# ECE 462
# Object-Oriented Programming using C++ and Java

## Lecture 17

**Yung-Hsiang Lu**

**yunglu@purdue.edu**

# Program Performance

- Why is performance important?
  - Users expect more and better features $\Rightarrow$ programs are more and more complex
  - Response time can determine market success $\Rightarrow$ if your program is too slow, you can lose customers.
  - Programs are ported to resource-limited systems, such as cellular phones.
  - Processor speed stops improving, in terms of the clock speed. New processors have more cores.
- What should we do?
  - Algorithmic improvement $\Rightarrow$ ECE368
  - Programming techniques $\Rightarrow$ understand different choices

# How to Improve Performance

- use compiler optimization, g++ -O and javac –O
- avoid IO: print to screen, save to disk, send to / receive from network
- avoid creating / destroying objects, pass by reference
- reduce the overhead in checking error conditions

```
if (common case) { handle the case }
else {
    if (error1) { handle error case 1 }
    if (error2) { handle error case 2 }
    if (error3) ...
```

# Virtual Function

- Does a virtual function take longer than a non-virtual function? Yes
- Does it take longer for a virtual function deeper in class hierarchy?

```
class C1 {
   virtual void func() { ... }
}
class C2: public C1 {
  virtual void func() { ... }
}
class C3: public C2 {
  virtual void func() { ... }
}
```

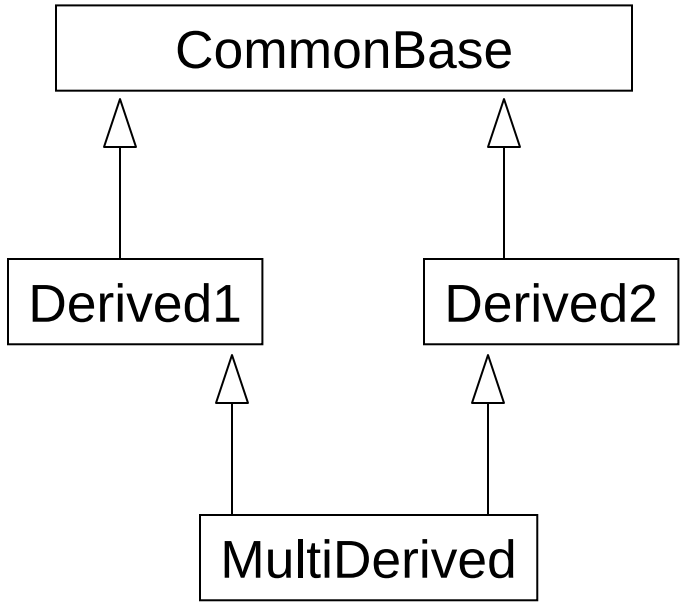
```
C1 * c1obj = new C1;
c1obj -> func();

C1 * c13obj = new C3;
c13obj -> func();

C3  * c3obj = new C3;
c3obj -> func();
```

Do they take the same amount of time? Yes

# Performance Cost of Multiple Inheritance

Does it take longer to call a virtual function in MultiDerived than in Derived1 or in Derived2?

No



# Avoid Temporary Objects

- Are these two code segments the same? All are string objects.

```
s6 = s1 + s2 + s3 + s4 + s5;
```

```
s7 = s1;
s7 += s2;
s7 += s3;
s7 += s4;
s7 += s5;
```

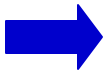
- Yes, in functionality (s6 == s7)
- No, in performance. Which one is faster?

For s6 = s1 + s2 + s3 + s4 + s5, the compiler has to create a temporary object t1 to hold the result of s1 + s2, another temporary object t2 for t1 + s3, another object t3 for t2 + s4 ... It is **faster** to obtain **s7**.

# Allocate Memory First

If an operation requires frequent memory allocation / release, try to calculate the needed amount and allocate it right away.

```
snew = s1;
snew += s2;
snew += s3;
snew += s4;
snew += s5;
....
```



```
snew.resize(s1.length() + s2.length() + ...);
index = 0;
snew.replace(index, s1.length(), s1);
index += s1.length();
snew.replace(index, s2.length(), s2);
index += s2.length();
....
```

# Profile Program Performance

- C++: Use profile tools, such as **gprof**

  > g++ -O **-p** source.cpp -o executable

  > executable

  > gprof executable > profile

  ⇒ percentage of time spent in each function

  > man gprof for the options

  > g++ -O -fprofile-arcs -ftest-coverage src.cpp -o dest

  > dest

  > **gcov** src.cpp

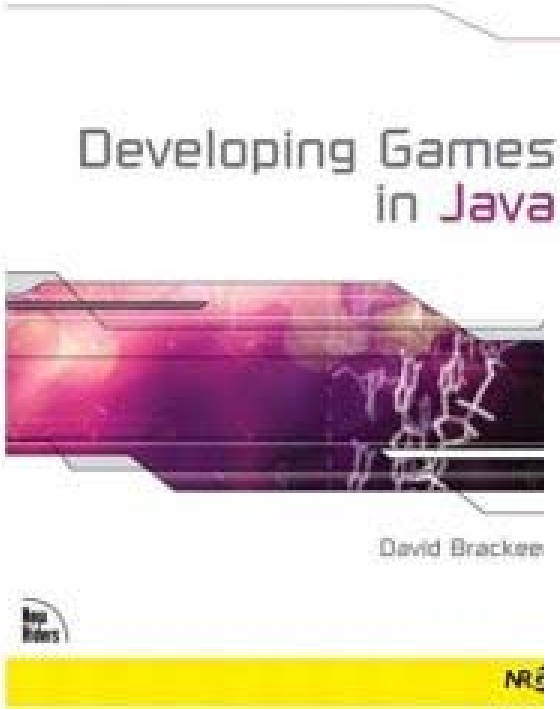
  > less src.cpp.gcov

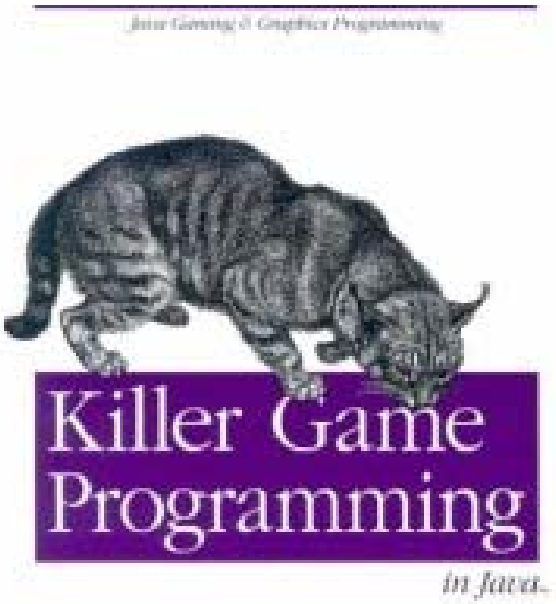
- Java:

