


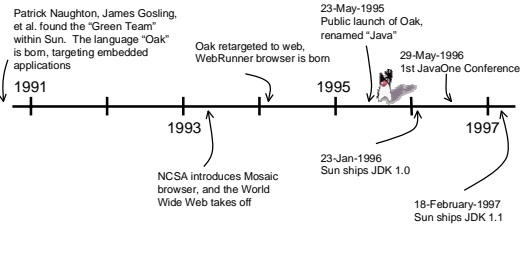
Java



“A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language.”
— Sun

Alan Borning
(stolen from Greg J. Badros)
University of Washington
CSE 341, Spring 2000

Java: A Timeline



1991: Patrick Naughton, James Gosling, et al. found the "Green Team" within Sun. The language "Oak" is born, targeting embedded applications.

1993: NCSA introduces Mosaic browser, and the World Wide Web takes off.

1995: Oak retargeted to web, WebRunner browser is born.

23-May-1995: Public launch of Oak, renamed "Java".

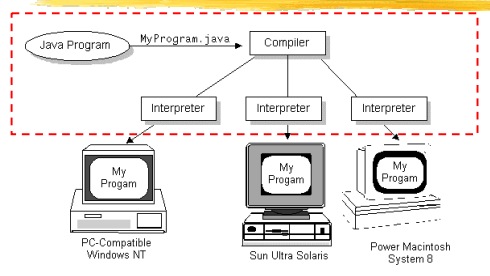
23-Jan-1996: Sun ships JDK 1.0.

29-May-1996: 1st JavaOne Conference.

18-February-1997: Sun ships JDK 1.1.

Spring 2000 2

Java: The Language



Spring 2000 3


Ref: Sun- What is Java? page

Hello World!

```


HelloWorld.java
/** Application HelloWorld
    Just output "Hello World!" */
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}

% javac HelloWorld.java
% java HelloWorld
Hello World!
    
```



Spring 2000 4

Java vs. C++



```


JAVA
/** Application HelloWorld */
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}

C++
// Application HelloWorld
#include <iostream.h>

int main(int argc, char* argv[]) {
    cout << "Hello World!" << endl;
}
    
```

Spring 2000 5

Unlike C++, Java has....



- No global functions — everything is in a class!
- Real **String** objects — not just char[]
- No pointers — everything is a reference
- No user operator overloading
- No preprocessor — cpp not needed
- unicode instead of ascii

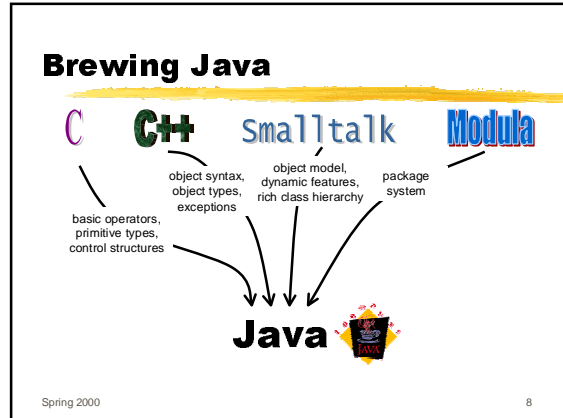
Spring 2000 6

Command Line Arguments

```

PrintArgs.java
/** Application PrintArgs
 prints the command line arguments */
public class PrintArgs {
 public static void main(String[] args) {
 for (int i = 0; i < args.length; i++)
 System.out.println(args[i]);
 }
 }
    
```

Spring 2000 7



Java vs. C++, Revisited

	<pre> Ball ball = new Ball(50, 50); PinballAnimationPane pap = new PinballAnimationPane(); pap.addObject(ball); ball.animate(); </pre>
	<pre> Ball *pball = new Ball(50,50); PinballAnimationPane *pxpap = new PinballAnimationPane(); pxpap->addObject(pball); pball->animate(); </pre>
	<pre> Ball ball(50,50); // creates ball on stack PinballAnimationPane xpap(); // creates xpap on stack xpap.addObject(ball); // calls: addObject(Ball &b); ball.animate(); </pre>

Spring 2000 9

Java's Hybrid Object Model

- Primitive types on stack
 - May be *wrapped* or *boxed* into a real object


```
Integer anInteger = new Integer(43);
```

 (useful for storing in java.util.*s collections)
 - Unboxed primitives very similar to in C++
- All object instances live in the heap (not stack)
 - all object creation is done with `new`
 - No "delete" — Java uses garbage collection, but also provides `finalize()` method

