1 # CS 1114 Introduction to Java

(Lab 1)

TA: Maryam Kamvar

mkamvar@cs.columbia.edu

2 ## Homework 0 compiling and running a java program

- http://home.janak.net/cs10034/homeworks/hw0/HelloWorld.java
- Connect to your cunix account (sometimes referred to as a Unix shell)
  - ssh to cunix.columbia.edu
  - Do we need a brief Unix tutorial?
- Open an emacs file called HelloWorld.java (careful with capitalization)
  - In your unix shell, Go to Setup => SSH Forwarding => Display Remote X Applications on Local X Server should be checked.
  - Go to Setup => Save Setup => Save
  - Disconnect, and reconnect. Have a local X server running.
  - In your unix shell type emacs HelloWorld.java &
- Copy-paste code from the internet to the blank emacs file
- Save the file by pressing Ctrl-X Ctrl-S (or use the menu)
- Compile your code.
  - In your unix shell, type **javac HelloWorld.java**
- What does Compilation mean? What does it do? (.class file)
- Run your program
  - In your unix shell type **java HelloWorld**

3 ## HelloWorld - The details

```
/**
 * CS1004
 * HW#0 Sample Code
 *
 * @author Janak J Parekh (janak at cs dot columbia dot edu)
 */
public class HelloWorld {
  /**
   * Print and quit.
   */
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

4 ## Structure of a program the main keywords

5 ## Comments, Classes, Methods

- Comments
  - /* */
  - //
- Classes
  - Every file in Java is a class
  - Constructors (will come later…)
- Methods
  - Written by you
  - Written by the Java people for you to use

6 ## Variables and Statements

- Variables
  - Declaration
    - Just specify the type
  - Definition / initialization
    - Specify the value
- Statements
  - Any “action” in a program.
  - ;

    - Declarations of variables are a statement
    - Methods, and declarations of classes are not a statement. “The { rule”

7 Your first error

- Lets make an “error”
    - “Forget” a semi colon after a semicolon
    - “Accidentally” put an semicolon after something that’s not a statement
    - “Forget” to close a parenthesis
    - Etc etc.
- Syntax errors versus logic errors

8 Next time

- Command line input – args
- Brief intro to arrays
- Mathematical operations
- Conversion between data types
- Assignment 1

1 CS 1114 Introduction to Java

(Lab 2)

TA: Maryam Kamvar

mkamvar@cs.columbia.edu

2 Recap

- Compiling and running programs
- Parts of a program
  - Class declaration
  - Main method
    - What are the parts of any method?
- Statements
- Printing

3 Variables

- *Types*
  - int
  - double
  - String
  - boolean
- *Declaration*
  - *int varOne;*
  - *Double x;*
  - *String str;*
  - *boolean flag;*
- *Initialization*
  - *varOne = 5;*
  - *x=4.5;*
  - *str = "hello world";*
- *Reuse*

4 Variables – con't

- Arrays

5 I/O

Command Line Input

- Command line parameters
- String[] args
- java *filename* input1 input2 3 …
  - *Whats in args[0]*
    - *"Input1"*
  - *Whats in args[2]*
    - *"3"*
  - *What's in args[3]??*

6 I/O

Output with variables

- System.out.println("string literal" + var);

7 Conversion between data types

- Casts

  double num = 3.4;
  int var = (int) num + 5;
  - Useful in division!
- Strings to integers…
  - int input = Integer.parseInt(args[0]);
- Strings to doubles
  - double input =Integer.parseDouble(args[0])

8 Math operators

- Data types are important here!
- +, -, *, /
- ++, --
- %
- &&
- ||

9 Next time

- Methods in detail
  - Having more than one method in your program
  - Method calls
  - Input and variable scope
  - return values
- Conditional statements
- Loops

10 Assignment

- Write a program, Calculator.java (for Java students) that takes two non-zero integer command line parameters, performs the following calculations, and prints out the result of each operation. The operations which you need to implement are:
- a+b (addition) , a*b (multiplication) , a/b (division) , a mod b (modulus) , ((a-1)*(b-1))+(b+1)+(b+2)
- Please return the results in the above order and clearly label each result. See below for a sample output.
- **Sample Output for java Calculator 2 4**

```
    The sum of 2 and 4 is 6
The product of 2 and 4 is 8
2 divided by 4 is .5
2 % 4 is 2
I computed (((2-1)*(4-1))+(4+1)+(4+2) to equal 14
```

1 COMS W1114 - Java Lab

Lab 3
Wednesday, February 11, 2004
&
Thursday, February 12, 2004

2 Note

- Reading:
  - Theory: Ch 0, 5.1-5.3, 1.1-1.6, **4.1-4.4**
  - Programming: Ch 1, Ch 2, 3.1, **3.1-3.7**
- HW1 due today, February 12 at 5p
  - submit programming online
- HW2 is out. Due XX/XX/04.
  - start soon! It's longer that HW1 and will take more time.

3 What we are covering today

- Quick review from lab 2
  - Casting
  - Arrays
- `If…else`
- Iteration/Looping
  - `while` statements
  - `do` statements
  - `for` statements

4 Arrays (1)

- Declaration:
  `int[] myArray = new int[10];`
- Trying to grab cell outside the array bounds causes an error (runtime)
  `myArray[10] = 4;` // error: arrayoutofboundsexception

5 Arrays (2)

Declaring and Initializing an array :