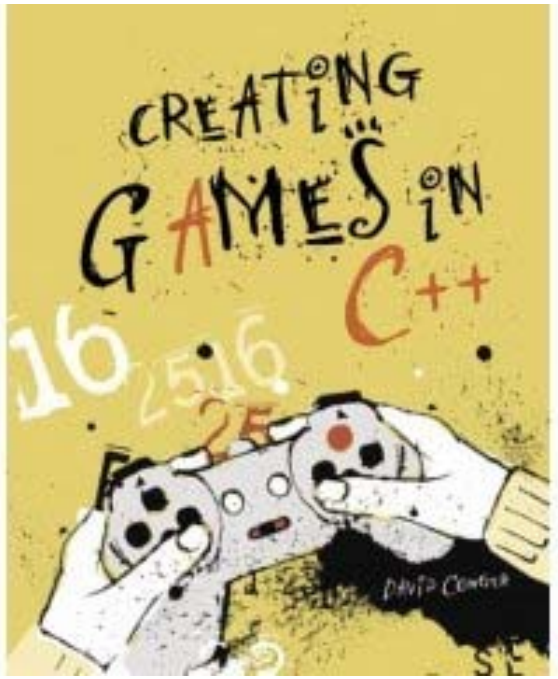
  > java -Xprof ajavaclass

# Developing Complex Programs
## (Using Games as Examples)

**Reference Books**
**(purchase not necessary)**
**Learn the languages**
**in a "bigger" context:**
**Games**



# Java Games

- Applet games: running through web browser
  - + no installation needed, easy upgrade (the web version is always the latest)
  - – security restrictions, cannot save game status
- Window games:
  - + no restrictions like applets
  - – players may be distracted by other windows
- Full-screen games:
  - + no other program can appear to distract players
  - – do not allow players to engage in other activities (such as instant messaging)

# Full Screen Game

```
// Modified from Developing Games in Java by David Brackeen
// http://www.brackeen.com/javagamebook/#download
import java.awt.*;
import javax.swing.JFrame;
public class FullScreenTest extends JFrame {
    private int xpos;
    private int ypos;
    private int xdir;
    private int ydir;
    private int width;
    private int height;
    public static void main(String[] args) {
        DisplayMode displayMode;
```

```
        if (args.length == 3) {
            displayMode =
              new DisplayMode(
                                  Integer.parseInt(args[0]),
                                  Integer.parseInt(args[1]),
                                  Integer.parseInt(args[2]),
                                  DisplayMode.REFRESH_RATE_UNKNOWN);
        }
        else {
            displayMode = new
              DisplayMode(800, 600, 16,
                          DisplayMode.REFRESH_RATE_UNKNOWN);
        }
        FullScreenTest test =
         new FullScreenTest(displayMode.getWidth(),
                            displayMode.getHeight());
        test.test(displayMode);
    }
```

```
private static final long DEMO_TIME = 50000;
public FullScreenTest(int w, int h) {
 xdir = 1;
 ydir = 1;
 width = w;
 height = h;
 xpos = width / 7;
 ypos = height / 11;
}
public void test(DisplayMode displayMode) {
    setBackground(Color.blue);
    setForeground(Color.white);
    setFont(new Font("Dialog", 0, 24));
    SimpleScreenManager screen = new SimpleScreenManager();
    try {
        screen.setFullScreen(displayMode, this);
```

```
        try {    Thread.sleep(DEMO_TIME);        }
        catch (InterruptedException ex) { }
    }
    finally {  screen.restoreScreen();    }
}
public void paint(Graphics gfx) {
 gfx.setColor(Color.blue);
 gfx.drawString("x= " + xpos + " y= " + ypos, 50, 50);
 gfx.fillOval(xpos, ypos, 40, 40);
 xpos += (int) ( Math.random() * 3) * xdir;
 ypos += (int) ( Math.random() * 3) * ydir;
 if (xpos < 20) { xpos = 20; xdir = - xdir; }
 if (xpos > (width - 40)) { xpos = width - 40; xdir = - xdir; }
 if (ypos < 20) { ypos = 20; ydir = - ydir; }
 if (ypos > (height - 40)) { ypos = height - 40; ydir = - ydir; }
```

```
        gfx.setColor(Color.white);
        gfx.fillOval(xpos, ypos, 40, 40);
        gfx.drawString("x= " + xpos + " y= " + ypos, 50, 50);
        // add some delays
        long nextTime = System.currentTimeMillis() + 1;
        while (System.currentTimeMillis() < nextTime) { }
        repaint();
    }
}
```

```
import java.awt.*;
import javax.swing.JFrame;
/**
    The SimpleScreenManager class manages initializing and
    displaying full screen graphics modes.
*/
public class SimpleScreenManager {
    private GraphicsDevice device;
    /**        Creates a new SimpleScreenManager object.     */
    public SimpleScreenManager() {
        GraphicsEnvironment environment =
            GraphicsEnvironment.getLocalGraphicsEnvironment();
        device = environment.getDefaultScreenDevice();
    }
    /**
        Enters full screen mode and changes the display mode.
    */
```

```
public void setFullScreen(DisplayMode displayMode, JFrame window)
{
    window.setUndecorated(true);
    window.setResizable(false);
    device.setFullScreenWindow(window);
    if (displayMode != null &&
        device.isDisplayChangeSupported())
    {
        try {       device.setDisplayMode(displayMode);           }
        catch (IllegalArgumentException ex) {
            // ignore - illegal mode for this device
        }
    }
}
/**
    Returns the window currently used in full screen mode.
*/
```

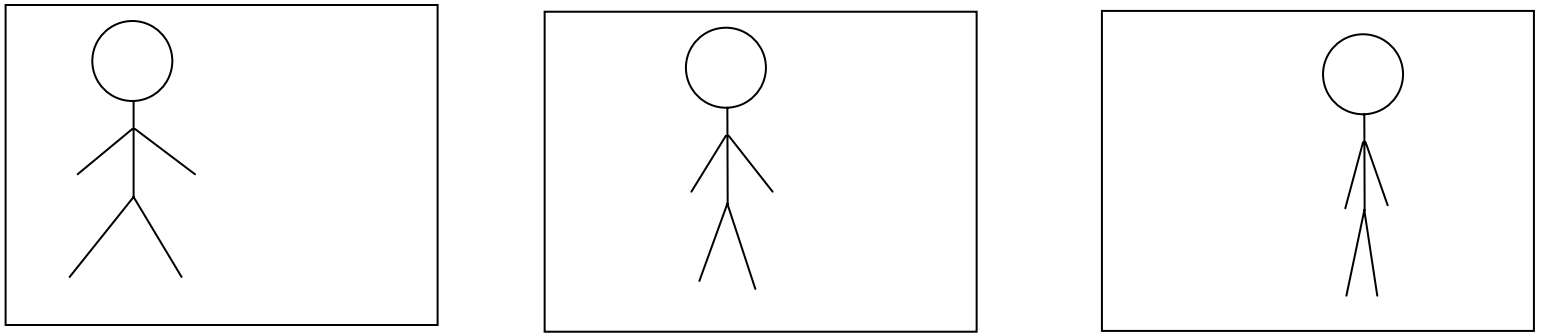
```
    public Window getFullScreenWindow() {
        return device.getFullScreenWindow();
    }
    /**
        Restores the screen's display mode.
    */
    public void restoreScreen() {
        Window window = device.getFullScreenWindow();
        if (window != null) {
            window.dispose();
        }
        device.setFullScreenWindow(null);
    }
}
```

# Determine Parameters

- DisplayMode(width, height, bitDepth, refreshRate)
  - typical resolutions: 800x600 and 1024x768. higher resolution = better image quality, if the system is sufficiently fast.
  - bit depth = 16 or higher better color quality, 8 is too low. high quality color = 8 bits for R, G, and B (24 bits total)
  - REFRESH_RATE_UNKNOWN usually works well. Human eyes cannot detect refresh rate higher than 75Hz