Spring 2000 10

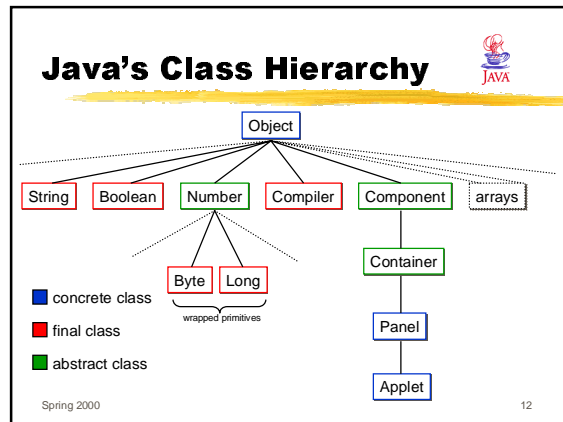
Primitive types in Java

- boolean
- char (16-bit) //unicode
- byte (8-bit signed)
- short (16-bit signed)
- int (32-bit signed)
- long (64-bit signed)
- float (32-bit signed)
- double (64-bit signed)

Integer types: int, long

Floating point types: float, double

Spring 2000 11



Java Documentation



Class java.lang.Boolean

public final class Boolean
extends Object
implements Serializable

The Boolean class wraps a value of the primitive type boolean in an object. An object of type Boolean contains a single field whose type is boolean.

In addition, this class provides many methods for converting a boolean to a String and a String to a boolean, as well as other constants and methods useful when dealing with a boolean.

Field Index

- FALSE** The Boolean object corresponding to the primitive value false.
- TRUE** The Boolean object corresponding to the primitive value true.
- TRUE** The Class object representing the primitive type boolean.

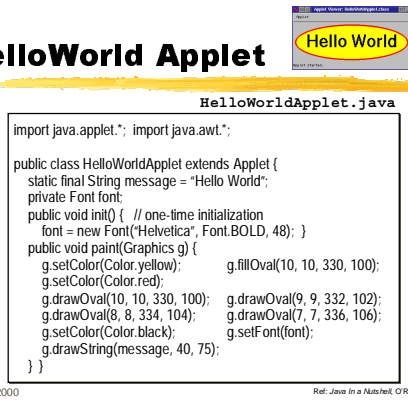
Constructor Index

- Boolean(boolean)** Allocates a Boolean object representing the value argument.
- Boolean(String)** Allocates a Boolean object representing the value true if the string argument is not null and is equal, ignoring case, to the string "true".

Method Index

Spring 2000 13

HelloWorld Applet



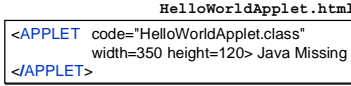
```

HelloWorldApplet.java
import java.applet.*; import java.awt.*;

public class HelloWorldApplet extends Applet {
    static final String message = "Hello World";
    private Font font;
    public void init() { // one-time initialization
        font = new Font("Helvetica", Font.BOLD, 48); }
    public void paint(Graphics g) {
        g.setColor(Color.yellow); g.fillOval(10, 10, 330, 100);
        g.setColor(Color.red);
        g.drawOval(10, 10, 330, 100); g.drawOval(9, 9, 332, 102);
        g.drawOval(8, 8, 334, 104); g.drawOval(7, 7, 336, 106);
        g.setColor(Color.black); g.setFont(font);
        g.drawString(message, 40, 75);
    }
}
    
```

Spring 2000 Ref: Java in a Nutshell, O'Reilly 14

Running the HelloWorld Applet




```

HelloWorldApplet.html
<APPLET code="HelloWorldApplet.class"
width=350 height=120> Java Missing
</APPLET>
    
```

Add "." to your \$CLASSPATH, then
% appletviewer HelloWorldApplet.html

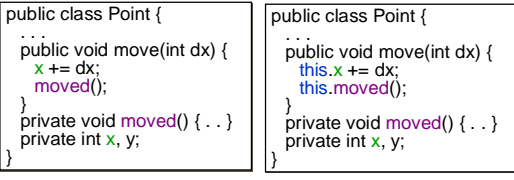
Run on the .html file



Or use a web browser on the .html file ...