```
          int[] myArray = new int[5];
          //initialize myArray
          myArray[0] = 0;
          myArray[1] = 10;
          myArray[2] = 20;
          myArray[3] = 30;
          myArray[4] = 40;
```

6 `if`

```
if (condition) {
   <statement1>
   <statement2>
}
```

- Recall a `condition` evaluates to a value. **`true`** /**`false`**
- Try it

```
if (myArray.length>0) {
   System.out.println("Inside if.
        Setting myArray[0] to 10.");
   myArray[0]=10;
}
```

7 if…else

1
```
if (condition){
   <statement1>
   <statement2>
}
else {
   <statement3>
   <statement4>
}
```
2
```
if (myArray.length>0) {
   myArray[0]=10;
}
else {
   System.out.println(
    “myArray length is “ +
     myArray.length);
}
```

8 if…else…if

1
```
if (condition){
   <statement1>
   <statement2>
}
else if (condition){
   <statement3>
   <statement4>
}
```
2
```
if (myArray.length<5) {
   myArray[2]=100;
}
else if(myArray.length==5){
         myArray[2]=50;
   }
}
```

9 Iteration/Looping

- Often, we want to do things many times.
- Repetitive tasks often have a structure. Exploit it using loops.
- Let’s look at our array initialization from before:

```
myArray[0] = 0;
myArray[1] = 10;
myArray[2] = 20;
myArray[3] = 30;
myArray[4] = 40;
```

  – Is there a pattern?

10 The `while` loop (1)

- “While ‘condition’ is true, run my statements.”

```
while (condition){
   <statement1>
   <statement2>
}
```

- Similar to the `if` structure
- What makes it stop?
  – You must do something to make condition evaluate to false!

11 The `while` loop (2)

- Let’s solve our array initialization problem using `while`

```
while (what is true?) {
   <do what?>
   }
```

Think about our pattern…

12 The `while` loop (3)

- What do we want to do?

```
while (what is true?) {
   myArray[i]=i*10;
   }
```

13 The `while` loop (4)

- When do we stop?

```
while (we have not looked at all cells) {
   myArray[i]=i*10;
   }
```

- **How many cells are there?**
  - **5 (actually myArray.length)**
- **Where do we start?**
  - **0 (beginning of myArray indices)**
- **How does the index “a” change?**
  - **increment by 1**

14 The `while` loop (4)

- Put it all together

```
int i = 0;
while (i < myArray.length) {
   myArray[i]=i*10;
   i=i+1;
   }
```

15 The `while` loop (5)

Try it:

```
int i = 0;
while (i < myArray.length) {
   myArray[i]=i*10;
   System.out.println(
   “i=“ +i+ ” and myArray[“+i+”] is “+myArray[i]);
   i=i+1;
   }
```

16 The do loop (1)

```
do {
   <statement1>
   <statement2>
   …
} while (condition);
```

- Similar to `while` statement. So what’s the difference?

17 The `do` loop (1)

```
do {
   <statement1>
   <statement2>
   …
} while (condition);
```

- Similar to `while` statement. So what’s the difference?
  - The <statement>s are guaranteed to run at least once.

18 The `do` loop (2)

- Can we populate myArray as before using a do loop instead?
  - Yes!

  What do we need?
- An index.

  `i = 0;  //reuse index i`
- Statement.

  `myArray[i]=i*10;`
- Increment.

  `i++;  //same as i=i+1`
- Condition.

  `i<myArray.length`

19 The `do` loop (3)

```
i=0; //reusing prior index
do {
   myArray[i]=i*10;
   i++;
} while (i<myArray.length);
```

- Potential problems?

20 The `for` loop (1)

- What have we seen is needed for looping?
  - a looping variables (`i` in previous examples)
  - a start value for looping variable
  - a stop condition
  - a way to change the looping variable so we reach our stop condition
- The `for` loop is no different. Just a different structure.
  - not based on the ‘if’

21 The `for` loop (2)

```
for (int var = start; check; update) {
   <statement1>
   <statement2>
}
```

- `int var = start` (creating our loop variable)
- `check` (our condition)
- `update` (our changing the loop variable)

22 The `for` loop (3)

Let's do our `myArray` changes again with `for`:

```
for (int i = 0; i<myArray.length; i++) {
   myArray[i]=i*100;
}
```

23 The `for` loop (3)

Let's do our `myArray` changes again with `for`:

```
for (int i = 0; i<myArray.length; i++) {
   myArray[i]=i*100;
}
```

`int var = start` (creating our loop variable)

24 The `for` loop (3)

Let's do our `myArray` changes again with `for`:

```
for (int i = 0; i<myArray.length; i++) {
   myArray[i]=i*100;
}
```

`check` (our condition)

25 The `for` loop (3)

Let's do our `myArray` changes again with `for`:

```
for (int i = 0; i<myArray.length; i++) {
   myArray[i]=i*100;
}
```

`update` (our changing the loop variable)

26 The `for` loop (3)

Let's do our `myArray` changes again with `for`:

```
for (int i = 0; i<myArray.length; i++) {
   myArray[i]=i*100;
}
```

Our Statement.
That's it!

27 Some things to think about

- How would we loop backwards using a for loop?
  - with a while? or do…while?
- Do we always have to change the condition variable by 1?

- Can we have complex conditions? (&&, ||, !, etc.)

28 Wrap up

- Next time:
  - Methods
  - More decision and control statements
  - Basic Input/Output (I/O)
- HW2 is out. Due TUESDAY XX/XX/04
  - Get started. Longer than HW1.

1 COMS W1114 - Java Lab

Lab 4
Wednesday, February 18, 2004
&
Thursday, February 19, 2004

2 Note

- HW2 is already out, due February 26 at 5pm
- HW1 is graded.

3 What we are covering today

- Quick review from lab 3
  - `If…else`
  - Iteration/Looping
- Switch Statements
- Methods
- Variable scope

4 `if…else…if`

1
```
if (condition){
      <statement1>
      <statement2>
}
else if (condition){
      <statement3>
      <statement4>
}
else{
      <statement5>
}
```

2
```
if (args.length==0) {
      System.out.println(“no input entered”);
}
else if(myArray.length==1){
      System.out.println(“one input entered”);
}
else {
      System.out.println(“>1 input entered”);
}
```

5 Methods

- A method groups together statements in a logical manner
- So far we have seen a single method in any given java program
```
public static void main(String[] args){
     //method body (statements) goes here
}
```
- Components of a method declaration
  - public :: other java classes could hypothetically call this method
  - static ::
  - void :: return type
  - main :: identifier – name of method
  - ( ) :: delimits the input variables
  - String[] args :: the input variable TYPE and NAME (>1 variable are comma separated)

6 Methods

- There can be more than one method in a program. The way to jump from method to method is by **calling** the method

7 main method is static

- The main method declaration will *always* look like that
- It *must* always be declared static

- Therefore it is not an ideal place to write the body of your program

8 Constructors

- Every class has a special method called a constructor.
- Like main, the constructor has a special syntax. No return value etc, only needs an identifier. The identifier *must* match the class name.

9 Constructors can have input

10 Example.java with a constructor

11 HW2

- IMPORTANT! Name your class Palindrome
  - (your file should be called Palindrome.java)

- Write a method called *isPalindrome* that takes one parameter (a string) and returns a boolean (Java) indicating whether or not the supplied String is a palindrome.
- Modify the palindrome checking procedure so that it's case-insensitive. Hint: use java's Character.toLowerCase method
- Modify the palindrome checking procedure so that it ignores whitespace and punctuation. In particular, handle spaces ( ), periods (.), commas (,), and apostrophes (')

12 The Java API

- The java API contains information about all of java's methods

1 # COMS W1114 - Java Lab

Lab 5
Wednesday, February 25, 2004
&
Thursday, February 26, 2004

2 ## Note

- HW2 now due Tuesday, March 2 at 11a
  - Extra credit for submissions at original date of 2/26 @ 5p
- HW1 Theory is graded
- Midterm on March 9 (<2 weeks!)

3 ## What we are covering today

- Review from Lab 4
  - Switch Statements
  - Methods
  - Constructors
- Variable scope
- Basic I/O
- Big Picture so far

4 ## `if…else…if`

1

```
if (condition){
      <statement1>
      <statement2>
}
else if (condition){
      <statement3>
      <statement4>
}
else{
      <statement5>
}
```

2

```
if (args.length==0) {
      System.out.println("no input entered");
}
else if(myArray.length==1){
      System.out.println("one input entered");
}
else {
      System.out.println(">1 input entered");
}
```

5 ## Methods

- A method groups together statements in a logical manner
- So far we have seen a single method in any given java program

```
public static void main(String[] args){
      //method body (statements) goes here
}
```

- Components of a method declaration
  - public :: other java classes could hypothetically call this method
  - static ::
  - void :: return type
  - main :: identifier – name of method
  - ( ) :: delimits the input variables
  - String[] args :: the input variable TYPE and NAME (>1 variable are comma separated)
- There can be more than one method in a program. The way to jump from method to method is by **calling** the method
- Components to a method call:

  input values, return value

6 ## Constructors

- Every class has a special method called a constructor.
- Like main, the constructor has a special syntax. No return value etc, only needs an identifier. The identifier *must* match the class name.
- In the main method, we will be calling the constructor for the class. to call the constructor method, we use the keyword 'new' before its identifier

- Difference between a regular method call and a constructor method call
  - *new* keyword
  - **never** a return value

7 Example of constructors

8 Variable Scope

9 Basic I/O = Input/Output

- Interactive Input
  - So far we've seen command line input and output to System.out
  - Not interactive
- Package Access
  - use objects someone else writes
- File I/O
- Basic Exception handling

10 Interactive Input(1)

- Java uses *streams*
  - simply a sequence of data that comes from a source
    - keyboard data
    - file data

- There are predefined classes to use!
  - InputStreamReader
  - BufferedReader

11 Interactive Input(2)

- InputStreamReader
  - (check out the Java API)
- BufferedReader
  - (check out the Java API)
- To use them, we must import them; they are not default features.
  - use the **import** statement
  - **import** belongs at the beginning of your class file
  - import each class
    - import java.io.InputStreamerReader;
    - import java.io BufferedReader;
    - OR
    - import java.io.*;

12 Interactive Input(3)

13 Interactive Input(4)

- Create a BufferedReader called *stream*…

```
BufferedReader stream = new BufferedReader();
```

let's look at the constructor for BufferedReader in the API
- We need to send it an input stream. So, create an InputStreamReader object:

```
new InputStreamReader();
```

- We see from the API it needs an input stream to connect to.
  Use System.in (look familiar?)
    new InputStreamReader(System.in);
- Put it all together:

BufferedReader stream = new BufferedReader(new InputStreamReader(System.in));

14 Interactive Input(5)

15 Interactive Input(6)

- Now we have an object called *stream* to use
- It gives us access to System.in (here, the keyboard)
- So how do we use it?
  - look at the methods a BufferedReader has
    - read()   Read a single character.
    - readLine()   Read a line of text. After you hit [Enter].
    - others…

16 Interactive Input(7)

17 Interactive Input(8)

- That's it? Yes.
- How to read non Strings?
  - as we did with command line input via *args*[] but read from *stream* instead:

double d = Double.parseDouble(**stream**.readLine()).doubleValue();

18 File I/O

- Input similar. Using different objects:
  - FileReader   instead of   inputStreamReader
  - new File("filename");   instead of System.in
  - No BufferedReader equivalent needed (for now)

FileReader inFile = new FileReader(new File("inputfile.txt"));
char input = inFile.read();

- Need to explicitly close our Files
  inFile.close();
- We'll cover output in next lab.

19 Basic Exception Handling(1)

- What happens if 'readLine()' called but you are at end of file (EOF)
  - an **Exception** (EOFException) is *thrown*
- What happens if there is a problem while keyboard input?
  - an **Exception** (IOException) is *thrown*

20 Basic Exception Handling(2)

```
try {
   <statements>
}
catch ( Exceptiontype e1) {
    statements to react and recover
}
```

```
catch ( Exceptiontype e2) {
    statements to react and recover
}
…etc
```

21 Basic Exception Handling(3)

- As we saw, we need to declare **`throws ExceptionType`** when an object throws an exception
- We need to catch the exception somewhere with the `try…catch` block.

22 The Java API

- API = Application Programming Interface
  - interface: a contract for objects
- The java API contains information about all of Java's Objects
  - http://java.sun.com/j2se/1.4.2/docs/api/

  - Constructors
  - Methods
  - Fields

23 The Big Picture So Far

- We've covered the fundamentals of programming:
  - Datatypes: Primitives, Objects, Arrays
  - Iteration/Looping: While, For, do…while
  - Conditionals: if…else…elseif, switch statement
  - Objects: Constructors, Methods, a Variable's Scope
  - Basic I/O: interactive I/O, file I/O, Basic Exception Handling
- What's next? How to use what we've learned do to something useful
  - Coding practices, Debugging tools, advanced I/O
  - Object Oriented (OO) Design
    - properties, references, abstraction, inheritance
  - GUIs, Event based programming
  - Packaging code, more Java API

24 Notes

- We have covered through Chapter 4 in Java Gently. Make sure you are caught up. There are many details to be sure you know.
  - We are not using the book's Display and Stream objects, so do not confuse those with what we did here.
- HW3 is going out. Start early! It is longer than what we've seen so far.
- Midterm on Tuesday, March 9.

1 COMS W1114 - Java Lab

Lab 6
Wednesday, March 3, 2004
&
Thursday, March 4, 2004

2 Note

- See me if you haven't picked up your graded HW1
- Midterm on March 9 (<1 weeks!)

3 What we are covering today

- Review from Lab 5
  - Interactive Input
  - File Input
- File Output
- Debugging Strategies
- Concept Review

4 Interactive Input

try-catch

5 File Input

try-catch

6 Interactive Input

methods that throw an exception

7 File Input

methods that throw an exception

8 File Output

- Instead of printing to the console (using System.out) we want to print to a file
- We need to create our own output object to redirect the output

9 Coding Practices

- Style
  - Class names are capitalized and match the filename
  - Variable names and Method names have a lowercase first letter and should be descriptive
  - Indentation – keep it structured! Tab every line in emacs for proper indentation setup
  - Usually main method is the last method in your class
  - Usually, constructors are the first method in your class
    - (the only thing that would be before your constructor are global (member) variables)

10 Debugging with print statements

Prelab activities

11 ▣

Fill in the blank

- A(n) _______ can store exactly one value as a time, whereas one _______ can contain individual data members
- Importing the _________ *class* allows the program to use Java's File Reading capabilities
- Importing the _________ *package* allows the program to use Java's File Reading capabilities
- _______ converts its String argument to a integer value
- A ________ variable is declared in the body of the class, but not within the body of a method
- A ________ variable is declared in the body of a method
- An if statement's condition is enclosed in _________
- If a method does not return a value, the return value type is __

12 ▣

Fill in the blank

- Conditional AND is ________
- Conditional OR is ________
- Logical Negation is _______
- ALL of the above are ________ operators
- Name these structures:
  - Repetition statement that tests the loop-continuation condition at the end of the loop, so that the body of the loop will be executed at least once.
  - Handles a series of decisions, in which a particular variable or expression is tested for values it can assume and different actions are taken
  - Handles all the details of counter-controlled repetition

13 ▣

Open-ended questions

- When will an infinite loop occur?

- Method Calls vs. Method Declarations

- What is variable scope?

1 COMS W1114 - Java Lab

Lab 7
Wednesday, March 10, 2004
&
Thursday, March 11, 2004

2 Note

- HW3 Due by Tuesday, March 23, at 11:00am
- Any midterm questions? How was it?

3 What we are covering today

- Review from Lab 6
  - File Output
  - Debugging Strategies
- Formatting & Advanced I/O
  - Strings
  - StringTokenizer
  - Envelopes
- HW3
  - review assignment specifications
  - Readme Files

4 File Output

- Instead of printing to the console (using System.out) we want to print to a file
- We need to create our own output object to redirect the output

5 Debugging

- Syntax vs. Semantic errors
- Basic testing (aka going beyond "it compiles!")
- Your friend: System.out.println();

6 Strings(1)

- String is a class. NOT a primitive data type.
```
String big = "hippopotamus";  //note: not using new
```
- String has built-in + operator
```
String s1="4";
String s2="5";
String s3 = s1 + s2;  //  s3 is "45";
```
- Once created, cannot be changed
  - string can be assigned new value but not inherently changed.
- A difference between initialized but empty Strings and non-initialized (null) Strings.
```
String s1;
String s2 = "";
```

7 Strings(2)

- String as several constructors (see Java API)
  - We will usually use assignment from a string literal or variable\
```
String s3="Hello";
String s4=args[0];
String s5= in.readLine(); //assume you have a reader in
```
- Strings have many methods (see Java API)
  - class methods
  - instance methods

8 Strings(3)

- Examples:

