2D g2 = (Graphics2D) g;

    Rectangle box = new Rectangle(5, 10, 50, 75);
    g2.draw(box);

    box.translate(80,100);

    g2.draw(box);
  }
}
```


Now let's draw something

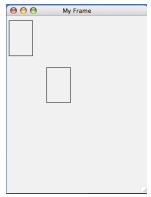
One more thing: we have to add the rectangle component to our frame object.

Now let's draw something

```
import javax.swing.JFrame;

public class FrameViewer
{
  public static void main(String[] args)
  {
    JFrame myframe = new JFrame();  // make a new JFrame object

    final int F_WIDTH = 300;        // 300 pixels wide
    final int F_HEIGHT = 400;       // 400 pixels high

    myframe.setSize(F_WIDTH, F_HEIGHT);
    myframe.setTitle("My Frame");   // this is optional
    myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    RectangleComponent component = new RectangleComponent();
    myframe.add(component);

    myframe.setVisible(true);
  }
}
```


Here's what we drew

```
> java FrameViewer
```



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.

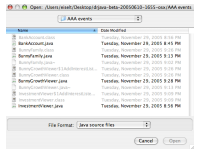
Your Windows or Macintosh operating system provides you with a GUI so you don't have to remember all sorts of instructions to type at the command line.

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.



Event handling

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.

Event handling

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

An event listener that responds to button clicks must belong to a class that implements the ActionListener interface. That interface, supplied by the Abstract Windowing Toolkit (AWT), looks like this:

```
public interface ActionListener
{
  void actionPerformed(ActionEvent event);
}
```

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

Event handling

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

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

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

The actionPerformed() method contains the instructions we want to be executed when our button is clicked.

Event handling

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.

Event handling

```
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.setSize(F_WIDTH, F_HEIGHT);
    myframe.setTitle("Button Tester");
    myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);




    myframe.setVisible(true);
  }
}
```


Event handling

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.

Event handling

```
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.setSize(F_WIDTH, F_HEIGHT);
    myframe.setTitle("Button Tester");
    myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JButton button = new JButton("Click me!");
    myframe.add(button);



    myframe.setVisible(true);
  }
}
```


Event handling

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.

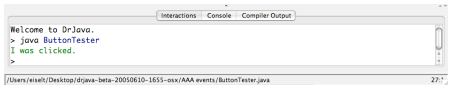
Finally we create an event listener object called **`ClickListener`** and attach it to the button we just made.

Event handling

```
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.setSize(F_WIDTH, F_HEIGHT);
    myframe.setTitle("Button Tester");
    myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JButton button = new JButton("Click me!");
    myframe.add(button);
    ActionListener listener = new ClickListener();
    button.addActionListener(listener);

    myframe.setVisible(true);
  }
}
```


Event handling

```
> java ButtonTester
```



Event handling

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:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

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

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

Event handling

```
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent; // note this addition

public class ButtonTester2
{
  public static void main(String[] args)
  {
    JFrame myframe = new JFrame();
    final int F_WIDTH = 100;
    final int F_HEIGHT = 60;
    myframe.setSize(F_WIDTH, F_HEIGHT);
    myframe.setTitle("Button Tester");
    myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JButton button = new JButton("Click me!");
    myframe.add(button);
```


Event handling

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

    ActionListener listener = new ClickListener();
    button.addActionListener(listener);

    myframe.setVisible(true);
  }
}
```