Eric Roberts
CS 106A

Handout #12
April 9, 2012

Simple Java

Simple Java

Eric Roberts
CS 106A
April 9, 2012

The "Hello World" Program

One of the important influences on the design of Java was the C programming language, which was developed at Bell Labs in the early 1970s. The primary reference manual for C was written by Brian Kernighan and Dennis Ritchie.

On the first page of their book, the authors suggest that the first step in learning any language is to write a simple program that prints the message "hello, world" on the display. That advice remains sound today.

1.1 Getting Started

The only way to learn a new programming language is to write programs in it. The first program to write is the same for all languages:

Print the words
`hello, world`

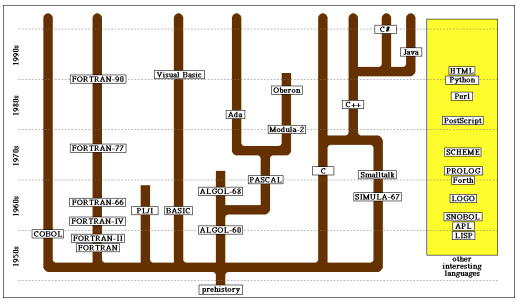
This is the big hurdle; to leap over it you have to be able to create the program text somewhere, compile it, load it, run it, and find out where your output went. With these mechanical details mastered, everything else is comparatively easy.

In C, the program to print "`hello, world`" is

```
#include <stdio.h>

main() {
    printf("hello, world");
}
```

Evolution of Computer Languages



The 2002 ACM Turing Award

Kristen Nygaard

The most prestigious prize in computer science, the ACM Turing Award, was given in 2002 to two Norwegian computing pioneers, who developed the first object-oriented programming language in 1967.

Ole Johan Dahl

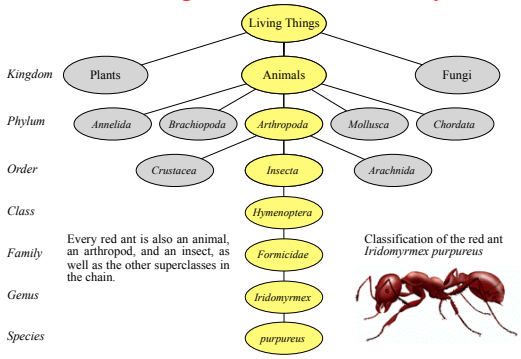
Systems of Classification

In the mid-18th century, the Scandinavian botanist Carl Linnaeus revolutionized the study of biology by developing a new system for classifying plants and animals in a way that revealed their structural relationships and paved the way for Darwin's theory of evolution a century later.

Linnaeus's great contribution was to recognize that organisms fit into a hierarchical classification scheme in which the placement of individual species within the hierarchy reflects their anatomical similarities.

Carl Linnaeus (1707–1778)

Biological Class Hierarchy



Every red ant is also an animal, an arthropod, and an insect, as well as the other superclasses in the chain.

Classification of the red ant *Iridomyrmex purpureus*

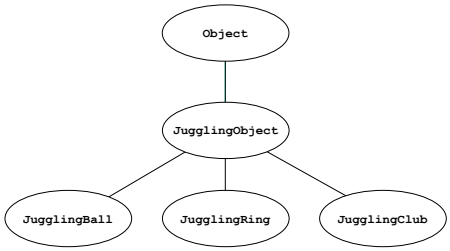
Instances *vs.* Patterns

Drawn after you, you pattern of all those.
—William Shakespeare, Sonnet 98

In thinking about any classification scheme—biological or otherwise—it is important to draw a distinction between the category defined by a particular class and specific instances of that class. In the most recent example, the designation *Iridomyrmex purpureus* is not itself an ant, but rather a ***class*** of ant. There can be (and usually are) many ants, each of which is an individual of that class.