```
String big=“hippopotamus”;
char study [] = big.toCharArray();

int bang = big.indexOf(“pop”);
int fizz = big.indexOf(“up”);

String small = big.substring(3,5);

System.out.println(big.equals(small);
System.out.println(big.compareTo(small);
```

- What do we expect as the result of each?
- ChequeDetector.java example from book.

9 StringTokenizer(1)

- Breaks a String into tokens
  – Tokens are substrings separated by some character (space, comma, tab, etc.)
- declaration

```
StringTokenizer st = new StringTokenizer(aString);
```

- Now easily iterate through the tokens

```
 while(st.hasMoreTokens()){
     System.out.println(st.nextToken());
}
```

- Assume

```
String aString=“The quick brown fox jumped over the lazy dog.”;
```

- What does the above code do?

10 StringTokenizer(2)

- You do not have to tokenize on a space (“ “). You can change the delimiter when you declare the StringTokenizer:

```
 StringTokenizer st = new StringTokenizer(aString,”,. ”,false );
```

- Now what happens with:

```
String aString=“Really, the quick brown fox jumped over the (lazy) dog.No joke.”;
StringTokenizer st = new StringTokenizer(aString,”,. ”,false);
  while(st.hasMoreTokens()){
     System.out.println(st.nextToken());
}
```

11 Envelopes (aka. wrappers)

- We’ve seen them already - used for data conversion
- Integer and int conversion examples:

```
Integer myInteger = new Integer(50);
Integer myInteger2 = Integer.valueOf(“100”);
int iterations = Integer.parseInt(argv[0]);
int a = myInteger2.intValue();
int b = myInteger.parseInt(“100”);
```

  – results of each?

Also:

- Java rule: Values of primitive types and Objects cannot be mixed
- This is only a problem when a package requires an object and all we have is a primitive. Must “wrap” our primitive with an Object (Boolean, Character, Double, Float, Integer, Long)

12 HW3

- Let’s review the specs
  – be sure to follow the naming conventions
  – be sure to only what you are asked (I.e. no spurious System.out.print’s
- A Readme file
  – a TEXT file (no msword, postscript, etc. think: notepad)
  – includes your name and UNI

- homework #
- a sentence or two outlining what your program does
- a few instructions how to run your program
- list known limitations or bugs
- add anything else you deem important.

13 Example Readme

```
William Beaver  (wmb2013)
cs1004 HW3 - Bank.java

My program, Bank.java, keeps  track of bank accounts by name and balance. It uses two  arrays -- one for people's
names and one for their balances.  While running the program, type “h” for help on the commands to execute.
etc…

To compile, type “javac Bank.java” at the command prompt
To run, type “java Bank” after compiling.
Follow the onscreen instructions to run the program.

There are no known bugs or limitations based on the significant testing I’ve done.

I added a feature to etc…in addition to the original specifications.  Etc. Etc.
(end.)
```

14 Next time

- Formatters
  - Locale
  - DateFormat
  - SimpleDateFormat
  - NumberFormat
  - DecimalFormat
  - MessageFormat
- Begin OO (Design, properties, references)

1 COMS W1114 - Java Lab

Lab 8
Wednesday, March 24, 2004
&
Thursday, March 25, 2004

2 Note

- HW4 out
- You should have your grades for hw1, hw2, and your midterms

3 What we are covering today

- Review from Lab 6
  - How to use the Java API
- Object Oriented design
  - Example.java

4 Java

5 Example.java

- PROBLEM DESCRIPTION
- We want to keep a record of all the students in cs1004
  - Two objects
    - Lab object
      - 4 Member Variables of this object are:
        - » lab number, lab instructor, number of students in the lab, list of the students in the lab
    - Student object
      - 3 Member Variables of this object are
        - » first name, last name, midterm grade

6 Notes

- For classes which are only going used as data objects *you don't need a main method*
- Note that if you specify a method will throw an exception *at the method declaration* you must try – catch the **method call**
- **One more time: method calls vs. method declarations**

7 To Do:

- Comment the code to show you understand what's going on
- Add an attribute called "average" to your lab object – this will represent the midterm average of the students in the lab
- In Lab.java, add a method `average()` that will go through each student in the list, and compute the average of their midterm grades
- After every time you add a new student to a lab, calculate the new average by calling that labs average method.
- In the "lab info" print statement, also print out the lab's midterm average

8 Structure of our classes

9 Next Time

- Method/Constructor Overloading
- More Object Oriented design

1 # COMS W1114 - Java Lab

Lab 9
Wednesday, March 31, 2004
&
Thursday, April 1, 2004

2 ## Note

- HW4 out. Due Tuesday 11a.

3 ## What we are covering today

- Quick review/question of Lab 8.
- More OOD! (wrap up Java Gently. Ch 8.)
  - overloading/overriding
  - Properties: private and final
  - references

4 ## Structure of our classes

5 ## Questions from Lab8?

- Add an attribute called “average” to your lab object – this will represent the midterm average of the students in the lab
- In Lab.java, add a method `average()` that will go through each student in the list, and compute the average of their midterm grades
- After every time you add a new student to a lab, calculate the new average by calling that labs average method.
- In the “lab info” print statement, also print out the lab’s midterm average

6 ## Modeling a Point

- Todays Problem: model a point in 2D Cartesian space.
- What do you model? A few things:
  - Access each component of a point independently
  - Shift the point if you are given offset
  - Rotate the point 90 degrees
  - Calculate the Euclidean distance from your Point to another
  - Calculate the midpoint between your point and another
  - Determine if your point is equal to another

7 ## Some help

- Rotate a point 90 degrees
  - New x = old y
  - New y = old x * -1
- Distance between two points
  - Difference between each component.
  - Use Pythagorean Theorem to calculate the distance as the squareroot of the squared distance.
- Midpoint
  - A point
  - Each coordinate is the midpoint of the difference of each component

8 ## Next Time

- Even More OOD!

– Inheritance, abstraction (Java Gently. ch 9)

1 COMS W1114 - Java Lab

Lab 10
Wednesday, April 7, 2004
&
Thursday, April 8, 2004

2 Note

3 Some Definitions

- **`this` identifier**
  - every object can access all of its members by name. The *full name* of the identifier is `this.name`
  - The use of `this` is only needed when there is ambiguity over variable names in a particular scope

- **overloading a method** - where you provide different versions of a method, but keep the same name. The parameter list (input) for the two methods must be distinct so that there is no confusion about which is being called:

```
public void withdraw(String balance, int double){
        //do stuff here
}
public void withdraw(String balance, String double){
        withdraw(balance, Double.parseDouble(double));
}
public void withdraw(String balance){
        withdraw(balance, 60.0);
}
```

- **overriding methods** – a subclass can decide to supply its own version of a method already supplied.
  - All of the classes we create are a subclass of the Object Class in Java

4 Object Class Methods

- equals
- The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true).

- toString
- Returns a string representation of the object. In general, the toString method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method. The toString method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object. In other words, this method returns a string equal to the value of getClass().getName() + '@' + Integer.toHexString(hashCode())

5 More Definitions

- `instanceof`
  - The instanceof operator tests whether its first operand is an instance of its second. boolean val = op1 instanceof op2 op1 must be the name of an object and op2 must be the name of a class. **An object is considered to be an instance of a class if that object directly or indirectly descends from that class**.

6 Program from last lab…

7 Questions

- how could we use instanceof operator here?
  - mp12 instanceof Point
  - mp12 instanceof Object

8 Inheritance

- **An object is considered to be an instance of a class if that object directly or indirectly descends from that class**.

- descent?!
- so far, we have created classes with no hierarchy between them:
  - remember:

- But is it useful to create a hierarchy when classes are related:

9 Inheritance Example

```
class Person {
     protected String name;  // a protected variable can be accessed ONLY
     protected int age;      //  from this class and the classes descend from this class…

     public Person(String n, int age){
      name =n;
      this.age = age;
     }
     public String toString(){
      return name;
     }
}

class Student extends Person {  //since student extends Person, student inherits all the properties that Person had…
      private  int studentID;
      private double GPA;
      public Student(String n, int a, int id, double gpa){
               super(n,a);   // we are calling the super-class's constructor MUST BE THE FIRST LINE OF CODE. this is called              //superconstructing

               studentID=id;
               GPA = gpa;
      }
      public String toStrint(){
               return "Student: " + name;  //name refers to the PROTECTED Person variable
      }
}
```

10 Some questions

- Review of modifiers
  - public
  - protected
  - private
- how could we use instanceof operator here?

```
Person p = new Person("Maryam", 23);
– p instanceof Person ?
– p instanceof Student ?