Spring 2000 Ref: Java in a Nutshell, O'Reilly 15

Methods: A Closer Look



```

public class Point {
    ...
    public void move(int dx) {
        x += dx;
        moved();
    }
    private void moved() { .. }
    private int x, y;
}

public class Point {
    ...
    public void move(int dx) {
        this.x += dx;
        this.moved();
    }
    private void moved() { .. }
    private int x, y;
}
    
```

- this** is implicit on instance fields and methods
 - can be explicit if the field is hidden by a local or formal
 - analogous to self in Smalltalk
- also **super** keyword, as in Smalltalk (no C++ :: operator)
 - also used for constructor chaining with arguments

Spring 2000 16

More on Methods

- Instance methods** (no **static** keyword)
 - have implicit **this** argument
 - can use **super** keyword
 - no need to use **"->"** operator as in C++ just **.** operator since **this**, **super** are references
- static** (class) methods
 - do not have implicit **this** argument
 - cannot use the **super** keyword

Spring 2000 17

Default Arguments

- No language support—must use overloading instead

```

public class Point {
    public Point() { this(0,0); }
    public Point(int x, int y) { this.x=x; this.y=y; }
    public void move() { move(1); }
    public void move(int dx) { x += dx; }
    private int x, y;
}
    
```

special use of "this"

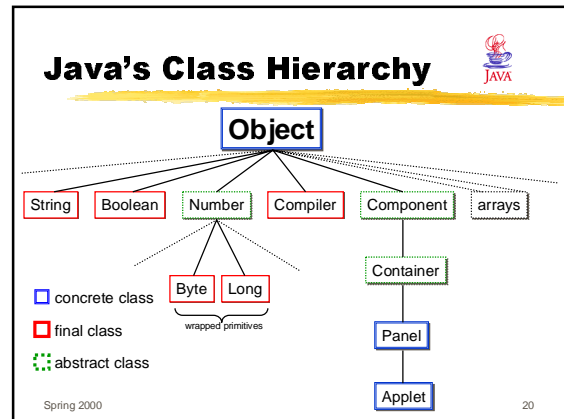
Note: two different x's and y's

Spring 2000 18

“Override” vs. “Overload”

- Override**
 - replace a superclass’s method with a specialized version
 - signatures must match
(including return type; C++ permits narrowing of return types, Java does not)
- Overload**
 - write several methods for a given class with the same name
 - language can disambiguate based on number or types of arguments

Spring 2000 19



What can an Object do for you today?

- Object `clone()`
Return a duplicate copy of `self`
- boolean `equals(Object obj)`
Defaults to `==` but can be overridden.
- String `toString()`
Return printable representation of `self`
- int `hashCode()`
Return a reasonable hash code for `self`
- Class `getClass()`
Return the class object for `self`

Spring 2000 21

More on equals()

```
public class Ball {
    public Ball(int x, int y) { this.x =x; this.y=y; }

    public boolean equals(Object b) {
        // && doesn't evaluate its second arg unless
        //necessary
        return (b instanceof Ball &&
                x==b.x && y==b.y)
    }
}
```

Spring 2000 22

Objects and Identities

```
Ball ball = new Ball(50,50);
Ball sameBall = ball;
```

Test object identity:
ball == sameBall ⇒ true

Test object value's equality:
ball.equals(sameBall) ⇒ true

Spring 2000 23

Cloning Objects

```
Ball ball = new Ball(50,50);
Ball sameBall = ball;


Ball anotherBall = (Ball) ball.clone();
```

Test object identity:
ball == anotherBall ⇒ false

Test object value's equality:
ball.equals(anotherBall) ⇒ true

Spring 2000 24

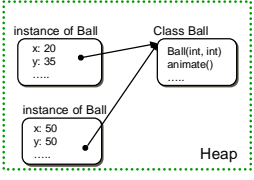
Changing a Ball



```
ball.setPosition(20,35);
sameBall;
anotherBall;
```


Test object identity:
 ball == sameBall ⇒ ???
 ball == anotherBall ⇒ ???

Test object value's equality:
 ball.equals(sameBall) ⇒ ???
 ball.equals(anotherBall) ⇒ ???



