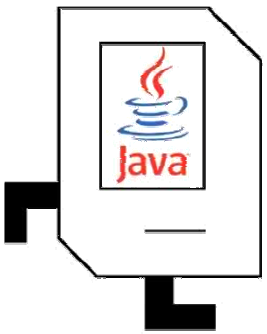
CS 106A, Lecture 27
"Real" Java

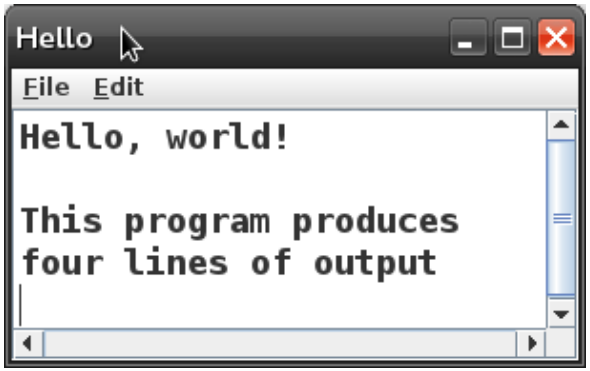
(without Stanford libraries)

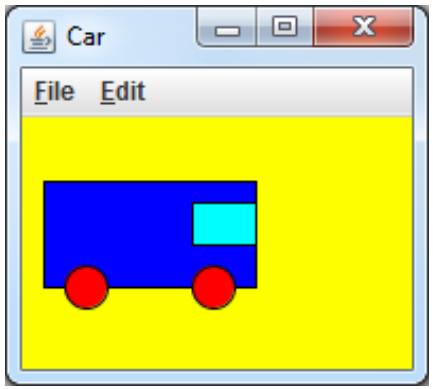
reading: none

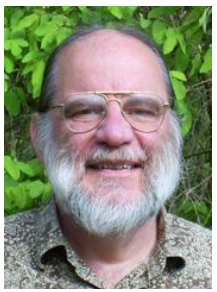
Lecture at a glance

- All quarter we have used a set of **Stanford/ACM Java libraries**.
  - Karel, ConsoleProgram, RandomGenerator
  - GraphicsProgram, GOval, GRect, GOval, GLine, GImage, ...
- These were built by Prof. Roberts to simplify Java programming.
  - Also facilitates new kinds of programs, e.g. **graphics**/animation
- Today we will see how **standard Java** programs are made.
  - Most features are still there, but require more cumbersome syntax.









Stanford CS Prof.
Eric Roberts

A Stanford program

```
import acm.program.*;

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

- This is a console program written using the Stanford libraries.
  - It uses the **`ConsoleProgram`** class to represent a console.
  - The **`run`** method contains the program code.
  - The **`println`** method prints output to the graphical console.

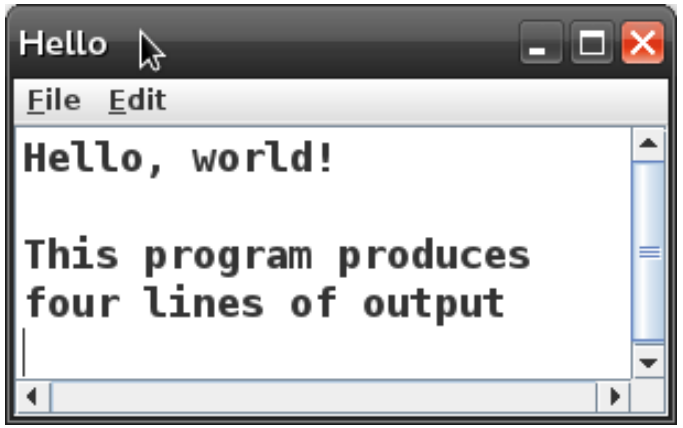
A Java program

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

- The method **`main`** is the true entry point for a Java program.
  - It must have the exact heading shown above.
  - The `String[]` **`args`** are "command line arguments" (ignored).
  - The **`println`** command's true name is `System.out.println`.
  - Standard Java methods are **`static`** unless part of a class of objects.

The ConsoleProgram class

- What does the **`ConsoleProgram`** library class do?
  - Creates a new graphical **window**
  - Puts a scrollable **text area** into it
  - Provides `print` and `println` commands to send text **output** to that window
  - contains a **`main` method** that calls your program class's `run` method
    - `ConsoleProgram`'s `run` is empty, but you extend and override it




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

ACM console input