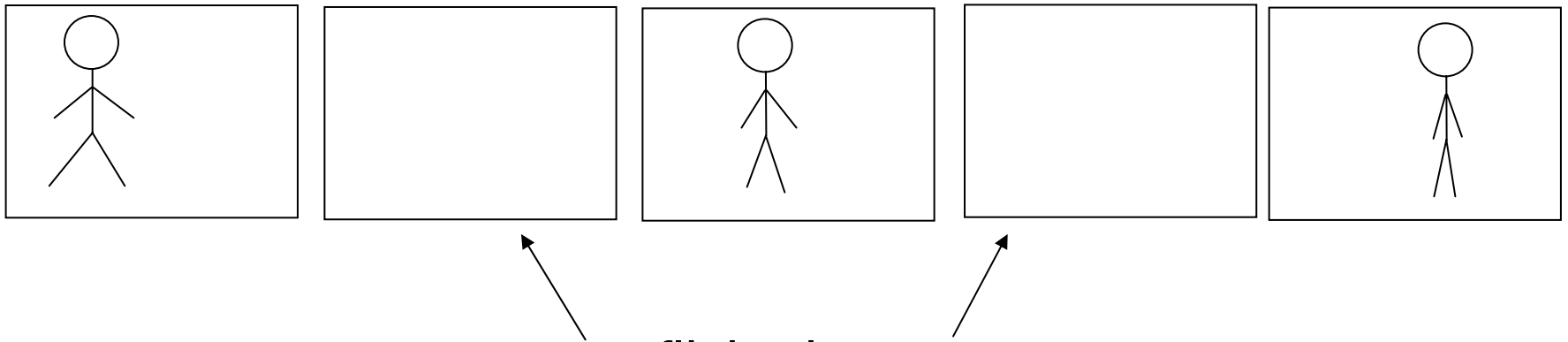
# Create Motion (Animation)

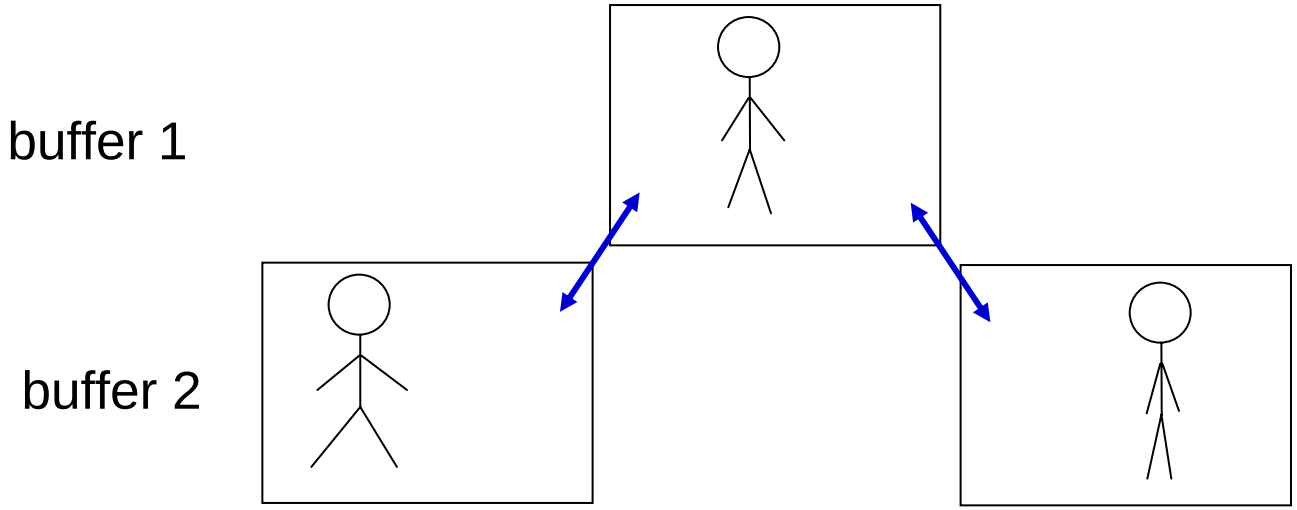
- Draw an object
- Erase the object in the original location
- Draw it in the new location
- Repeat



# Flickering and Double Buffering

- If erasing occurs in the same image that is being shown, users can see the object disappearing for a short period of time $\Rightarrow$ flickering
- Solution: double buffering (images)
  - show a completed image with the object already drawn
  - have a second image on which the object is erased, drawn at the new location
  - show the second image to user
  - repeat between the two images


flickering


buffer 1
buffer 2

# Animation with Double Buffering

```
import java.awt.Image;
import java.util.ArrayList;
/** The Animation class manages a series of images (frames) and the
    amount of time to display each frame.  */
public class Animation {
    private ArrayList frames;
    private int currFrameIndex;
    private long animTime;
    private long totalDuration;
    /** Creates a new, empty Animation.  */
    public Animation() {
        frames = new ArrayList();
        totalDuration = 0;
        restart();
    }
```

```
/** Adds an image to the animation with the specified duration
    (time to display the image).  */
public synchronized void addFrame(Image image, long duration)
{
    totalDuration += duration;
    frames.add(new AnimFrame(image, totalDuration));
}
/** Starts this animation over from the beginning.  */
public synchronized void restart() {
    animTime = 0;
    currFrameIndex = 0;
}
/** Updates this animation's current image (frame),  if neccesary.  */
public synchronized void update(long elapsedTime) {
    if (frames.size() > 1) {
        animTime += elapsedTime;
        if (animTime >= totalDuration) {
```

```
            animTime = animTime % totalDuration;
            currFrameIndex = 0;
        }
        while (animTime > getFrame(currFrameIndex).endTime) {
            currFrameIndex++;
        }
    }
}
/** Gets this Animation's current image. Returns null if this
    animation has no images.  */
public synchronized Image getImage() {
    if (frames.size() == 0) {
        return null;
    }
    else {
        return getFrame(currFrameIndex).image;
    }
}
```

```
    private AnimFrame getFrame(int i) {
        return (AnimFrame)frames.get(i);
    }
    private class AnimFrame {
        Image image;
        long endTime;
        public AnimFrame(Image image, long endTime) {
            this.image = image;
            this.endTime = endTime;
        }
    }
}
```

```
import java.awt.*;
import javax.swing.ImageIcon;
public class AnimationTest2 {
    public static void main(String args[]) {
        AnimationTest2 test = new AnimationTest2();
        test.animate();
    }
    private static final DisplayMode POSSIBLE_MODES[] = {
        new DisplayMode(800, 600, 32, 0),
        new DisplayMode(800, 600, 24, 0),
        new DisplayMode(800, 600, 16, 0),
        new DisplayMode(640, 480, 32, 0),
        new DisplayMode(640, 480, 24, 0),
        new DisplayMode(640, 480, 16, 0)
    };
    private static final long DEMO_TIME = 40000;
    private ScreenManager screen;
```

```
private Image bgImage;
    private Animation anim;
    public void loadImages() {
        // load images
        bgImage = loadImage("images/background.jpg");
        Image player1 = loadImage("images/player1.png");
        Image player2 = loadImage("images/player2.png");
        Image player3 = loadImage("images/player3.png");
        // create animation
        anim = new Animation();
        anim.addFrame(player1, 250);
        anim.addFrame(player2, 150);
        anim.addFrame(player1, 150);
        anim.addFrame(player2, 150);
        anim.addFrame(player3, 200);
        anim.addFrame(player2, 150);
    }
```

```
private Image loadImage(String fileName) {
        return new ImageIcon(fileName).getImage();
    }
    public void animate() {
        screen = new ScreenManager();
        try {
            DisplayMode displayMode =
                screen.findFirstCompatibleMode(POSSIBLE_MODES);
            screen.setFullScreen(displayMode);
            loadImages();
            animationLoop();
        }
        finally {
            screen.restoreScreen();
        }
    }
```

```
public void animationLoop() {
    long startTime = System.currentTimeMillis();
    long currTime = startTime;
    while (currTime - startTime < DEMO_TIME) {
        long elapsedTime =  System.currentTimeMillis() - currTime;
        currTime += elapsedTime;
        // update animation
        anim.update(elapsedTime);
        // draw and update screen
        Graphics2D g = screen.getGraphics();
        draw(g);
        g.dispose();
        screen.update();
        // take a nap
        try {               Thread.sleep(20);          }
        catch (InterruptedException ex) { }
    }
 }
```

```
    public void draw(Graphics g) {
        // draw background
        g.drawImage(bgImage, 0, 0, null);
        // draw image
        g.drawImage(anim.getImage(), 0, 0, null);
    }
}
```

