“Systems” Part of the Class

• What is the “system”? 
  - Loosely defined as anything that’s not your application
• Why should you care? 
  - Learn more about the pieces that constitute a large part of your 
    daily computing life 
  - The boundaries between the different pieces are becoming 
    increasingly fussy in this age, so an “application” can have 
    elements of “the system” built in

Outline

• Introduction
  - History
  - Java vs. C
  - How to learn
• The basics
• Object-oriented niceties
• Intro to applets
• Conclusions

Roadmap

• Java
  - Superficially, a continuation of the programming part
  - But, there is a profound connection between Java and OS
• Operating systems
  - The missing link between hardware and applications
• Networking
History (cont.)

- Joy and Gosling joined forces, FirstPerson, Inc. (1992)
  - Targeting consumer electronics: PDAs, appliances, phones, all with cheap infra-red kind of networks
- Need a language that’s small, robust, safe, secure, wired
  - Started working on C++
  - Soon gave up hope, decided to start from scratch
- Again, a little ahead of its time
  - PDAs died with the demise of Apple Newton
  - Switched to interactive TV (ITV)
  - The resulting language was called “Oak”
  - Then ITV died too
- The net exploded in 1993
  - Oak became Java!

Java vs. C

- Comparison inevitable, but...
- “Java is best taught to people not contaminated by C”
  - Important to “think Java”, instead of “translating C to Java”
- Similarities between C and Java are skin-deep
  - Syntactic sugar to make it easy to swallow
  - Terseness is good
  - Underlying philosophies are like day and night
- Theme of this class: levels of abstraction
  - C exposes the raw machine
  - Java virtualizes the machine

History

- Bill Joy and Sun
  - BSD god at Berkeley
  - Founding of Sun (early 80s)
  - “The network is the computer” (a little ahead of its time)
  - Missed the boat on PC revolution
  - Sun Aspen Smallworks (1990)
- James Gosling
  - Early fame as the author of “Gosling Emacs” (killed by GNU)
  - Then onto Sun’s “NeWS” window system (killed by X)
  - Lesson 1: keeping things proprietary is kiss of death
  - Lesson 2: power of integrating three things:
    + an expressive language
    + network-awareness, and
    + a GUI (graphical user interface)

History (cont.)

- Many success stories in CS
  - Very much like what we said about Unix
  - Not a technological breakthrough
  - All of the features of Java were present in earlier research systems
  - The “genius” lies in the good taste of assembling a small and elegant set of powerful primitives that fit together well and tossing everything else
- Luck helps a lot too
How to Learn

- The best language to learn on-line, which is the best way to learn Java!
  - http://www.javasoft.com
  - http://java.sun.com/j2se/1.3/docs/api/index.html
- Start with existing code, read code, read docs
- Experiment by making small changes and adding functionality progressively
- My personal opinion: learning a second programming language in a class is a waste of time :-)
- So, it’s really just a highlight

Java vs. C (cont.)

- Bad things you can do in C that you can’t do in Java
  - Shoot yourself in the foot (safety)
  - Others shoot you in the foot (security)
  - Ignoring wounds (error handling)
- Dangerous things you have to do in C that you don’t in Java
  - Handling ammo (memory management: malloc/free)
- Good things that you can do in C but you don’t; Java makes you
  - Good practices (objected-oriented methodology)
- Good things that you can’t do in C but you can now
  - Kills with a single bullet (portability)
- An interesting lesson in abstraction (and politics?): making things better by “taking away” power
- [We will revisit these differences after we learn more about Java]

Your First Java Program

```java
moca:tmp% cat > hello.java

import java.util.Scanner;

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

moca:tmp% javac hello.java

moca:tmp% ls hello.*
hello.class hello.java

moca:tmp% java hello
Hello World!
```

- Source file: “hello.java”
- Java compiler: `javac`
- Byte code: “hello.class”
- Java interpreter: `java`
- Can install JDK on any machine, including your PC
- Other tools in JDK: `jdb`, `javap`
**Classes, Methods, and Objects**

```
public class MyStack {
    Object[] items;
    int n;

    public MyStack() {
        items = new Object[1000];
        n = 0;
    }

    public void push(Object item) {
        items[n++] = item;
    }

    public Object pop() {
        return items[--n];
    }

    public boolean empty() {
        return n == 0;
    }
}
```

import MyStack;

class StackTest {
    public static void main(String[] args) {
        MyStack s = new MyStack();
        s.push("first");
        s.push("second");
        s.push("third");
        while (!s.isEmpty())
            System.out.println(s.pop());
    }
}