```
public class Age extends ConsoleProgram {
    public void run() {
        String name = readLine("What's your name? ");
        int age = readInt("How old are you? ");
        int years = 65 - age;
        println(name + " has " + years
                + " years until retirement!");
    }
}
```

- The Stanford library has simple console input commands like `readLine`, `readInt`, `readDouble`, and so on.
- These methods display a 'prompt' message, wait for input, re-prompt if the user types a bad value, and return the input.

Java console input

```
public class Age {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("What's your name? ");
        String name = console.nextLine();
        System.out.print("How old are you? ");
        int age = console.nextInt();
        int years = 65 - age;
        System.out.println(name + " has " + years
                + " years until retirement!");
    }
}
```

- In "real" Java, you must create a `Scanner` or similar object to read input from the console, which is also called `System.in`.
  - It does not automatically re-prompt and can crash on bad input.

ACM random numbers

```
import acm.util.*;

public class DiceRoller extends ConsoleProgram {
    public void run() {
        RandomGenerator rg = RandomGenerator.getInstance();
        int sum = 0;
        while (sum != 7) {
            int d1 = rg.nextInt(1, 6);   // roll the dice
            int d2 = rg.nextInt(1, 6);
            sum = d1 + d2;
            println(d1 + "+" + d2 + "=" + sum);
        }
    }
}
```

- The ACM libraries use a **`RandomGenerator`** class with methods for getting random integer, double, boolean, etc. values.

Java random numbers

```
import java.util.*;

public class DiceRoller {
    public static void main(String[] args) {
        Random randy = new Random();
        int sum = 0;
        while (sum != 7) {
            int d1 = randy.nextInt(6) + 1;  // roll the dice
            int d2 = randy.nextInt(6) + 1;
            sum = d1 + d2;
            System.out.println(d1 + "+" + d2 + "=" + sum);
        }
    }
}
```

- Java has a very similar **Random** class, but its `next` methods take only a *max* and return a value *r* in range $0 \le r < max$.
  - To adjust the range/probability, you may need to add or multiply.

Complete ACM program

```
public class Boxes extends ConsoleProgram {
    public void run() {
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }

    public void line(int count) {              // Prints the given
        for (int i = 1; i <= count; i++) {     // number of stars
            print("*");                        // plus a line break.
        }
        println();
    }

    public void box(int width, int height) {
        line(width);
        for (int line = 1; line <= height - 2; line++) {
            print("*");
            for (int space = 1; space <= width - 2; space++) {
                print(" ");
            }                                  // Prints a box of *
            println("*");                      // of the given size.
        }
        line(width);
}   }
```

Complete Java program

```
public class Boxes {
    public static void main(String[] args) {
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }

    public static void line(int count) {      // Prints the given
        for (int i = 1; i <= count; i++) {    // number of stars
            System.out.print("*");            // plus a line break.
        }
        System.out.println();
    }

    public static void box(int width, int height) {
        line(width);
        for (int line = 1; line <= height - 2; line++) {
            System.out.print("*");
            for (int space = 1; space <= width - 2; space++) {
                System.out.print(" ");
            }                                 // Prints a box of *
            System.out.println("*");          // of the given size.
        }
        line(width);
}   }
```

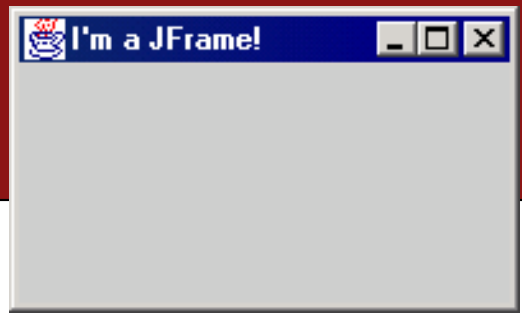
ACM GUIs