# Implementing Double Buffering

```
import java.awt.*;
import java.awt.image.BufferStrategy;
import java.awt.image.BufferedImage;
import java.lang.reflect.InvocationTargetException;
import javax.swing.JFrame;
/** The ScreenManager class manages initializing and displaying full
    screen graphics modes.  */
public class ScreenManager {
    private GraphicsDevice device;
    /** Creates a new ScreenManager object.  */
    public ScreenManager() {
        GraphicsEnvironment environment =
            GraphicsEnvironment.getLocalGraphicsEnvironment();
```

```
    device = environment.getDefaultScreenDevice();
}
/** Returns a list of compatible display modes for the default
    device on the system.  */
public DisplayMode[] getCompatibleDisplayModes() {
    return device.getDisplayModes();
}
/** Returns the first compatible mode in a list of modes.  Returns
    null if no modes are compatible.  */
public DisplayMode findFirstCompatibleMode(
    DisplayMode modes[])
{
    DisplayMode goodModes[] = device.getDisplayModes();
    for (int i = 0; i < modes.length; i++) {
        for (int j = 0; j < goodModes.length; j++) {
            if (displayModesMatch(modes[i], goodModes[j])) {
                return modes[i];
```

```
            }
        }
    }
    return null;
}
/** Returns the current display mode.  */
public DisplayMode getCurrentDisplayMode() {
    return device.getDisplayMode();
}
/** Determines if two display modes "match". Two display modes
    match if they have the same resolution, bit depth, and refresh
    rate. The bit depth is ignored if one of the modes has a bit
    depth of DisplayMode.BIT_DEPTH_MULTI.  Likewise, the refresh
    rate is ignored if one of the modes has a refresh rate of
    DisplayMode.REFRESH_RATE_UNKNOWN.  */
public boolean displayModesMatch(DisplayMode mode1,
    DisplayMode mode2)
{
```

```
    if (mode1.getWidth() != mode2.getWidth() ||
        mode1.getHeight() != mode2.getHeight())
    {           return false;       }
    if (mode1.getBitDepth() != DisplayMode.BIT_DEPTH_MULTI &&
        mode2.getBitDepth() != DisplayMode.BIT_DEPTH_MULTI &&
        mode1.getBitDepth() != mode2.getBitDepth())
    {           return false;       }
    if (mode1.getRefreshRate() !=
        DisplayMode.REFRESH_RATE_UNKNOWN &&
        mode2.getRefreshRate() !=
        DisplayMode.REFRESH_RATE_UNKNOWN &&
        mode1.getRefreshRate() != mode2.getRefreshRate())
     {          return false;       }
     return true;
}
```

```
/** Enters full screen mode and changes the display mode.  If the
    specified display mode is null or not compatible with this
    device, or if the display mode cannot be changed on this
    system, the current display mode is used.  <p> The display
    uses a BufferStrategy with 2 buffers.  */
public void setFullScreen(DisplayMode displayMode) {
    final JFrame frame = new JFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setUndecorated(true);
    frame.setIgnoreRepaint(true);
    frame.setResizable(false);
    device.setFullScreenWindow(frame);
    if (displayMode != null &&
        device.isDisplayChangeSupported())
    {
        try {               device.setDisplayMode(displayMode);          }
        catch (IllegalArgumentException ex) { }
```

```
    // fix for mac os x
    frame.setSize(displayMode.getWidth(),
        displayMode.getHeight());
}
// avoid potential deadlock in 1.4.1_02
try {
    EventQueue.invokeAndWait(new Runnable() {
        public void run() {
            frame.createBufferStrategy(2);
        }
    });
}
catch (InterruptedException ex) {
    // ignore
}
```

```
            catch (InvocationTargetException  ex) {
            // ignore
        }
    }
    /** Gets the graphics context for the display. The ScreenManager
        uses double buffering, so applications must call update() to
        show any graphics drawn.  <p> The application must dispose of
        the graphics object.  */
    public Graphics2D getGraphics() {
        Window window = device.getFullScreenWindow();
        if (window != null) {
            BufferStrategy strategy = window.getBufferStrategy();
            return (Graphics2D)strategy.getDrawGraphics();
        }
        else {            return null;        }
    }
    /** Updates the display.  */
```

```
public void update() {
    Window window = device.getFullScreenWindow();
    if (window != null) {
        BufferStrategy strategy = window.getBufferStrategy();
        if (!strategy.contentsLost()) {
            strategy.show(); // swap buffers
        }
    }
    // Sync the display on some systems.
    // (on Linux, this fixes event queue problems)
    Toolkit.getDefaultToolkit().sync();
}
/** Returns the window currently used in full screen mode.
    Returns null if the device is not in full screen mode.  */
public JFrame getFullScreenWindow() {
    return (JFrame)device.getFullScreenWindow();
}
```

```
/** Returns the width of the window currently used in full screen
    mode. Returns 0 if the device is not in full screen mode.  */
public int getWidth() {
    Window window = device.getFullScreenWindow();
    if (window != null) {           return window.getWidth();       }
    else {          return 0;       }
}
/** Returns the height of the window currently used in full screen
    mode. Returns 0 if the device is not in full screen mode.  */
public int getHeight() {
    Window window = device.getFullScreenWindow();
    if (window != null) {           return window.getHeight();       }
    else {          return 0;       }
}
/** Restores the screen's display mode.  */
public void restoreScreen() {
```

```
        Window window = device.getFullScreenWindow();
        if (window != null) {            window.dispose();        }
        device.setFullScreenWindow(null);
    }
    /** Creates an image compatible with the current display.  */
    public BufferedImage createCompatibleImage(int w, int h,
        int transparancy)
    {
        Window window = device.getFullScreenWindow();
        if (window != null) {
            GraphicsConfiguration gc =
                window.getGraphicsConfiguration();
            return gc.createCompatibleImage(w, h, transparancy);
        }
        return null;
    }
}
```

```
public void draw(Graphics2D g) {
    // draw background
    g.drawImage(bgImage, 0, 0, null);
    AffineTransform transform = new AffineTransform();
    for (int i = 0; i < NUM_SPRITES; i++) {
        Sprite sprite = sprites[i];
        // translate the sprite
        transform.setToTranslation(sprite.getX(),  sprite.getY());
        // if the sprite is moving left, flip the image
        if (sprite.getVelocityX() < 0) {
            transform.scale(-1, 1);
            transform.translate(-sprite.getWidth(), 0);
        }
        // draw it
        g.drawImage(sprite.getImage(), transform, null);
    }
}
```

# ECE 462
# Object-Oriented Programming using C++ and Java

## Lecture 18

**Yung-Hsiang Lu**

**yunglu@purdue.edu**

# Lab 10 Creating Moving and Rotating Balls with Changing Background

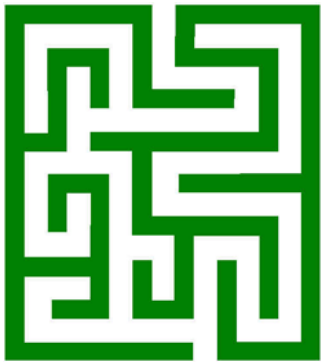
# Procedure

- **Download source code from http://www.brackeen.com/javagamebook/#download**
- **Enter chapter 02**
- **Modify SpriteTest2**
- **Download the images from /home/shay/a/sfwtools/public/data**
- **different numbers of balls**
- **translate (i.e. move)**
- **rotate**
- **change background**
- **detect collision, between a ball and the boundary and between balls (not shown in the demonstration)**