Spring 2000 25

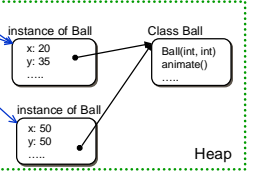
Inequality in Balls!



```
ball.setPosition(20,35);
sameBall;
anotherBall;
```

Test object identity:
 ball == sameBall ⇒ true
 ball == anotherBall ⇒ false

Test object value's equality:
 ball.equals(sameBall) ⇒ true
 ball.equals(anotherBall) ⇒ false



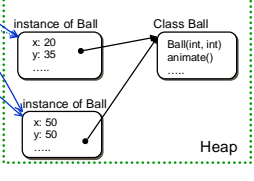
Spring 2000 26

Assignment just changes the pointer

```
ball;
sameBall = anotherBall;
anotherBall;
```


Test object identity:
 ball == sameBall ⇒ false
 ball == anotherBall ⇒ false

Test object value's equality:
 ball.equals(sameBall) ⇒ false
 ball.equals(anotherBall) ⇒ false



Spring 2000 27

Java variables hold...



- primitive


```
boolean foo; // boolean, not bool as in C++
char aChar = 'a'; // 16 bit char (unicode)
```
- Object reference (may be null)



```
ColoredBall cball = new Ball();
Ball ball = cball;
```
- Array reference


```
int[] intArray = { 1, 2, 3, 4, 5, };
String[] strArray = { "Hello", "World", };
// same as
String[] strArray = new String[2];
strArray[0] = new String("Hello");
strArray[1] = new String("World");
```

String literals actually invoke constructor e.g., new String("World")

Spring 2000 28


Arrays



Similar to:


- Java arrays are 1st-class Objects
- 0-indexed
- Bounds checking performed
- Store/Retrieve using [] operator
 strArray[0] = strArray[1];
- Have implicit length field
 strArray.length ⇒ 2

A field, not a method!



Spring 2000 29

2-d and 3-d Arrays



- No special language support for 2-d arrays -- just make an array of arrays

```
public class myArray {
    public static void main (String[] args) {
        double [] [] mat = {{1, 2., 3., 4.}, {5., 6., 7., 8.},
                            {9., 10., 11., 12.}, {13., 14., 15., 16.}};
        for (int y = 0; y < mat.length; y++) {
            for (int x = 0; x < mat[y].length; x++)
                System.out.print(mat[y][x] + " ");
            System.out.println();
        }
    }
}
```

Spring 2000 30

Strings

- The **String** class provides **read-only** strings and supports operations on them
- A **String** can be created **implicitly** either by using a quoted string (e.g. "HUB food") or by the concatenation of two **String** objects, using the **+** operator.

Spring 2000 31

Strings are Immutable

Since you **cannot modify existing strings**, there are methods to create **new strings** from existing ones.

- public **String** substring(int beginIndex, int endIndex)
- public **String** replace(char oldChar, char newChar)
- public **String** concat(**String** str)
- public **String** toLowerCase()
- public **String** toUpperCase()
- public **String** trim()

Spring 2000 32

Identifiers

- Everything has a globally-unique name

```

Java.lang.String
Java.util.Hashtable
Java.applet.Applet
EDU.Washington.grad.gjb.cassowary.Variable.toString()
    
```

Package name
Class name
Method name

- Pretty wordy, so...

Spring 2000 33

import statement

- Two forms:
 - import java.util.Hashtable;
 - Just make the Hashtable class available from package java.util
 - import EDU.Washington.grad.gjb.cassowary.*;
 - Make all classes from package available on demand
- Always an implicit "import java.lang.*"
- Permits using simple (short) names
 - Not like C++'s "#include"
 - More like C++'s "using namespace"

Spring 2000 34

How Java Finds a Class...

- Package names mirror the directory structure
- package statement informs the compiler

```

./EDU/Washington/grad/gjb/cassowary/Variable.java
package EDU.Washington.grad.gjb.cassowary;

public class Variable extends AbstractVariable {
    ...
}

class Helper { ... }
    
```

Spring 2000 35

Compilation of Source File

```

% ls
Variable.java
% javac Variable.java
% ls
Variable.java
Variable.class
Helper.class
    
```

One java source file may create multiple .class files containing the byte-compiled code

Spring 2000 36

Class Access Protection

```

package EDU.Washington.grad.gjb.cassowary;
public class Variable extends AbstractVariable {
    ...
}
class Helper { ... }
    
```

- Only one **public** class per file
- No specifier \Rightarrow package protection
 visible to all classes in the package
 no "package" keyword — remember it is a statement

Spring 2000 37

Private: most restrictive access modifier

```

public class Point {
    private int x, y;
    void setXY(int x, int y) {
        this.x = x; this.y = y;
    }
    protected void move(int x, int y) {
        setXY(this.x+x, this.y+y);
    }
    public int getX() { return x; }
    public int getY() { return y; }
}
    
```

