

# Mat 2170

## Chapter Two: Programming by Example

Java Programming

Spring 2012

# Student Responsibilities

- Reading: Textbook, Chapter 2

- Labs

- 1 Lab 1 : electronic submission
- 2 Lab 1 Postlab and printouts (due at beginning of Lab 2 Thursday)
- 3 Lab 2 Prelab (due at beginning of Lab 2 Thursday)
- 4 Lab 2 Postlab and printouts (due at beginning of Lab 3, next week)

- Attendance

# Chapter Two Overview

## Programming by Example

- 2.6 Graphical programs (used in Lab 2)
- 2.1 Parts of a program
- 2.2 Programming Perspectives
- 2.3 Add2Integers
- 2.4 Programming idioms and patterns
- 2.5 Classes and objects

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

## 2.6 Graphical Programs

Mat 2170

Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

- The **GraphicsProgram** class makes it possible to create simple pictures on the screen.
- The conceptual model is that of a collage composed of objects on a canvas or a felt board.
- Running a **GraphicsProgram** creates a **window** that serves as the background **canvas** for the collage.

- You cause a picture to appear by **creating** graphical objects of various kinds, and then **adding** those objects to the canvas.
- We will be learning how to work with labels (textual graphics), rectangles, ovals, and lines using the classes **GLabel**, **GRect**, **GOval**, and **GLine**.
- The complete set of graphics classes is discussed in Chap. 9.

# GLabel Objects — Unnamed vs Named

## In One Step:

**Create and Send Directly to Graphics Window:**

```
public void run()
{
    add(new GLabel("Hello, World!", 100, 75));
}
```

## In Two Steps:

**Declare a Named Object (–MyLabel–) of type GLabel, which is then sent to Graphics Window:**

```
public void run()
{
    GLabel MyLabel = new GLabel("Hello, World!", 100, 75);
    add(MyLabel);
}
```

# Sending Messages to Objects

- We may wish to change the **appearance** (color) or **location** (position) of a graphical object after it's been created.
- In object-oriented languages such as Java, these changes are the **responsibility** of the **object**.
- Thus, to change the color of an object, you send a **message** to it telling it to change color.
- At this point in the semester, **in order to send a message to an object, it must have been declared with a name** — as the GLabel1 **MyLabel** was on the last slide.

# Sending Messages to Objects

- To send a message to an object, Java uses the following syntax:

```
receiver.methodName(arguments);
```

where:

- **receiver** is the (named) object to which the message is directed
- **methodName** identifies which message is sent
- **arguments** is a list of values used to specify any other information associated with the message

# Sending Messages to a GLabel

This program illustrates sending a message to an object. Note that the label **doesn't appear** until it is added to the canvas.

---

```
public class HelloProgram extends GraphicsProgram
{
    public void run()
    {
        GLabel MyLabel = new GLabel("Hello",100,75);
        MyLabel.setFont("SansSerif-36");
        MyLabel.setColor(Color.RED);
        add(MyLabel);
    }
}
```

---

contents of **MyLabel**:

**"Hello", 100, 75**

# The Java Coordinate System

Mat 2170

Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

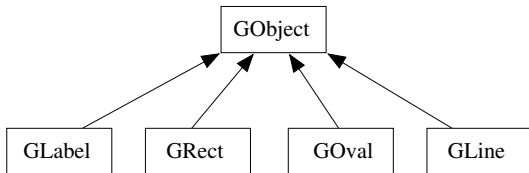
Classes

- Positions and distances in a graphics program are measured in terms of **pixels**, which are the individual dots that cover the screen.
- Unlike traditional mathematics, Java defines the **origin** of the graphics coordinate system to be the upper left corner of the window.

- Values for the  $x$  coordinate increase from left to right.
- Values for the  $y$  coordinate increase from **top to bottom**.
- Creating a `GLabel` at a particular  $x$  and  $y$  position means that the baseline of the first character in the label appears at that point — i.e., the  $(x, y)$  coordinate is for the **lower left** corner of the label.

# The GObject Hierarchy

The classes that represent graphical objects form a hierarchy, part of which looks like this:



- Operations are defined at each level of the hierarchy.
- Operations that apply to all graphical objects are specified at the GObject level — where they are inherited by each subclass.
- Operations that apply to a particular subclass are specified as part of the definition of that class.