MyStack.java
StackTest.java

* (Don’t need to understand everything in this code, yet)
  * A program is a sequence of classes (no .h files!)
  * A class is like a struct, one difference: methods: operations that act on the data
    that makes up the class
  * A method is like a function. (Note how they are invoked.)
  * An object to a class in Java is like a variable to a type in C

**Arrays (still same example)**

```
public class MyStack {
    Object[] items;
    int n;

    public MyStack() {
        items = new Object[1000];
        n = 0;
    }

    public void push(Object item) {
        items[n++] = item;
    }

    public Object pop() {
        return items[--n];
    }

    public boolean empty() {
        return n == 0;
    }
}
```

Arrays are first class citizen of Java.
  * No other back-doors of accessing them, for example, no pointer arithmetic
  * Array reference bounds are checked at run time
  * No seg faults possible, tremendous help in reducing headaches
  * Also important implications for safety, security, and encapsulation

**Compiling vs. Interpreting**

- Interpreter: a level of abstraction: the “virtual machine”
  * The advantage of interpreting is beyond portability
  * A convenient place to exercise all sorts of control
  * Disadvantage: slower

**More Thoughts/Details on This Example**

```
public class MyStack {
    Object[] items;
    int n;

    public MyStack() {
        items = new Object[1000];
        n = 0;
    }

    public void push(Object item) {
        items[n++] = item;
    }

    public Object pop() {
        return items[--n];
    }

    public boolean empty() {
        return n == 0;
    }
}
```

- Other than the primitives such as int, char, boolean, all variables are objects
- Concepts of object declaration, allocation, and a constructor
- How to design a Java program: think objects!
  * What objects do I break the problem into?
  * What operations do they allow?
  * How do I implement them using even smaller objects?
Java Libraries (Packages)

• Huge number of pre-written libraries
• Always check before you reinvent something of your own
• Watch out for version differences
  - http://java.sun.com/j2se/1.3/docs/api/index.html
  - Reading these docs is a major part of learning/programming Java
  - Get a big picture of what they are but read details on-demand
  - “java.util” library has a lot of useful data structure stuff: linked list, stacks, ...
• On the next slide, I will give a third implementation of the stack using a library class: Vector is an array that doesn’t require you to pre-specify a size and doesn’t fill up!

Points and Linked List

Class MyNode {
  Object item;
  MyNode next;
  MyNode(Object item, MyNode next) {
    this.item = item;
    this.next = next;
  }
}

public class MyStack {
  MyNode list = null;
  public MyStack() {} 
  public void push(Object item) {
    list = new MyNode(item, list);
  }
  public Object pop() {
    Object obj = list.item;
    list = list.next;
    return obj;
  }
  public boolean empty() {
    return list == null;
  }
}

MyStack.java

• Officially no pointers anywhere, behind the scene, each object is a pointer, called a reference, special null reference part of language
• No pointer arithmetic, no *, no ->, no free(), no pointer bugs, no pain
• Reimplement stack using a linked list
  - push() code tricky: it allocates a new node, made by calling the constructor, which puts the old list head into the next field of the new node.

Example Use of Library

import java.util.*;
public class MyStack {
  Vector items;
  public MyStack() {
    items = new Vector();
  }
  public void push(Object item) {
    items.addElement(item);
  }
  public Object pop() {
    int end = items.size()-1;
    Object obj = items.elementAt(end);
    items.removeElementAt(end);
    return obj;
  }
  public boolean empty() {
    return items.isEmpty();
  }
}

MyStack.java

Sort of like #include

Vector is a class implemented by the java.util library, called a package

All of these are operations implemented by the package. You find out about them by reading the documentation, which you can download as a whole or read online.