Y	N	N	N	private
Y	Y	N	N	package
Y	Y	Y	N	protected
Y	Y	Y	Y	public

same class
 class in same package
 subclass in different package
 non-subclass, different package

Spring 2000 Ref: Java in a Nutshell, O'Reilly 38

Java Accessibility vs. C++

- No "friend" keyword
- Every field or method has an access specifier (no "public:" sections)
- Default is package-visibility which has no associated keyword (not private)

Spring 2000 39

No Need for Forward Declarations

```

public class Point {
    private PointColor c;
    // setXY(int,int) used below before its definition in the source
    protected void move(int x, int y) { setXY(this.x+x, this.y+y); }
    void setXY(int x, int y) { this.x = x; this.y = y; }
    private int x, y;
} // no trailing semicolon (C++ requires one)

// PointColor already used above before this definition
class PointColor {
    byte red, green, blue;
}
    
```

■ Definition
■ Use

Spring 2000 40

Final Fields

```

public final class Circle {
    private final double MY_PI = 3.1415;
    public double area() { return MY_PI * r*r; }
}
    
```

- final** fields correspond to C++'s "const"
- final** fields cannot be changed once initialized
- cannot use **final** in function signatures
 (less flexible than C++ — **const** is an unused reserved word in Java)

Spring 2000 41

Ball and CBall Example

```

BallExample/Ball.java
package BallExample;
public class Ball implements Bounceable {
    private int x, y;
    public Ball(int x, int y) {
        this.x=x; this.y = y;
    }
    public void Bounce() {
        System.err.println("Ball bounces");
    }
    static public void ClassFn() {
        System.err.println("Ball.ClassFn()");
    }
}

BallExample/CBall.java
package BallExample;
public class CBall extends Ball {
    private int colorSelector;
    public CBall(int x, int y) {
        super(x,y); // chain constructors
        colorSelector = 0; // for black
    }
    public void Bounce() {
        System.err.println("CBall bounces");
    }
    static public void ClassFn() {
        System.err.println("CBall.ClassFn()");
    }
}
    
```

Spring 2000 42

Inheritance Mechanisms

- **extends superclass**
 - similar to “: public” in C++
 - for expressing an “is-a” relation
- **implements superinterface**
 - similar in use to C++’s multiple inheritance
 - for expressing an “is-capable-of” or “knows-how-to” relation

Spring 2000 43

Java Interfaces

```
public interface Bounceable {
    public void Bounce();
    private void BounceNow(); // error
}
```

```
public interface BounceDropable
    extends Bounceable {
    public void Drop();
}
```

- Interfaces can only specify public methods
- Similar to protocols in Smalltalk
- May be used as a type for a variable
- Can specify sub-interfaces and can extend multiple interfaces at a time

Spring 2000 44

Bounceable Interface


```
package BallExample;
public interface Bounceable {
    public void Bounce();
}
```

```
package BallExample;
public class BallTest {
    public static void main(String[] args) {
        Ball b1 = new Ball(10,10);
        Ball b2 = new CBall(20,20);
        Bounceable b3 = new Ball(30,30);
        Bounceable b4 = new CBall(40,40);

        b1.Bounce();    b2.Bounce();
        b3.Bounce();    b4.Bounce();

        b1.ClassFn();   b2.ClassFn();
        b3.ClassFn();   b4.ClassFn();

        CBall cb1 = (CBall) b1;
        CBall cb2 = (CBall) b2;
        cb2.ClassFn();
    } // end class
```

Errors? 

Output?

Spring 2000 45

Ball Example Output and Errors

```
% java BallExample.BallTest
Ball bounces
CBall bounces
Ball bounces
CBall bounces
Ball.ClassFn()
Ball.ClassFn()
CBall.ClassFn()
CBall.ClassFn()
```

```
package BallExample;
public class BallTest {
    public static void main(String[] args) {
        Ball b1 = new Ball(10,10);
        Ball b2 = new CBall(20,20);
        Bounceable b3 = new Ball(30,30);
        Bounceable b4 = new CBall(40,40);

        b1.Bounce();    b2.Bounce();
        b3.Bounce();    b4.Bounce();

        b1.ClassFn();   b2.ClassFn();
        b3.ClassFn();   b4.ClassFn();

        // compile time errors
        // b3.ClassFn();    b4.ClassFn();

        // CBall cb1 = (CBall) b1;    ClassCastException
        CBall cb2 = (CBall) b2; // ok
        cb2.ClassFn();
    } // end class
```