# Operations on the GObject Class

The following operations apply to **all** GObject

**object.setColor(color)**

Sets the color of the object to the specified color constant (default is **BLACK**)

**object.setLocation(x, y)**

Changes the location of the object to the point (x, y)

**object.move(dx, dy)**

Moves the object on the screen by adding the **displacements** dx and dy to its current coordinates

# The java.awt Package Standard Color Names

Mat 2170

Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

Color.BLACK

Color.DARK\_GRAY

Color.GRAY

Color.LIGHT\_GRAY

Color.WHITE

Color.MAGENTA

Color.PINK

Color.RED

Color.YELLOW

Color.GREEN

Color.CYAN

Color.BLUE

Color.ORANGE

In order to use these colors, you will need to add:

```
import java.awt.*;
```

to your program.

# Operations on the GLabel Class

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

**GLabel**

Methods

Programs

Patterns

Classes

## Constructor:

```
new GLabel(text, x, y)
```

Creates a label containing the specified text that begins at the point (x, y).

# GLabel Example

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

**GLabel**

Methods

Programs

Patterns

Classes

```
public void run()  
{  
    GLabel msg = new GLabel("Hello, World!", 100, 75);  
    add(msg);  
}
```

# Creating Geometric Objects

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

## Constructors:

```
new GRect(x, y, width, height)
```

Creates a rectangle whose upper left corner is at (x, y ) of the specified size.

```
new GOval(x, y, width, height)
```

Creates an oval that fits inside the rectangle with the same dimensions.

```
new GLine(x0, y0, x1, y1)
```

Creates a line extending from (x0, y0) to (x1, y1).

# Methods Shared by the GRect and G Oval Classes

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

```
object.setFilled(fill)
```

If **fill** is **true**,  
the interior of the object is shaded;  
if **false**, only the outline is shown.

```
object.setFillColor(color)
```

Sets the color used to fill the interior,  
which can be different from the border.

# The GRectPlusGOval Program

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

```
public class GRectPlusGOval
    extends GraphicsProgram
{
    public void run()
    { // Create & draw a red rectangle
        GRect MyRect = new GRect(100, 50, 125, 60);
        MyRect.setFilled(true);
        MyRect.setColor(Color.RED);
        add(MyRect);

        // Create & draw a green oval inside
        GOval MyOval = new GOval(100, 50, 125, 60);
        MyOval.setFilled(true);
        MyOval.setFill(Color.GREEN);
        add(MyOval);
    }
}
```

# Resulting Output

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

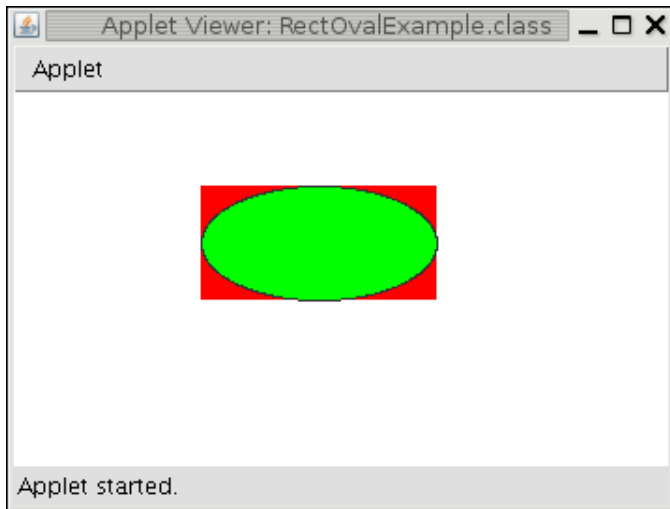
GLabel

Methods

Programs

Patterns

Classes



# Questions, Questions, Questions...

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

- What would we need to change in the last program to have the rectangle be drawn **after** the oval? What would happen in that case?
- What would we need to change if we wanted a cyan oval with a blue border?

- What is the difference between the **setColor()** and the **setFillColor()** methods for GRects and GOvals?
- What GraphicsProgram method displays a GObject in the graphics window?
- Suppose we'd like to draw a stick figure in a graphics window. How could we go about determining the coordinates of the parts?