Student s = new Student("Jack", 19, 223344, 3.7);
– s instanceof Person ?
– s instanceof Student ?
```

11 Casting examples using inheritance properties

- any object of a superclass can be assigned to a subclass with a cast!
  - `Person p = new Person("Maryam", 23);`
  - `Student s = (Student) p;`   `// the id and gpa of student s are currently null!!`
- any object of a subclass can be assigned to an object of its superclass!
  - `Student s = new Student("Brit", 22, 6655, 2.7);`
  - `Person p = s;`

12 Static

- a static variable
  - a member (global) variable which exists only ONCE even though there may be multiple objects created.
  - good for when you want to collect information about an entire class (vs. a single object) such as
    - static int students_created;
    - Student.students_created; ← access it from the class, not the object!
- a static method
  - not part of the a specific object, part of the general class
  - main method must be static. no other methods should be declared static
  - you can never access an object's main() from another object.

1 COMS W1114 - Java Lab

Lab 11
Wednesday, April 14, 2004
&
Thursday, April 15, 2004

2 Notes

- HW5 out. Due Tuesday 11a. Any questions?
- Only three more labs :(
  - Today: GUI programming! awt and Swing (Ch 9)
  - April 22 - Event-based programming (Ch 10)
  - April 29 - 1) applets and 2) code packaging/APIs
- Homework 6 out next Tuesday-ish

3 Lab 10 Review (1)

- this identifier - The use of this is only needed when there is ambiguity over variable names in a particular scope.
- overloading a method - where you provide different versions of a method, but keep the same name.
- overriding methods – a subclass can decide to supply its own version of a method already supplied.
- Object class methods - ie. toString(), equals, etc.
- instanceOf operator - tests whether its first operand is an instance of its second.
  - boolean val = op1 instanceof op2;
  - op1 must be the name of an object and op2 must be the name of a class. An object is considered to be an instance of a class if that object directly or indirectly descends from that class.

4 Lab 10 Review (2)

- Inheritance - An object is considered to be an instance of a class if that object directly or indirectly descends from that class.
- Protected - a protected variable can be accessed ONLY from this class and the classes descend from this class.
- Static
  - a static variable - a member (global) variable which exists only ONCE even though there may be multiple objects created.
  - a static method - not part of the a specific object, part of the general class

5 Graphics and UIs

- Two packages for dealing with graphics (for now)
  - java.awt (awt - the abstract windowing toolkit)
  - javax.swing (swing)
- Each provides access to tools/code for writing GUIs, drawing, etc.
- awt uses much of the OS's facilities - so UIs look like the platform they are run on
- swing is implemented independently of the OS
- They each are quite large packages and, like many things in the class, you can take an entire course on them alone.
- We will start with awt, then migrate to swing next lab. They are rather similar.
- Our goal: be able to write some simple graphics programs.

6 Graphics and UIs

- Structure of AWT (diagram in book. pp 385)
  - graphics
  - components (windows and menus too)
  - layout managers
  - event handlers
  - image manipulation

7 Example 1

- Look at warning box example from the book (pp 389). We want to display a window with some text in it.
- Frame - the basic window
    - Frame is a subclass of the Window component
    - Our code will inherit the Frame code
- Add Graphics (paint and repaint methods)
- Viola!
- see sample code “Warning.java”
- Pretty simple, right?

8 Example 2

- Remember our old Point class from lab 9? (NOT the point class you are building for HW5!)
- Recall: modeled a point in 2D Cartesian space.
- [See javadoc for Point]
- Now, say we want to plot the points in a graph on your machine? Let’s build a Plotter2 class that is a real plotter!
- Where to start? Just like before:
    – Frame - the basic window
    – Add paint (and repaint) using the Graphics object
    – Here, our painting is a bit more imvolved.

9 More Graphics

- Now, what if you want more than one window or more control?
    – a Canvas

- Walkthrough book example 10.3 (FlagMaker2)

10 A few more graphic objects

- Frame and Canvas are great for simple drawing. What if you want to make an interactive application?
- Want Labels
    – `add(new Label (“some text”));`
- Want Buttons
    – a little more involved, but rather straightforward
    1. create a Button object
       `Button myButton = new Button(“Submit”);`
    2. add it to the Frame/Canvas - recall, these are Container objects. Note that Containers have this *add* method (seen with Labels)
       `add(myButton);`
- Why no x/y coordinates for the Button???
    – there is a *Layout Manager* to coordinate placement (nice :)

11 Layout Manager

- Layout Manger take control of the over the positioning of components and arrange them sensibly.
- There are 5 different managers! We’ll only talk about three:
    – FlowLayout, BorderLayout(default) and GridLayout

```
setLayout(new Manager(parameter)); //format
```

example:

```
setLayout(new FlowLayout(FlowLayout.CENTER,horigap,vertigap));
```

We’ll see it used in a minute….

12 Simple Event

- Make a button do something
- We have our button myButton and we’ve added it

```
Button myButton = new Button(“Submit”);
  add(myButton);
```

- Now need to “listen” for actions/events we care about

```
myButton.addActionListener (this);
```

this means the current frame will be responsible for the code for some *ActionPerformed* method(what?! pretty easy….)

```
public void actionPerformed (ActionEvent e){
 if (e.getSource() == buttonname1) {
         statements;
 } else
 if (e.getSource() == buttonname2) {
         statements;
 } //etc
}
```

13

Putting it all together

- (See the ButtonTest code example)

1 ## COMS W1114 - Java Lab

Lab 12
Wednesday, April 21, 2004
&
Thursday, April 22, 2004

2 ## Note

- Last homework will be out soon. You will be using AWT to create a GUI (graphical user interface)
- Your grades are now up off of a link on the course website. Report any errors to Janak!

3 ## What we are covering today

- Review from Lab 9
  - AWT
  - Graphics object
- Event based programming

4 ## AWT

- AWT is a java package that we will be using in order to create graphical user interfaces
- Some important classes within the AWT package
  - Containers:
    - Frame ← has an titlebar, can contain many 'things'
    - Canvas
    - Panel
  - What we will generally do is create our own class, which ***extends*** one of the above classes
  - Each of the above containers has a paint method that we will ***inherit*** but will usually ***override*** when we want to customize the container's graphcs.

5 ## Paint method

you never have to call the paint method. Java will automatically call the paint method for you:
    1) when the container appears
    2) when the container is being moved around

```
public void paint(Graphics g){
    //java code here
}
```

but if you want to explicitly repaint your canvas without waiting for the user to move the window around you should call

```
    repaint();
```

6 ## Paint method cont'd

```
public void paint(Graphics g){
    //java code here
}
```

so what's the deal with Graphics g?
g is the variable name of the Graphics object that is passed into the paint method automatically. (this can be renamed)
In the Graphics class, you will see many useful methods
    drawLine(….);
    fillCircle(….); etc

which you can now access through the graphics object!