# Rotation in Java

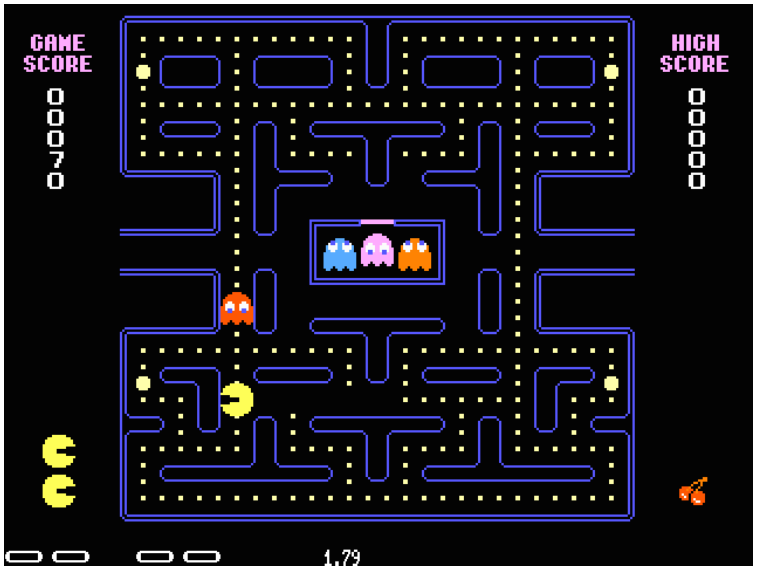
```
AffineTransform transform = new AffineTransform();
transform.rotate(theta);
// most likely, rotate around the center of the image
transform.rotate(theta, imageToShow.getWidth(null) / 2,
                        imageToShow.getHeight(null) / 2);
g.drawImage(imageToShow, transform, null);
```

# Collision Detection



- required in many games, such as
  - player hits walls in a maze
  - missile hits enemy spacecraft
  - ball bounces at the boundary of the play field
  - a Tetris piece hits a square and cannot rotate
  - a fighter's sword hits another fighter
  - a racing car hits a guardrail
  - a baseball hitter misses the ball

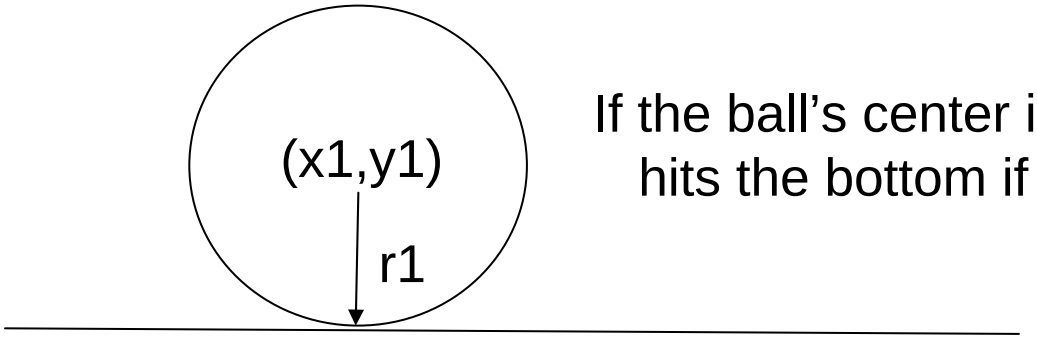

HOME
18
01:17
28
AWAY


GAME SCORE
00070
HIGH SCORE
00000


Space Invaders Game
File
High Score: 1690
Score: 0
LIVES

# Collision Detection in Lab 10

- ball bounces after hitting a wall

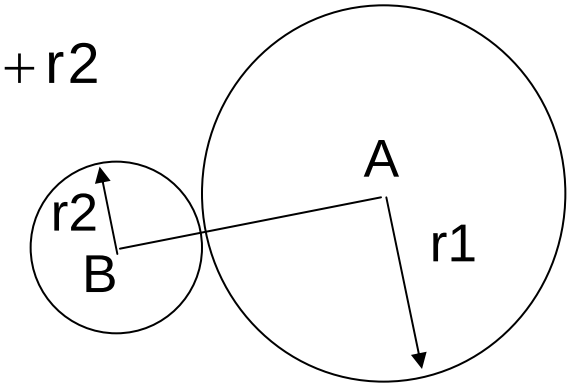



If the ball's center is (x1, y1), the ball hits the bottom if (Ymax - y1) < r1

- two balls bounces after hitting each other, if $\overline{AB} < r1 + r2$

$$\overline{AB} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} < r1 + r2$$

remember that both balls are moving



# Programming Hints

- add an attributes for the angle (theta) and the angular velocity (dt)
- in Sprite::update, add theta += dt * elapsedTime;

```
public void draw(Graphics2D g) {
   if (changeBackground == false) { bkIndex = 0; }
   g.drawImage(bgImage[bkIndex], 0, 0, null);
   AffineTransform transform = new AffineTransform();
   for (int i = 0; i < NUM_BALLS; i++) {
      Sprite sprite = balls[i];
      if (translate)  { transform.setToTranslation(sprite.getX(), sprite.getY()); }
      if (rotate)  { transform.rotate(sprite.getT(), soImage.getWidth(null) / 2,
                                      soImage.getHeight(null) / 2); }
      g.drawImage(soImage, transform, null);
     }
  }
```

# Abstract Class for Game Programs
## (Brackeen's Book Chapter 3)

```
public abstract class GameCore {
    ...
   public void gameLoop() {
      long startTime = System.currentTimeMillis();
      long currTime = startTime;
      while (isRunning) {
         long elapsedTime =
            System.currentTimeMillis() - currTime;
         currTime += elapsedTime;
    ...
   public abstract void draw(Graphics2D g);
   public void stop() {
      isRunning = false;
   }
```

# Handle Keyboard Events

- In Netbeans, an event handler is created by the programming interface.
- When writing code, use "**Listener**" for events.
- Pressing a key triggers one event: keyPressed
- Releasing a key triggers two events: keyReleased and keyTyped. The former indicates that no key is being pressed. The latter indicates which key is pressed and released.
- To know which key is pressed:

```
public void keyPressed(KeyEvent e) {
  int keyCode = e.getKeyCode();
  String txt = KeyEvent.getKeyText(keyCode);
  // more information available in Java's KeyEvent class and
  // http://java.sun.com/docs/books/tutorial/uiswing/events/keylistener.html
```

```
import java.awt.*;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import java.util.LinkedList;
import com.brackeen.javagamebook.graphics.*;
import com.brackeen.javagamebook.test.GameCore;
/** A simple keyboard test. Displays keys pressed and released to the
    screen. Useful for debugging key input, too.  */
public class KeyTest extends GameCore implements KeyListener {
    public static void main(String[] args) {
        new KeyTest().run();
    }
    private LinkedList messages = new LinkedList();
    public void init() {
        super.init();
        Window window = screen.getFullScreenWindow();
```

```
    // allow input of the TAB key and other keys normally used for focus
    // traversal, call getFocusTraversalKeys to find which they are
    window.setFocusTraversalKeysEnabled(false);
    // register this object as a key listener for the window
    window.addKeyListener(this);
    addMessage("KeyInputTest. Press Escape to exit");
}
// a method from the KeyListner interface
public void keyPressed(KeyEvent e) {
    int keyCode = e.getKeyCode();
    // exit the program
    if (keyCode == KeyEvent.VK_ESCAPE) { // each key has it own code
        stop(); // set isRunning to false
    }
    else {
        addMessage("Pressed: " +  KeyEvent.getKeyText(keyCode));
```

```
            // make sure the key isn't processed for anything else
            e.consume();
        }
    }
    // a method from the KeyListner interface
    public void keyReleased(KeyEvent e) {
        int keyCode = e.getKeyCode();
        addMessage("Released: " + KeyEvent.getKeyText(keyCode));
        // make sure the key isn't processed for anything else
        e.consume();
    }
    // a method from the KeyListener interface
    public void keyTyped(KeyEvent e) {
        // this is called after the key is released - ignore it
        // make sure the key isn't processed for anything else
        e.consume();
    }
```

```
/** Add a message to the list of messages.  */
public synchronized void addMessage(String message) {
    messages.add(message);
    if (messages.size() >= screen.getHeight() / FONT_SIZE) {
        messages.remove(0);
    }
}
/** Draw the list of messages */
public synchronized void draw(Graphics2D g) {
    Window window = screen.getFullScreenWindow();
    g.setRenderingHint(
        RenderingHints.KEY_TEXT_ANTIALIASING,
        RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
    // draw background
    g.setColor(window.getBackground());
    g.fillRect(0, 0, screen.getWidth(), screen.getHeight());
```

```
        // draw messages
        g.setColor(window.getForeground());
        int y = FONT_SIZE;
        for (int i=0; i<messages.size(); i++) {
            g.drawString((String)messages.get(i), 5, y);
            y+=FONT_SIZE;
        }
    }
}
```