Outline

• Introduction
• The basics
• Object-oriented niceties
  - Inheritance
  - Encapsulation
  - Code reuse
  - Multiple implementations
• Intro to applets
• Conclusions
Encapsulation and Access Control

```java
public class MyStack {
    protected Object[] items;
    protected int n;
    public MyStack() {
        items = new Object[1000];
        n = 0;
    }
    public void push(Object item) {
        items[n++] = item;
    }
    public Object pop() {
        return items[--n];
    }
    public boolean empty() {
        return n == 0;
    }
}
```

- User of this class sees only what he’s allowed to see
- Three key words:
  - `private`: accessible only by this class
  - `protected`: subclasses can see it too
  - `public`: accessible to all
- (additional deals for “packages”, read about them on-line if you care)

Inheritance

```java
public class MyImprovedStack extends MyStack {
    public Object pop() {
        if (n <= 0)
            return null;
        return items[--n];
    }
    public Object peek() {
        if (n <= 0)
            return null;
        return items[n-1];
    }
}
```

- MyImprovedStack is a subclass of MyStack
- This example: adding functionality
- Another example use: “specialization” -- a student class inherits from a person class

Multiple Implementations

```java
import MyStack;
import MyArrStack;
import MyListStack;
class StackTest {
    public static void main(String[] args) {
        MyStack s;
        s = new MyArrStack();
        s.push("first"); s.push("second");
        while (!s.empty()) System.out.println(s.pop());
        s = new MyListStack();
        s.push("first"); s.push("second");
        while (!s.empty()) System.out.println(s.pop());
    }
}
```

- As long as a common interface is agreed upon
- We can pick and choose different implementations
- How’s this done? Next slide...

Code Reuse

```java
import MyStack;
class StackTest {
    public static void main(String[] args) {
        MyStack s1 = new MyStack();
        s1.push("first"); s1.push("second");
        while (!s1.empty()) System.out.println(s1.pop());
        MyStack s2 = new MyStack();
        s2.push(new Integer(1)); s2.push(new Integer(2));
        while (!s2.empty()) System.out.println(s2.pop());
    }
}
```

- This example: no need to write different codes for stack of Strings and stack of Integers

```java
import MyStack;
```
Java vs. C (Revisit)

- Bad things you **can** do in C that you **can’t** do in Java
  - Shoot yourself in the foot (safety)
  - Others shoot you in the foot (security)
  - Ignoring wounds (error handling)
- Dangerous things you **have to** do in C that you **don’t** in Java
  - Handling ammo (memory management: malloc/free)
- Good things that you **can** do in C but you don’t; Java **makes** you
  - Good hunting practices (objected-oriented methodology)
- Good things that you **can’t** do in C but you **can** now
  - Kills with a single bullet (portability)

Abstract Classes

```java
public abstract class MyStack {
    public abstract void push(Object item);
    public abstract Object pop();
    public abstract boolean empty();
}
```

MyStack.java

```java
import MyStack;
public class MyArrStack extends MyStack {
    ...
}
```

MyArrStack.java

```java
import MyStack;
public class MyListStack extends MyStack {
    ...
}
```

MyListStack.java

- Abstract classes specify interfaces, no implementation
- Implementations inherit abstract classes and fill in implementation details

Outline

- Introduction
- The basics
- Object-oriented niceties
- Intro to applets
- Conclusions

What We Have Learned

- These are highlights, by no means complete
- Best way of learning
  - Study the tutorial online
  - Read and experiment with existing code
  - Read docs
- **I don’t expect people to memorize or be able to reproduce syntactic details**
- **I do** expect people to be able to **read** and **understand** given code and concepts discussed
**Mini-Outline**

- **Your first applet and more tools of trade**
- **Life cycle of an applet, “funny” part**
  - You have to write a whole bunch of methods you don’t call
  - You call a whole bunch of methods that you didn’t write
- **Simple drawing and events**

**Applets: Beyond Animated Clowns**

- What can you do when you can slurp code over the net?
- **Extensibility**
  - Bill Joy: “No more protocols; just code!”
  - No need for hard wired network protocols
  - No need for hard wired information content protocols
- A brave new world
  - New way of structuring applications (local or distributed)
  - New way of structuring operating systems (local or distributed)
- Today is only an introduction to the bare basics
  - Encourage interested people to explore on their own
  - It’s fun and there’s nothing hard