# The Stick Figure

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

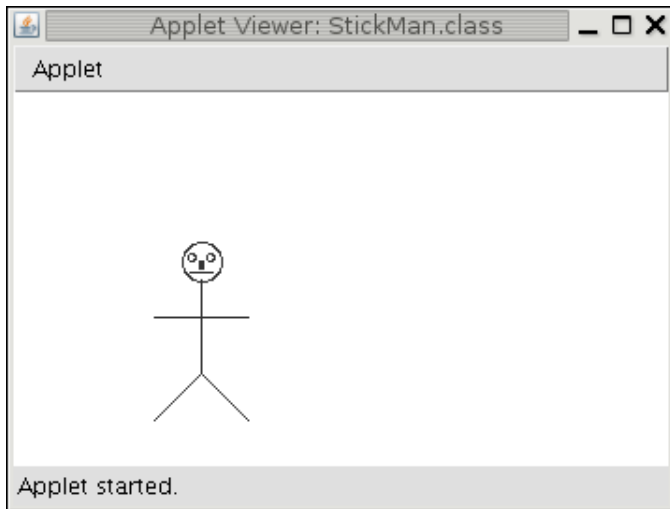
GLabel

Methods

Programs

Patterns

Classes



# Comments Are Desirable

```
// body
add(new GLine(100, 100, 100, 150));
// legs
add(new GLine(100, 150, 75, 175));
add(new GLine(100, 150, 125,175));
// arms
add(new GLine(75,120,125,120));
// head
add(new G Oval(90, 80, 20, 20));
// eyes
add(new G Oval(93,86,3,3));
add(new G Oval(103,86,3,3));
//nose
add(new GLine(99,90,99,94));
add(new GLine(100,90,100,94));
add(new GLine(101,90,101,94));
add(new GLine(101,90,101,94));
```

**What must change to make a Blue Man?**

## 2.1 Parts of a Java Program

```
/* file: HelloProgram.java */
import acm.graphics.*;
import acm.program.*;

public class HelloProgram
    extends GraphicsProgram {
    public void run() {
        // create and display a greeting
        add(new GLabel("hello, world", 100, 75));
    }
}
```

- Header Comments, line comments (for humans)
- Imports: Allows use of shorter names for library classes
- The main class – HelloProgram – and its run() method

# Comments

- Comments are for humans; computer ignores them
- **Block comments:**
  - use `/*` and `*/` around text
  - can extend over several lines
- **Line comments:**
  - start with `//`
  - extend only to end of line

# A Program to Add Two Numbers — Dialog

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

```
/* file: Add2Integers.java */
import acm.program.*;

public class Add2Integers
    extends DialogProgram
{
    public void run()
    {
        println("This program adds two integers");
        int n1 = readInt("Enter first number: ");
        int n2 = readInt("Enter second number: ");
        int total = n1 + n2;
        println("The total is " + total + ".");
    }
}
```

# A Program to Add Two Numbers — Console

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

```
/* file: Add2Integers.java */
import acm.program.*;

public class Add2Integers
    extends ConsoleProgram
{
    public void run()
    {
        println("This program adds two integers");
        int n1 = readInt("Enter first number: ");
        int n2 = readInt("Enter second number: ");
        int total = n1 + n2;
        println("The total is " + total + ".");
    }
}
```

# A Program to Add Two Numbers — Floating Point

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

```
/* file: Add2Doubles.java */
import acm.program.*;

public class Add2Doubles
    extends ConsoleProgram
{

    public void run()
    {
        println("This program adds two floating " +
                "point values");
        double n1 = readDouble("Enter first number: ");
        double n2 = readDouble("Enter second number: ");
        double total = n1 + n2;
        println("The total is " + total + ".");
    }
}
```

# Two Perspectives on the Programming Process

- **Reductionism:** a whole can be understood completely if you understand its parts and the nature of their 'sum'.

When programming, you need to understand the **language** (such as Java) well before you can write correct, efficient programs.

- **Holism:** the whole is greater than the sum of its parts.

When programming, you need to be able to see the **logical** "big picture" and create the underlying logic (algorithm) before you can write correct, efficient programs.