# Handle Mouse Events

- four types of mouse events: button click, motion, wheel scroll, enter and leave
- create a MouseListener, MouseMotionListener, MouseWheelListener

```
import java.awt.*;
import java.awt.event.*;
import java.util.LinkedList;
import com.brackeen.javagamebook.graphics.*;
import com.brackeen.javagamebook.test.GameCore;
/** A simple mouse test. Draws a "Hello World!" message at the
    location of the cursor. Click to change to "trail mode" to draw
    several messages. Use the mouse wheel (if available) to change colors.  */
```

```
public class MouseTest extends GameCore implements KeyListener,
    MouseMotionListener, MouseListener, MouseWheelListener
{
    public static void main(String[] args) {
        new MouseTest().run();
    }
    private static final int TRAIL_SIZE = 10;
    private static final Color[] COLORS = {
        Color.white, Color.black, Color.yellow, Color.magenta
    };
    private LinkedList trailList;
    private boolean trailMode;
    private int colorIndex;
    public void init() {
        super.init();
        trailList = new LinkedList();
        Window window = screen.getFullScreenWindow();
```

```
        window.addMouseListener(this);
        window.addMouseMotionListener(this);
        window.addMouseWheelListener(this);
        window.addKeyListener(this);
    }
    public synchronized void draw(Graphics2D g) {
        int count = trailList.size();
        if (count > 1 && !trailMode) {
            count = 1;
        }
        Window window = screen.getFullScreenWindow();
        // draw background
        g.setColor(window.getBackground());
        g.fillRect(0, 0, screen.getWidth(), screen.getHeight());
        // draw instructions
        g.setRenderingHint(
                RenderingHints.KEY_TEXT_ANTIALIASING,
                RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
```

```
        g.setColor(window.getForeground());
        g.drawString("MouseTest. Press Escape to exit.", 5,  FONT_SIZE);
        // draw mouse trail
        for (int i=0; i<count; i++) {
            Point p = (Point)trailList.get(i);
            g.drawString("Hello World!", p.x, p.y);
        }
    }
    // from the MouseListener interface
    public void mousePressed(MouseEvent e) {
        trailMode = !trailMode;
    }
    // from the MouseListener interface
    public void mouseReleased(MouseEvent e) {
        // do nothing
    }
    // from the MouseListener interface
```

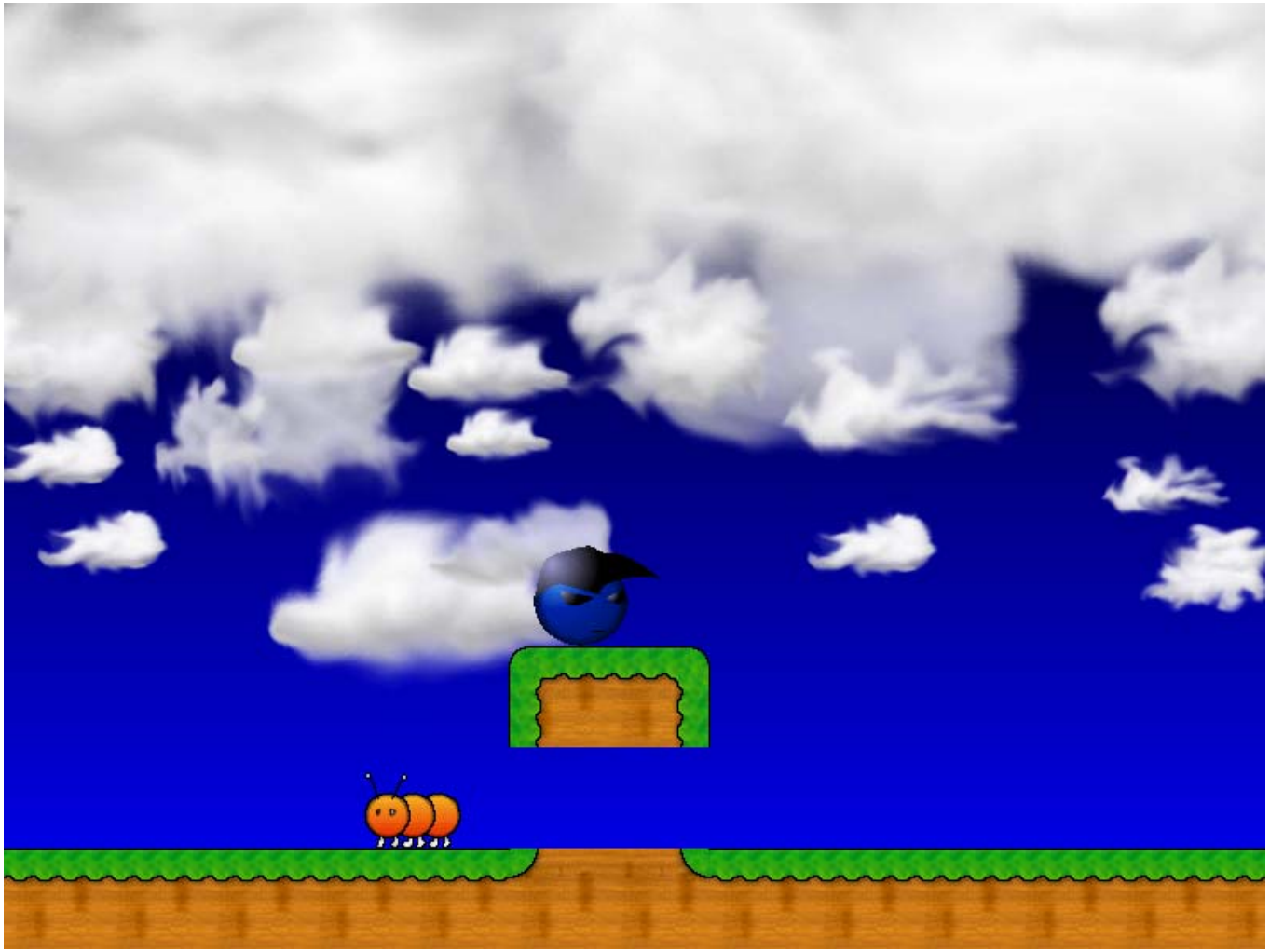
```
public void mouseClicked(MouseEvent e) {
    // called after mouse is released - ignore it
}
// from the MouseListener interface
public void mouseEntered(MouseEvent e) {
    mouseMoved(e);
}
// from the MouseListener interface
public void mouseExited(MouseEvent e) {
    mouseMoved(e);
}
// from the MouseMotionListener interface
public void mouseDragged(MouseEvent e) {
    mouseMoved(e);
}
```

```
    // from the MouseMotionListener interface
    public synchronized void mouseMoved(MouseEvent e) {
        Point p = new Point(e.getX(), e.getY());
        trailList.addFirst(p);
        while (trailList.size() > TRAIL_SIZE) {
            trailList.removeLast();
        }
    }
    // from the MouseWheelListener interface
    public void mouseWheelMoved(MouseWheelEvent e) {
        colorIndex = (colorIndex + e.getWheelRotation()) %
            COLORS.length;
        if (colorIndex < 0) {
            colorIndex+=COLORS.length;
        }
        Window window = screen.getFullScreenWindow();
        window.setForeground(COLORS[colorIndex]);
    }
```