**Life Cycle of an Applet**

```java
import java.applet.Applet;
import java.awt.Graphics;
public class Simple extends Applet {
    StringBuffer buffer;

    public void init() {
        buffer = new StringBuffer();
        addItem("initializing... ");
    }

    public void start() {
        addItem("starting...");
    }

    public void stop() {
        addItem("stopping...");
    }

    public void destroy() {
        addItem("preparing for unloading...");
        System.out.println(newWord);
        repaint();
    }

    public void paint(Graphics g) {
        g.drawString(buffer.toString(), 5, 15);
    }
}
```

- `init()`: browser calls it when applet first loaded
- `start()`: start execution (eg. after becoming visible)
- `stop()`: stop execution (eg. after switching to different page)
- `destroy()`: clean up after final exit
- `paint()`: browser tells it’s time to redraw

**Your First Java Applet**

```java
import java.applet.Applet;
import java.awt.Graphics;
public class Hello extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello world!", 125, 95);
    }
}
```

- To try it
  - Compile: `javac Hello.java`
  - Test: `appletviewer hello.html`
  - Or: put all these files in a publicly accessible directory (such as ~/public_html) and view using `netscape`
- What happens
  - .html and .class files are slurped over the net
  - The browser has a virtual machine (interpreter) in it
  - It checks for security violations and runs it if ok.
Example (cont.) -- Drawing

```java
public void paint(Graphics g) {
    // draw a black border and a white background
    g.setColor(Color.white);
    g.fillRect(0, 0, getSize().width - 1,
               getSize().height - 1);
    g.setColor(Color.black);
    g.drawRect(0, 0, getSize().width - 1,
               getSize().height - 1);

    // draw the spot
    g.setColor(Color.red);
    if (spot != null) {
        g.fillOval(spot.x - RADIUS,
                    spot.y - RADIUS,
                    RADIUS * 2, RADIUS * 2);
    }
}
```

Example (cont.) -- Event Handling

```java
public class ClickMe extends Applet
    implements MouseListener {
    private Spot spot = null;
    private static final int RADIUS = 7;

    public Spot(int size) {
        this.size = size;
        this.x = -1;
        this.y = -1;
    }
}
```

A Slightly Larger Example

```java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

class Spot {
    public int size;
    public int x, y;
    public Spot(int size) {
        this.size = size;
        this.x = -1;
        this.y = -1;
    }
}
```

Outline

- Introduction
- The basics
- Object-oriented niceties
- Intro to applets
- Conclusions

A helper class for the dot

```java
public class ClickMe extends Applet
    implements MouseListener {
    private Spot spot = null;
    private static final int RADIUS = 7;

    public Spot(int size) {
        this.size = size;
        this.x = -1;
        this.y = -1;
    }
}
```

A constant that can’t be changed

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

class Spot {
    public int size;
    public int x, y;
    public Spot(int size) {
        this.size = size;
        this.x = -1;
        this.y = -1;
    }
}
```

Outlining the event handling

```java
public class ClickMe extends Applet
    implements MouseListener {
    private Spot spot = null;
    private static final int RADIUS = 7;

    public Spot(int size) {
        this.size = size;
        this.x = -1;
        this.y = -1;
    }
}
```
### The “Truth” (cont.)

**“Productive”**
- Much less debugging headaches: no pointer probs, exceptions
- Stealing has never been easier: the net, portability, reusability
- Excellent documentation
- Large and growing body of libraries to help: utilities, media, GUI, networking, threads, databases, cryptography...
- Flip side: versions, large libraries

**“Slow”**
- Interpreted, too many tiny objects and methods
- Flip side: just-in-time compiling can make things almost as fast as native code

**“Hype”**
- Important for momentum which translates into community expertise and support, applications, tools, and libraries
- Flip side: hasty decision-making to feed the frenzy

**Only game in town?**
- Unprecedented roles for scripting languages on the net

### The “Truth”

**“KISS”**
- Large number of complicated features of C++ gone
- The language is incredibly small
- Flip side: huge number of libraries and you can’t be a serious Java programmer without knowing a lot about them

**“Modern”**
- Garbage collection, strongly typed, exceptions, support for multi-threading and networking
- Flip side: ideas have been around in the research community for ages: Modula-3, Smalltalk, Lisp, C++, Object C

**“Secure”**
- A nice three-tier protection system: verifier, class loader, and security manager.
- Can reason about it formally
- Flip side: bugs

**On only game in town?**
- Unprecedented roles for scripting languages on the net