Today

- Programming Assignment 6: Spampede! (10/20)
- Worksheet 2 due today...

On the horizon:

Wed 10/18 “Parsing” -- programming computers to understand text.
Fri 10/20 Spampede applet officially due (actually due Sun.)
Sun 10/22 Review session for midterm exam (time pref.)?
Mon 10/23 After class -- take-home exam given out (2 hr.)
Wed 10/25 Before class -- take-home exam due by 2:30pm.

x = 'N';
if (x == 'N') x = 'S';
if (x == 'S') x = 'N';

ouch!
Interfaces  “Classes without code”

interface KeyListener
{
    void keyPressed(KeyEvent evt);
    void keyReleased(KeyEvent evt);
    void keyTyped(KeyEvent evt);
}

interface Runnable
{
    void run();
}

interface ActionListener
{
    void actionPerformed(ActionEvent evt);
}

Idea: programming by contract
Events

- Events are things that happen to a graphical application
  - Button Presses
  - Text Entries
  - Key Presses, Key Releases, Key Events

- Each object receiving an event notifies its “Listener”
- The Listener then handles the event appropriately
Today

• Homework 6: Applet!
  *
  Using the Deque class to model a centipede...

• Midterm exam in class on: Wednesday, March 13
Today

• “Double secret extension” … for HW5’s Ex.Cr. to this weekend

• Homework 6: Applet!

  * Using the Deque class to model a centipede...

• Midterm exam in class on: Wednesday, March 13
Deque D1 = new Deque();
Deque D2 = new Deque();

D1.enqueue("0");
D1.enqueue("1");
D1.enqueue("2");
D2.enqueue("3");
D2 = D1;

D2.dequeue();
D2.enqueueFront("I");
D1.enqueue("N");
D2.enqueue("Q");

System.out.println(D1);
System.out.println(D2);
Commenting of each method is important, but those comments don’t have to be particularly lengthy:

```
// what the functions below do
// shall remain shrouded in mystery...

public static boolean is_empty(Queue Q) {  // S D Q M T ?
  return Q.isEmpty();
}

public boolean is_empty() {  // I 1 2 C F Z Q S M T .
  return (front == null && back == null);
}
```

Credits: Jim Norwood
Inheritance summary

**Ideas**

- Models the “kind-of” relationship among classes
- Factors out common code from those classes
- Function overriding allows old code to call new code

**Keywords**

- `extends`, `super`, `implements`

*used with interfaces, not necessarily inheritance...*
Inheritance Application: Applets!

- Applet: A small application designed to run in a browser
Java’s Graphics Support: AWT

- **java.awt 1.0**
  - Original version, accepted by most web browsers

- **java.awt 1.1**
  - Updated version
  - one difference: cleaner event handling
  - “event listeners”
  - supported by IE and Netscape by default

- **java.swing 1.2, 1.3, 1.4**
  - monstrous library of components
  - not supported by any browser by default
AWT Components

- **canvas** (for drawing)
- **applet** (“is a” panel)
- **panel** (contains other components)
Spampede Example (before)

http://www.cs.hmc.edu/~situdent/Spampede.html
import java.applet.*; /* and other fine imported files */
import java.awt.*;
import java.awt.event.*;

public class Spampede extends Applet
implements ActionListener, KeyListener, Runnable

{
    Graphics graphics; // where drawing takes place
    Image image; // off-screen image
    private Button redButton;
    private Button greenButton;
    private Button startButton;
    private Button pauseButton;
    private TextField textInput;
}
public void init()
{
    image = createImage(getSize().width, getSize().height); // double buffer
    graphics = image.getGraphics(); // for drawing
    clear();

    this.addKeyListener(this);

    redButton = new Button("Red"); // initialize things here… .
    redButton.addActionListener(this);
    redButton.addKeyListener(this);
    add(redButton);

    ...

    textInput = new TextField("Maze Name",25);
    textInput.addActionListener(this);
    add(textInput);
}

• No constructor needed -- use init()
Interfaces

```
interface KeyListener
{
    void keyPressed(KeyEvent evt);
    void keyReleased(KeyEvent evt);
    void keyTyped(KeyEvent evt);
}

interface Runnable
{
    void run();
}

interface ActionListener
{
    void actionPerformed(ActionEvent evt);
}
```

If you *claim* you’ll implement them, you *have* to implement them.