## 2.4 Programming Idioms and Patterns — A Holistic Approach to Programming

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

**Patterns**

Classes

- There are a variety of common operations
- A standard solution strategy exists for common operations
- The code that implements such a solution strategy is called a **programming idiom** or **programming pattern**
- Learning to use these patterns saves time

- For example, it is helpful to think of a statement like:

```
int n1 = readInt("Enter first number: ");
```

as a holistic **pattern** to read an integer from the user:

```
int variable = readInt("prompt");
```

- Then, switching to **floating point** values is much easier:

```
double variable = readDouble("prompt");
```

- The concept or pattern is the same
- The pattern serves as a template
- Don't have to remember so many details
- Recognize pattern and apply standard solution strategy
- This approach is scalable

## 2.5 Classes and Objects

- Java programs are written as **collections of classes**, which serve as templates for individual objects.
- Each object is an **instance** of a particular class.
- Classes can serve as a pattern for many different objects.
- Java classes form hierarchies — like a family tree.
- Except for the class named `Object` at the top of the hierarchy, every Java class is a **subclass (derived)** of some other **superclass (parent class)**.
- A class can have many subclasses, but only one superclass.

# Inheritance

- A class represents a **specialization** of its **superclass**.
- If you create an **object** that is an **instance** of a class, that object is also an instance of all other classes in the hierarchy above it in the superclass chain.
- When a new Java class is defined, that class automatically **inherits the behavior** of its superclass.
- The structure of Java's class hierarchy resembles the biological classification scheme which subdivides species to reflect anatomical similarities.

# Biological Class Hierarchy

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

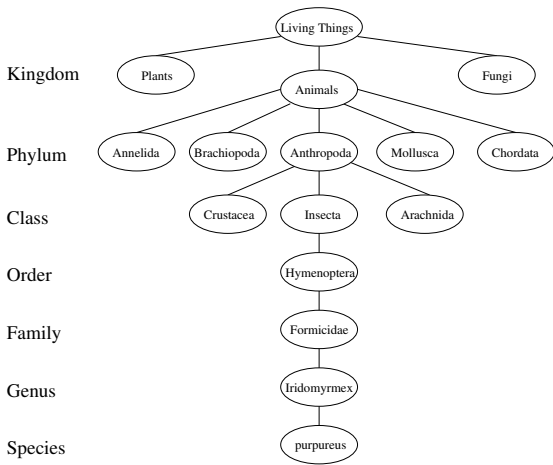
GLabel

Methods

Programs

Patterns

Classes



# The Red Ant



Classification of the red ant *Iridomyrmex purpureus*



Note that there can be many individual red ants, each of which is an **instance** of the same basic class.

# Java Class Hierarchies

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

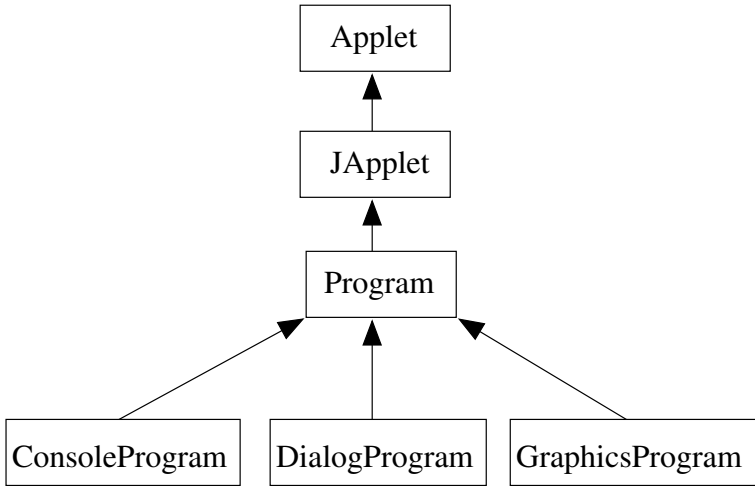
Programs

Patterns

Classes

- Java class hierarchies are similar to the biological class hierarchy.
- For example, on the next slide, we see the program hierarchy formed by the classes in the `acm.program` package.
- Every `ConsoleProgram` is also a `Program`, a `JApplet`, and an `Applet`.
- This means that every `ConsoleProgram` can (or is suppose to be able to) run as an applet on the web.
- The same is true for any `DialogProgram` or `GraphicsProgram`.

# The Program Hierarchy



Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

# The DialogProgram Class

Mat 2170  
Chapter Two:  
Programming  
by Example

Java  
Programming

Week2

Graphics

Messages

Coordinates

Hierarchy

Colors

GLabel

Methods

Programs

Patterns

Classes

- The **class definition** specifies the **behavior** of objects of that class.
- The `DialogProgram` class has exactly the **same operations** as the `ConsoleProgram` class — the only difference is that the input and output operations use pop-up dialogs instead of a console.