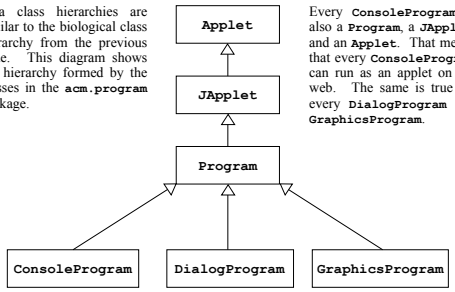
Each of these fire ants is an ***instance*** of the general category encompassing all ants of its class. Each instance is of the species *purpureus,* the genus *Iridomyrmex,* the family *Formicidae* (which makes it an ant), and so on. Thus, each ant is not only an ant, but also an insect, an arthropod, and an animal.

Another Hierarchy Example



The Program Hierarchy

Java class hierarchies are similar to the biological class hierarchy from the previous slide. This diagram shows the hierarchy formed by the classes in the `acm.program` package.



Every `ConsoleProgram` is also a `Program`, a `JApplet`, and an `Applet`. That means that every `ConsoleProgram` can run as an applet on the web. The same is true for every `DialogProgram` and `GraphicsProgram`.

Hello World as a Console Program

```
import acm.program.*;

public class HelloProgram extends ConsoleProgram {
   public void run() {
      println("hello, world");
   }
}
```

Hello World as a Dialog Program

```
import acm.program.*;

public class HelloProgram extends DialogProgram {
   public void run() {
      println("hello, world");
   }
}
```

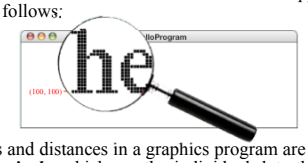
Hello World as a Graphics Program

```
import acm.graphics.*;
import acm.program.*;

public class HelloProgram extends GraphicsProgram {
   public void run() {
      add( new GLabel("hello, world", 100, 75) );
   }
}
```

The Java Coordinate System

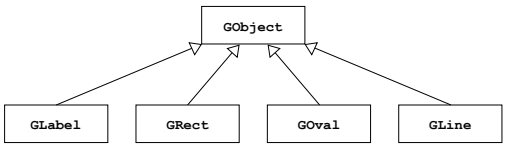
- Creating a **`JLabel`** at a particular *x* and *y* position means that the baseline of the first character in the label appears at that point, as follows:



- 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 coordinate system to be in the upper left corner. Values for the *x* coordinate increase as you move rightward across the screen; *y* coordinate values increase as you move downward.

The `GObject` Hierarchy

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



The **`GObject`** class represents the collection of all graphical objects. The four subclasses shown in this diagram correspond to particular types of objects: labels, rectangles, ovals, and lines. The class diagram makes it clear that any **`GLabel`**, **`GRect`**, **`GOval`**, or **`GLine`** is also a **`GObject`**.

Operations on the `GObject` Class

The following operations apply to all **`GObject`**s:

object`.setColor(`*color*`)` Sets the color of the object to the specified color constant.
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 *dx* and *dy* to its current coordinates.

The standard color names are defined in the **`java.awt`** package:

`Color.BLACK`	`Color.RED`	`Color.BLUE`
`Color.DARK_GRAY`	`Color.YELLOW`	`Color.MAGENTA`
`Color.GRAY`	`Color.GREEN`	`Color.ORANGE`
`Color.LIGHT_GRAY`	`Color.CYAN`	`Color.PINK`
`Color.WHITE`		

Drawing Geometrical Objects

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(`x_0`,` y_0`,` x_1`,` y_1`)` Creates a line extending from (x_0, y_0) to (x_1, y_1).

Methods shared by the **`GRect`** and **`GOval`** classes

object`.setFilled(`*fill*`)` If *fill* is **`true`**, fills in the interior of the object; if **`false`**, shows only the outline.
object`.setFillColor(`*color*`)` Sets the color used to fill the interior, which can be different from the border.

Operations on the `GLabel` Class

Constructor

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

Methods specific to the **`GLabel`** class

label`.setFont(`*font*`)` Sets the font used to display the label as specified by the font string.

The font is typically specified as a string in the form

`"`*family*`-`*style*`-`*size*`"`

where

family is the name of a font family
style is either **`PLAIN`**, **`BOLD`**, **`ITALIC`**, or **`BOLDITALIC`**
size is an integer indicating the point size