(They can be empty methods, but they have to exist.)

Idea: programming *by contract*
Events

• Events are things that happen to a graphical application
  • Button Presses  • Text Entries
  • Key Presses, Key Releases, Key Events

• Each object receiving an event notifies its “Listener”

• The Listener then handles the event appropriately

```java
public void keyPressed(KeyEvent evt)
{
    graphics.setColor(Color.white);
    graphics.fillRect(300,180,100,40);
    graphics.setColor(Color.black);
    graphics.drawString("Key " + evt.getKeyChar() + " pressed",300,200);
    repaint();
}
```
Avoid “magic numbers”!

even Prof. Benjamin agrees...

```
static final int EAST = 0;
static final int WEST = 1;
static final int NORTH = 2;
static final int SOUTH = 3;
```

These constants make code easier to read and write, e.g.,

```
int currentHeading = EAST;
```
Drawing Calls

• The drawing commands are encapsulated in the Graphics class (graphics is the data member’s name)

void setColor(Color c)

void fillRect(int x, int y, int width, int height)
void fillOval(int x, int y, int width, int height)
void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)

Each of the above have “draw” versions:
drawRect, drawOval, drawPolygon

void drawString(String str, int x, int y)
void drawLine(int x1, int y1, int x2, int y2)
void drawImage(Image img, int x, int y, null)
Drawing

some event occurs, such as calling **repa\text\texttt{int}()** or making the window visible

**update**() is called on the window’s graphics, which then calls **paint**()

whatever is drawn in **paint** is displayed

These methods can be overridden.
Double Buffering

to avoid flicker

individual drawing commands

Off screen buffer

raster copy

Browser window

fillRect ...

public void update(Graphics g) {
    paint(g);
}

image

public void paint(Graphics g) {
    g.drawImage(image, 0, 0, null);
}

graphics

repaint()
Reuse! (others’ experience)

You need an HTML file to load your applet

```html
< HTML >
< APPLET CODE = “Spampede.class”
   WIDTH = 700
   HEIGHT = 500 >

< /APPLET >

< /HTML >
```

Use the Java Console ← anything printed will go there

Use shift-reload in Netscape, control-refresh in IE

Example applets available at www.cs.hmc.edu/~cs60grad
Reuse! (others’ code)

Component
  Container
    Panel
      Applet
        Spampede

- public Graphics getGraphics()
- public Dimension getSize()
- public void addKeyListener()
- public void paint()
- public void addNotify()
- public void update()
- public void add(Component c)
- public void addNotify()
- public void init()
- public void init()
- public void paint()
Bottom-up Software Strategies

Incremental Changes

CODE

TEST

concentrated despair phase

code
test
code
test
code
test
...
closing in on a solution
Building tools

KEY What will have to be done many times?
THEN write a method (function) to handle it.

Test each one!
Threads

Getting two programs for the price of one
Each thread is considered an independent process
They alternate in controlling the applet.
When they alternate is not specified (in general).

Abstraction

event handler (default thread)
centipede and spam updater

Implementation

event handler (default thread)
centipede and spam updater
Threads

Potential Problem 1
“The Corruptor”

Thread 1 is changing an object (say, adding 1 to the score) and gets switched out.

Thread 2 takes over and also changes the same object (say, subtracting 1 from the score).

What can go wrong?

How to add 1:

1. load `score` to a register (holding pen)
2. add 1 to the register
3. store the register’s new contents into `score`

“synchronized” makes code atomic
Threads

Thread 1 is holding an object, like the “Score” object, and needs another object (say, a network connection), so it sleeps (or switches out).

Thread 2 takes over and starts using the network connection when it realizes it needs to update the Score object. Since the Score object is locked by another thread, Thread 2 goes to sleep (or switches out), and...

Result: **Deadlock**