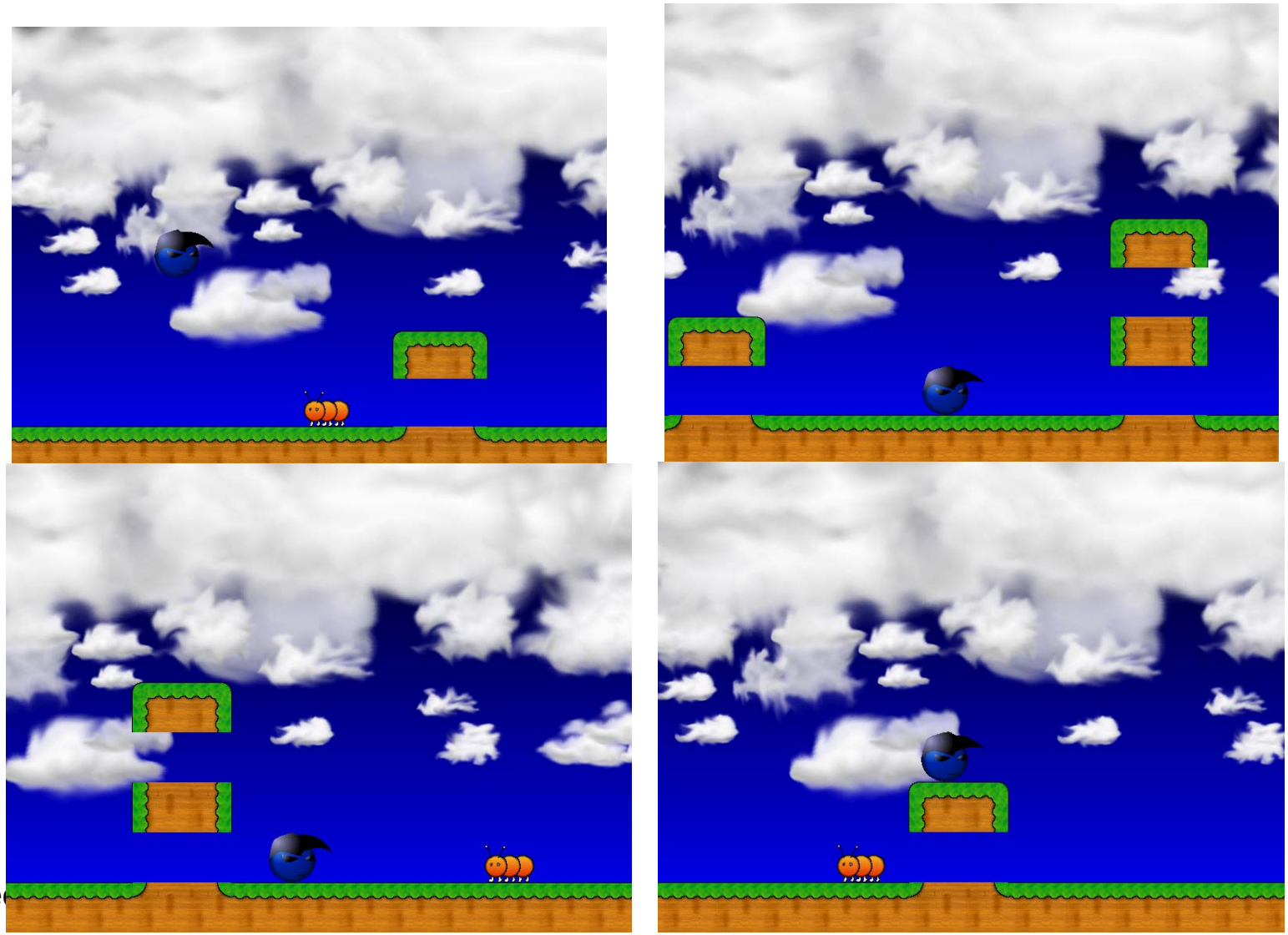
```
    // from the KeyListener interface
    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_ESCAPE) {
            // exit the program
            stop();
        }
    }
    // from the KeyListener interface
    public void keyReleased(KeyEvent e) {
        // do nothing
    }
    // from the KeyListener interface
    public void keyTyped(KeyEvent e) {
        // do nothing
    }
}
```

# Creating 2D Java Game

# Creating 2D Java Game



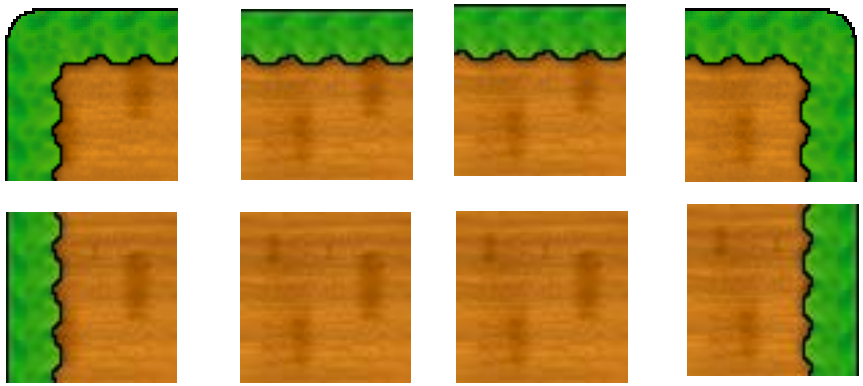
# Changing Background (Cloud)



# Changing Background

- background map: To provide a changing background without a very large actual background
- sky

- tiles



# Tile Map

- (review) Java objects are not deep copied (unless cloned); instead, they refer to the same objects and use much less memory than deep copy.

$\Rightarrow$ the titles do not need individual copies of the images

- collision detection based on the type of a tile
- complex map $\Rightarrow$ a special map editor
- simple map $\Rightarrow$ text editor sufficient
- Each object's location is relative to the map (not to the screen) so that a player can scroll left or right easily.