```
import acm.program.*;
import acm.gui.*;          // Stanford graphic components
import java.awt.*;         // Java graphical objects
import java.awt.event.*; // for Java events
import javax.swing.*;      // Java graphical objects

public class Name extends Program {
    public void init() {
        statements;
        addActionListeners();   // enable action events
    }

    // the code to run when the event occurs
    public void actionPerformed(ActionEvent event) {
        ...
    }
}
```

ACM GUI to Java GUI

- The ACM library actually does several things here:
  - Automatically creates and displays a **window** on the screen.
    - In standard Java, we must do this ourselves; it is called a `JFrame`.
  - Sets up a **drawing canvas** in the center of the window (if it is a `GraphicsProgram`).
    - In standard Java, we must create our own drawing canvas.
  - Provides convenient methods to listen for action and mouse events.
    - In standard Java, event handling takes a bit more code to set up.

JFrame



a graphical window to hold other components

`new JFrame("`***title***`")`	creates a new window with the given title
jf`.add(`***component***`);`	adds a GUI component to the window
jf`.pack();`	sizes the frame exactly to fit its contents
jf`.remove(`***component***`);`	removes a GUI component from the window
jf`.setLayout(`***layout***`);`	sets strategy used to position components
jf`.setLocation(`***x***`, `***y***`);`	sets (x, y) position of window on screen
jf`.setResizable(`***boolean***`);`	sets whether the window can be resized
jf`.setSize(`***w***`, `***h***`);`	sets window size in pixels
jf`.setVisible(`***boolean***`);`	pass `true` to show window on screen
jf`.setDefaultClose Operation(`***operation***`);`	Instructs Java what to do when the window is closed; by default, does nothing. Pass `JFrame.EXIT_ON_CLOSE` to tell the program to shut down when the window is closed.

Real events

- In real Java, when you want your class to listen to events, you must specify that your class implements a listener interface:

```
public class MyGUI implements ActionListener { ...
public class MyGUI implements MouseListener { ...
```

- The Stanford methods `addActionListeners();` and `addMouseListeners();` are not present in normal Java, so you must **manually add event listening** to each graphical component.

```
JButton button = new JButton("Click me!");
button.addActionListener(this);
...
```

ACM GUI example

```
public class ColorFun extends Program {
    public void init() {
        JButton button1 = new JButton("Red!");
        JButton button2 = new JButton("Blue!");
        add(button1, SOUTH);
        add(button2, SOUTH);
        addActionListeners();
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getActionCommand().equals("Red!")) {
            setBackground(Color.BLUE);
        } else {
            setBackground(Color.RED);
        }
    }
}
```

Java GUI example

```
public class ColorFun implements ActionListener {
    public static void main(String[] args) {
        new ColorFun().init();
    }

    private JFrame frame;

