

< Java

- The Java Language
- Object-oriented Programming
- Objects and Class
- Attributes and Behavior
- Organizing Classes and Class Behavior
-

< The Java Language

- Started in 1991 at Sun Microsystems, originally named “Oak”
- Named ‘Java’ in 1995
- Released in 1995
- Platform-independent, object-oriented
-

< Procedural programming

- Programs were procedure-oriented
- Involved with procedure rather than the data
 - depended more on having the steps defined

< Object-oriented Programming

- Program organized around the data
- “data controlling access to code”
- Define the data and the routines that act on the data

< Objects and Class

- Object – self-contained element of computer program that represents a related group of features and designed to complete specific tasks
- Class – template to create an object.

< Computer Data Storage

- Computer use 0s and 1s (binary) to store data
- Humans use base 10
- 0s and 1s are stored in bits (8 bits make a byte)
- A bit can only be on (1) or off (0)
- Values are evaluated by multiplying 2 to a power

< Data types

- To access computer data storage (location) requires that storage to be named – “Smith”
- “Datatype” informs us as to:
 - What type of information is going into that storage location and what size
 - What operations could be done with that location
 - use arithmetic or just comparison

< Computer Data Storage

- Java technology uses different data types that store different number of bits
- Range of values; $(-2^{x-1}$ to $2^{x-1}-1)$ where x is the number of bits
- A data type storing 8 bits would have:
 - 28 possible values
 - Highest and lowest values of (-2^7) and (2^7-1)
- Zero is positive
- Leftmost bit is reserved for the sign: 0 for positive and 1 for negative.

< Programming steps

- Design algorithm – write steps down in pseudocode
- Implement into java – enter source code into TextPad and save it as “.java” file
- Test and debug – compile with “javac” and execute with “java” Or use the applet approach with browser or ‘appletviewer’

< Primary Application Components

- The class code block is the primary structure of the program

- The variables are the data the program works with
- The method code block (*main* in procedural programs) is the structure for the program's actions
- Use braces ({ and }) to define code blocks
- Use semicolon at the end of each statement

< Compiling and Running a Program

- Source file requirements: only one public class per source code file
- Compiling: `javac filename.java`
 - Results: `public_class_name.class`
- Running: `java filename`
`java MyClass`

< Debugging

- Check line referenced in error message or prior line
- Check for semicolons
- Check for even number of braces

< Pre-Written Code to Use in Your Programs

< Simple ways to:

- Print information to the screen
- Use command-line input in a program
- Convert String to ints

< Using System.out.println

- println adds carriage return; print does not
 System.out .print ("print line, do not go to the next");
 System.out .println ("print the line and go the next");
- Use empty print statement for a blank line or use
 ("\n");
- Printing multiple items within one statement:
 - Put quotation marks around text
 - To use spaces, put them inside quotation marks
 - Use + to connect items ("Price is:" + price)
- – Expression result can be printed (price * discount)

< Command-Line Input

– (String args []) lets you pass values to main from the command line

```
1. Public class Intro
2. {
3. public static void main (String args[])
4. {
5. system.out.println ("My name is " + args[0] + "and
I am" + args[1]);
6. }
7. }
```

– Command line: java Intro John 19

– Prints: My name is John and I am 19 years old.

< The Import Statement

- Basic syntax of the import statement:
 import <pkg_name>[.<sub_pkg_name>].<class_name>;
 Or import <pkg_name>[.<sub_pkg_name>].*;
- Examples:

```
import java.applet.*;
import java.awt.*;
import java.io.*;
```

- Precedes all class declarations
- Tells the compiler where to find classes

< Source File Layout

- Basic syntax of a Java source file:

```
[<package_declaration>]
[<import_declaration>]
<class_declaration>
```

- Example, the *VehicleCapacityReport.java* file:

```
Package shipping.reports;
Import shipping.domain.*;
Import java.util.List;
Import java.io.*;
Public class VehicleCapacityReport {
Private List vehicles;
Public void generateReport (writer output) {...}
}
```

< Converting String Arguments to

< Integers

- main considers all arguments strings
 - To evaluate an argument as an int, use
 Integer.parseInt
 Integer.parseInt (variable_to_convert)
 int grade = Integer.parseInt (args[0]);

< Applets

- Why
- html file
- driver file

- class file

< GUI elements

- Label – inform user
- TextField – enter or show one line of information
- TextArea – multiple lines of information
- Button – used to indicate that an action is requested

< Label

- Initialization
 - `Label prompt1 = new Label("information");`
- Use

< textField

- Initialization
 - `textField tf = new textField("information");`
- Use
 - `tf.setText()`
 - `tf.getText()`

< TextArea

- Initialization
 - `– textArea ta = new textArea();`

- Use
 - – ta.setText()
 - – ta.getText()

< Button

- Initialization
 - Button cmd1 = new Button("Drink water");
- Use

< Example -- HTML file

```
<html>
<applet code="ATMApplet.class" width=400
height=400 >
</applet>
</html>
```

< Driver File

- Elements of the interface
- Set up interface
- Process the events

< Class File

- Get the parameters from the driver program
- Process the information
- Return output

< Applets and Applications

- Application standalone
- Applet on java browser

HTML tags

Appletviewer

< Creating Applets

- no main()
- Subclasses of Applet class in java.applet package
- public
- Work as part of browser
- Present GUI and take input from users

< Applet activities

- Initialization
 - When loaded
 - `public void init() { code }`
- Starting
 - After initialization
 - After return to page
 - `public void start() { code }`
- Stopping

- When user leaves the page or applet stops itself
- `public void stop () { code }`
- Destruction
 - When applet is finished or browser exits
 - `public void destroy() { code }` Introduction D01-33
- Painting
 - Whenever applet is brought back to screen
 - `public void paint (Graphics g) { code }`
 - `import java.awt.Graphics;`

< Applet on Web Page

- Tag <applet>
- Testing
- Putting on web

< Tag <applet>

- For including java applets in Web pages

< Testing

- Compile with javac
- Set up html file
- Browser, file, open page, choose file
(navigator)
- File, open, browse (explorer)
- appletviewer

< Align

- Where to place the applet on page
- Followed by
 - <BR CLEAR= [left, right, all] >

< Hspace, Vspace

- Amount of space between applet and other text (pixels)

< Code, Codebase

- Code is filename of applet's main class file
- Must show .class extension
- W/o codebase, in same folder as web page
- Codebase shows either folder or alternate web site

< <object> tag

- More general substitute than <applet> tag for other objects as well as java
- Use <object> in place of <applet>
- classid in place of code
- java: in front of class file

< Source Example

```
import java.applet.*;
import java.awt.*;
public class HelloWorldApplet extends Applet
{
private Label label = new Label("Hello World!");
public void init()
{
```

```
add( label );  
}  
}
```

< To run

- Enter source and save as “HelloWorldApplet.java”
- Write the html file and save it as
- “HelloWorldApplet.html”
- Compile as “javac HelloWorldApplet.java”
- Run it in a browser, using “file, open file” and enter the name of the html file
- Or use appletviewer
 - appletviewer HelloWorldApplet.html