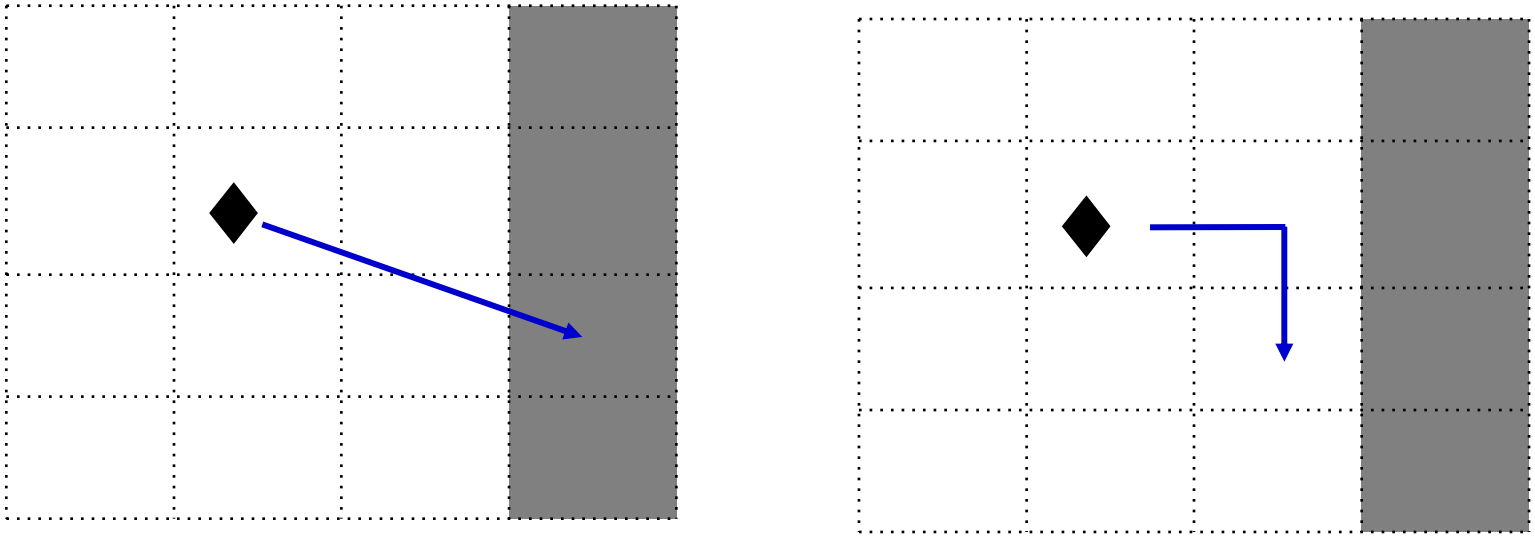
# Manage Visible Objects


- An object is visible if the location is between xoffset and xoffset + xresolution
- In a game with few moving objects, it is acceptable to draw every object. If it is outside the visible region, it cannot be seen.
- If the background moves slowly, it gives the impression of distance. This allows a smaller image for the background and saves memory.

# Tile-Based Collision Detection

- If the sprite's next location is (x,y), checking whether collision occurs is easy:

  tileMap.get(x,y) $\Rightarrow$ if a tile exists, collision



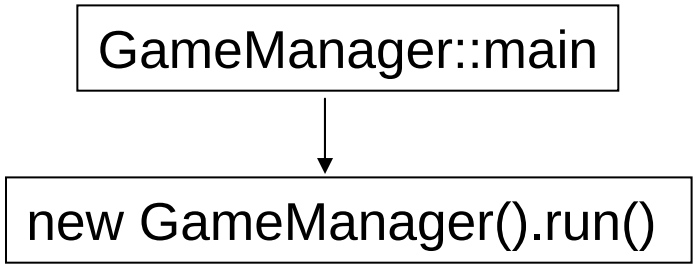
# Analyze a Complex Program

- run doxygen, already installed in ECN computers
  http://www.stack.nl/~dimitri/doxygen/
- doxygen -g ⇒ generate configuration, called Doxyfile
- change the following lines in Doxyfile
  OUTPUT_DIRECTORY = docs
  EXTRACT_ALL = YES
  EXTRACT_PRIVATE = YES
  INPUT = src/
  RECURSIVE = YES
- doxygen (again) to generate documentation

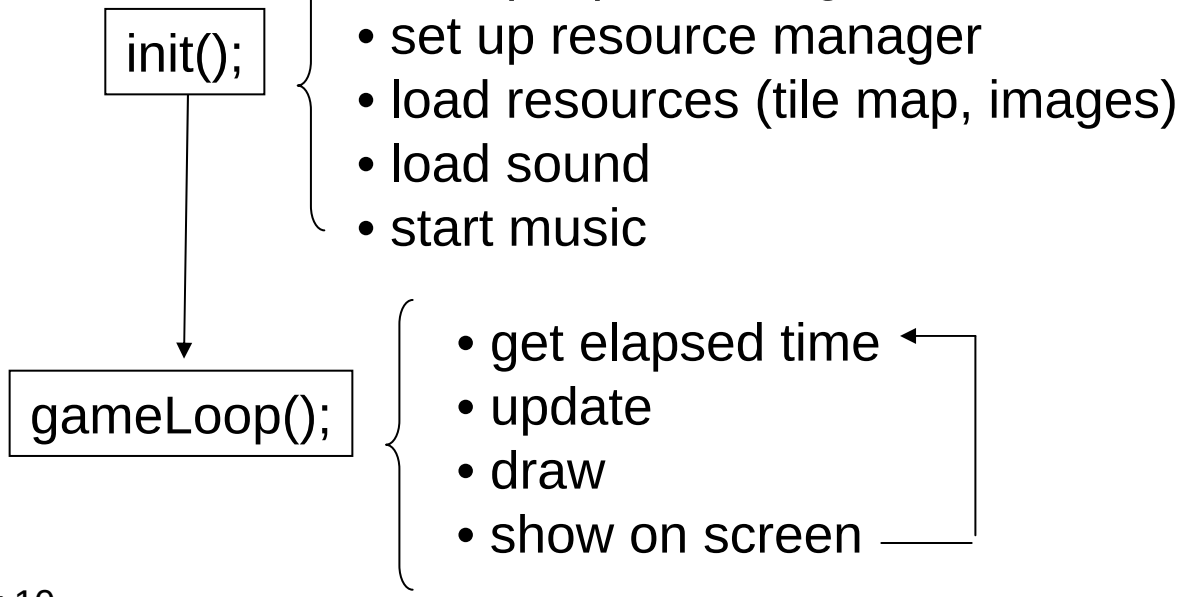
# Create and Execute the Program

- The source comes with build.xml. It is the “Makefile” for program ant (http://ant.apache.org/). In your ee462xxx account, type “ant” should create tilegame.jar.
- To execute the program, type “java -jar tilegame.jar”

# Execution Flow




calls GameCore::run because GameCore is the base class. This “run” has nothing to do with threads. It is a bad choice of the function’s name.



# Object-Oriented Programming

- create class hierarchies so that common functionalities are handled by the base classes.
- Example, Sprite - Creature - Fly
  - update is overridden in derived class, Creature
  - all of them have attributes x, y, dx, and dy
  - collision detection takes an object as the base class
- A member function is the base class can call another member function. If the second member function is overridden, the derived class' function is called. For example, GameCore::run call init().

# Objects Know What to Do

- Calling Each Object's "update": Six classes implement the "update" function. Pay special attention to Sprite::update and Creature::update.
- Each moving object keeps its location and velocity.
- The program, however, has some examples that are not "object-oriented".
  - Avoid getX and setX functions; they break encapsulation
  - When collision occurs, the object should decide what to do. The velocity should **not** be set externally.
- Do not use the name of a Java function, such as run

# Improve Java Performance

- String is immutable. Changing a String object means (1) discarding the original String object $\Rightarrow$ garbage (2) creating a new String object $\Rightarrow$ memory allocation.
- If a String object is changed frequently, use StringBuffer.
- Be careful about creating intermediate objects.
- Java String cannot have a derived class.

```
String st1 = "hello";
for (int iter = 0; iter < NUMBER_ITERATION; iter ++)
 { st1 += iter; }
StringBuffer st2 = new StringBuffer("hello");
for (int iter = 0; iter < NUMBER_ITERATION; iter ++)
{ st2.append(iter); }
```

# Compile-Time vs. Run-Time

```
String s3 = "Hello";
s3 = "Hello" + " World!";
String s4 = "Hello";
s4 = new String("Hello") + "World";
```

- Are they the same?
  - Yes, functionality
  - No, performance
- Assigning to s3 is faster because the String object can be created at compile-time.

# Tools for Java

- Profile tools:
  - java -Xprof xxx
  - Profile in netbeans
- Test coverage: EMMA