Spring 2000

Types vs. Classes C++

- **Types**
 - variables have types
 - used for checking validity of method invocations
 - may be an **interface**
- **Classes**
 - objects (i.e. instances) have classes
 - used for dynamic dispatch (binding of non-static function call)
 - Each class has a corresponding type — that hierarchy of types mirrors the class hierarchy

Spring 2000 47

Multiple Inheritance in Java

```
public class TopLevelWindow extends Window
    implements Drawable, Cloneable, Streamable
{ ... }
```

Spring 2000 48

Abstract Methods and Abstract Classes

```
// Note abstract keyword is used for the class, too
public abstract class Shape {
    public abstract void rotate(int); // no definition
    public abstract double area(); // no definition
}
```

- abstract methods correspond to C++'s "pure virtual functions" (But C++ uses "=0" syntax, and permits an implementation)
- abstract methods must be overridden in concrete subclasses
- Only abstract classes can have abstract methods (C++ infers abstract classes, Java requires you mark the class explicitly)

Spring 2000 49

Final Methods

```
public class Circle {
    ....
    public final double area() { return Math.PI * r*r; }
    double r; // radius
}
```

- final methods cannot be overridden
- final methods may be inlined (no "inline" keyword)
- similar to non-virtual member functions in C++ (but those can be overridden, they just do not dispatch dynamically)

Spring 2000 50

Final Classes

```
public final class Circle {
    ....
    public double area() { return Math.PI * r*r; }
    double r; // radius
}
```

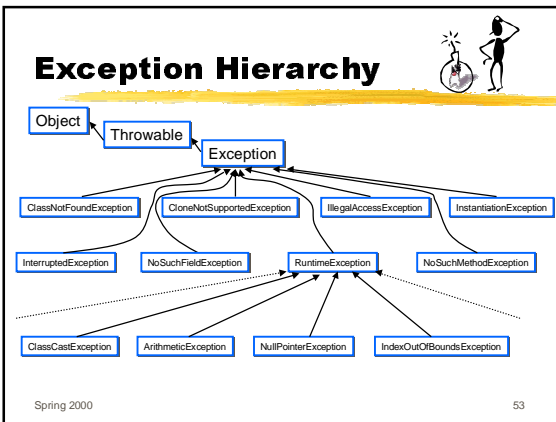
- final classes cannot be subclassed — they are leaves in the class hierarchy
- methods in final classes are implicitly final
- provides compiler with optimization opportunities

Spring 2000 51

try { throw } and catch, finally (exceptions)

```
class ExceptionExample {
    static public void main(String args[]) {
        try {
            // allocate some resource (besides memory)
            doSomething();
            if (!FThingsAreOkay()) {
                throw new RuntimeException("Things not ok");
            }
            doSomethingElse();
        } catch (RuntimeException e) {
            System.err.println("Runtime Exception: " + e);
        } catch (Exception e) {
            // similar to "catch (..)" in C++
            System.err.println("Exception: " + e);
        } finally {
            // finally is not in C++
            // cleanup resource
        }
    }
}
```

Spring 2000 52



Threads

```
public class Pendulum extends Applet implements Runnable {
    private Thread myThread;
    public void start() {
        if (myThread == null) {
            myThread = new Thread(this, "Pendulum");
            myThread.start();
        }
    }
    public void run() {
        while (myThread != null) {
            try { myThread.sleep(100); }
            catch (InterruptedException e) { /* do nothing */ }
            myRepaint();
        }
    }
    public void stop() { myThread.stop(); myThread = null; }
}
```

set thread's target to this Pendulum class, and use its run() method

Ref: Boone's Java Essentials for C and C++ Programmers

Spring 2000 54

Summary:
What Java Left Out from C++

- No stack objects, only heap objects
- No destructors, only `finalize()` method
- No pointers, everything is a reference
- No delete, garbage collector instead
- No const, only `final` (methods, fields, classes)
- No templates, no preprocessor
- No operator overloading
- No multiple inheritance of classes
- No enumerations or typedefs

Spring 2000

55

Summary:
What Java Put In (vs. C++)

- Garbage collector
- `Object`-rooted, rich class hierarchy
- `Strings`, first-class arrays with bounds checking
- Package system with `import`
- `interface`, `implements`, `extends`, `abstract`
- `finally` blocks, static/instance initializers
- Secure and portable JavaVM, threads
- Dynamic reflection capabilities, inner classes
- JavaDoc system

Spring 2000

56