    public void init() {
        frame = new JFrame("ColorFun");
        frame.setSize(500, 300);
        JButton button1 = new JButton("Red!");
        JButton button2 = new JButton("Blue!");
        button1.addActionListener(this);
        button2.addActionListener(this);
        frame.add(button1, "South");
        frame.add(button2, "South");
        frame.setVisible(true);
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getActionCommand().equals("Red!")) {
            frame.setBackground(Color.BLUE);
        } else {
            frame.setBackground(Color.RED);
}   }   }
```

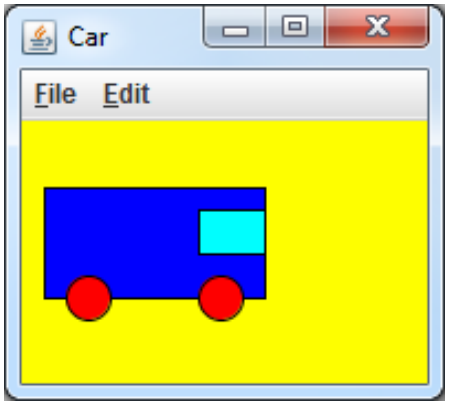
Java drawing canvas

- This is one area where the Stanford/ACM library is a big help.
- A graphical canvas in Java is represented by **extending `JPanel`**.
  - `JPanel` is an empty graphical component.
  - You can override its method **`paintComponent`** to draw shapes on the component.
  - The code is a bit odd and uses different graphical commands.
  - It is not object-oriented.
  - It does not offer methods for checking for collisions, mouse events, or several other features. You would need to implement that yourself.

- Since `JPanel` is so different from the ACM graphics library, we won't show every detail, just one quick example of a basic canvas.

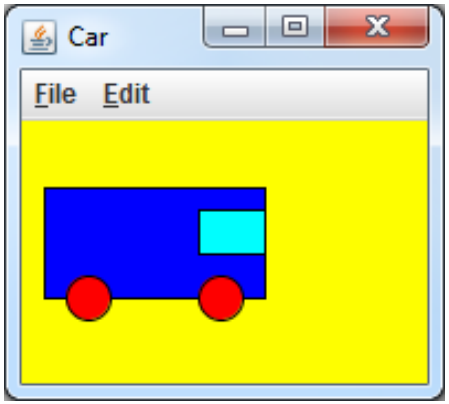
ACM GraphicsProgram

```
public class Car extends GraphicsProgram {
    public void run() {
        setSize(200, 180);
        setBackground(Color.YELLOW);

        GRect body = new GRect(10, 30, 100, 50);
        body.setFilled(true);
        body.setFillColor(Color.BLUE);
        add(body);

        GOval wheel1 = new GOval(20, 70, 20, 20);
        wheel1.setFilled(true);
        wheel1.setFillColor(Color.RED);
        add(wheel1);

        GOval wheel2 = new GOval(80, 70, 20, 20);
        wheel2.setFilled(true);
        wheel2.setFillColor(Color.RED);
        add(wheel2);

        GRect windshield = new GRect(80, 40, 30, 20);
        windshield.setFilled(true);
        windshield.setFillColor(Color.CYAN);
        add(windshield);
}   }
```



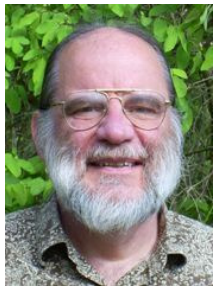
Java JPanel

```
public class CarCanvas extends JPanel {
    public CarCanvas() {
        setPreferredSize(new Dimension(200, 180));
        setBackground(Color.YELLOW);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        g.setColor(Color.BLUE);
        g.fillRect(10, 30, 100, 50);  // body

        g.setColor(Color.RED);
        g.fillOval(20, 70, 20, 20);   // wheel 1
        g.fillOval(80, 70, 20, 20);   // wheel 2

        g.setColor(Color.CYAN);
        g.fillRect(80, 40, 30, 20);
    }
}

public class Car {
    // create a JFrame, add a CarCanvas to it, show it, ...
    ...
}
```



Summary

- **Benefits of libraries:**
  - simplify syntax/rough edges of language/API
  - avoid re-writing the same code over and over
  - possible to make advanced programs quickly
  - leverage work of others



Stanford CS Prof.
Eric Roberts

- **Drawbacks of libraries:**
  - learn a "dialect" of the language ("Stanford/ACM Java" vs. "real Java")
  - lack of understanding of how lower levels or real APIs work
  - some libraries can be buggy or lack documentation
  - limitations on usage; e.g. Stanford/ACM library cannot be re-distributed for commercial purposes, so you can't use it at a startup :-(

Export to JAR

- **JAR**: Java Archive. A compressed binary of a Java program.
  - The typical way to **distribute a Java app as a single file**.
  - Essentially just a ZIP file with Java .class files in it.

- Making a JAR of your project in Eclipse:
  - File → Export ... →
    Java → **Runnable JAR File**

- *see handout on course web site*