```
g.drawLine(10,20, 30, 40);
g.fillOval(5,4,2,2)
```

7 More awt objects

- Frame and Canvas are great for simple drawing. What if you want to make an interactive application?
- Want TextFields
  - TextField t = new TextField("initial text", 15);
  - add (t)
- Want Labels
  - `add(new Label ("some text"));`
- Want Buttons
  - a little more involved, but rather straightforward
  1. create a Button object
     `Button myButton = new Button("Submit");`
  2. add it to the Frame/Canvas - recall, these are Container objects. Note that Containers have this *add* method (seen with Labels)
     `add(myButton);`
- Why no x/y coordinates for the Button???
  - there is a *Layout Manager* to coordinate placement (nice :)

8 awt objects

- awt objects (like every other java object) has methods associated with them
- for example the once you create a TextField, you can call methods such as getText() which will return the string inside your textField.
  - explore the API!

9 Layout Manager

- when you add components, you are adding them to your container, given that you have previously specified one (or will default to borderLayout)
- Layout Manger take control of the over the positioning of components and arrange them sensibly.
- There are 5 different managers! We'll only talk about three:
  - FlowLayout, BorderLayout(default) and GridLayout

```
setLayout(new Manager(parameter)); //format
```

example:

```
setLayout(new FlowLayout(FlowLayout.CENTER,horigap,vertigap));
```

10 Simple Event

- Make a button do something
- We have our button myButton and we've added it
  ```
  Button myButton = new Button("Submit");
     add(myButton);
  ```
- Now need to "listen" for actions/events we care about
  ```
  myButton.addActionListener (this);
  ```
  this means the current frame will be responsible for the code for some *ActionPerformed* method(what?! pretty easy….)

```
public void actionPerformed (ActionEvent e){
if (e.getSource() == buttonname1) {
       statements;
} else
if (e.getSource() == buttonname2) {
       statements;
} //etc
}
```

11 Different Kinds of events

- so far we've only worked with ActionEvent which reports if any action has been performed on a specified component

12 so what would you do to get info from a textfield?

- lets write the pseudo code.

13 Interfaces

- so you want to use one of the event listeners?
- java has Listener interfaces which specifies the methods that the listener MUST defined (listed on previous slide and on pg 423)
- if you want to detect any of the actions, you need to implement its Listener, and then be sure to define all its methods!

- see code example for syntax

14 End Notes

- Fill out the course evaluation! Win your iPod

http://oracle.seas.columbia.edu/wces/

- Please also remember to rate your TAs *(you can rate any TA in this class, not just your lab instructor!)*
- Maryam will be out of the country starting on Sunday 4/25- Thursday 5/6.

1 COMS W1114 - Java Lab

Lab 13
Wednesday, April 28, 2004
&
Thursday, April 29, 2004

2 Notes

- HW5 ready
- Your grades are now up off of a link on the course website. Report any errors to Janak!
- HW6 Due Wed, May 5 @ 5p.
- Check bboard for OH changes

3 What we are covering today

- Go over HW5 solution
- Review from Lab 12
  - Event based programming
- Applets
- Packages
- Review

4 Simple Event

- Make a button do something
- We have our button myButton and we've added it

```
Button myButton = new Button("Submit");
   add(myButton);
```

- Now need to "listen" for actions/events we care about

```
myButton.addActionListener (this);
```

this means the current frame will be responsible for the code for some *ActionPerformed* method(what?! pretty easy….)

```
public void actionPerformed (ActionEvent e){
if (e.getSource() == buttonname1) {
      statements;
} else
if (e.getSource() == buttonname2) {
      statements;
} //etc
}
```

5 Different Kinds of events

6 Interfaces

- so you want to use one of the event listeners?
- java has Listener interfaces which specifies the methods that the listener MUST defined (listed on previous slide and on pg 423)
- if you want to detect any of the actions, you need to implement its Listener, and then be sure to define all its methods!

- see code example for syntax

7 Applets (1)

- Want to display your programs/GUIs in a web browser?
- Write an applet! Really simple since we know how awt works
  - Import Applet and Graphics

```
import java.applet.Applet;
import java.awt.Graphics;
```

- Extend the Applet class
  - public class HelloWorld extends Applet{}
- Implement some Applet methods (paint!)

```
public void paint(Graphics g) {
   g.drawString(“Hello World”,50,25);
   }
}
```

8 Applets (2)

- Run it in a Browser

```
html here
<APPLET CODE=“HelloWorld.class” WIDTH=150 HEIGHT=25> </APPLET>
more html
```

- That’s it!
- http://java.sun.com/docs/books/tutorial/applet/

9 Packages

- We’ve seen package use already
  import java.awt.*;
- What if we want to write a class that conflicts with an existing name?
  - We package our code
  - Use the package command
  - Check out
  java.sun.com/docs/books/tutorial/java/interpack/packages.html

10 Congratulations!

- We’ve covered the fundamentals of programming:
  - Datatypes: Primitives, Objects, Arrays
  - Iteration/Looping: While, For, do…while
  - Conditionals: if…else…elseif, switch statement
  - Objects: Constructors, Methods, a Variable’s Scope
  - Basic I/O: interactive I/O, file I/O, Basic Exception Handling
- How to do things:
  - Coding practices, Debugging tools, advanced I/O
  - Object Oriented (OO) Design
    - properties, references, abstraction, inheritance
  - GUIs, Event based programming

11 Congratulations!

- You’ve built:
  - A simple calculator reading program arguments
  - A palindrome checker reading keyboard input
  - Bank Account Manager w/ interactive interface
  - Shape calculator w/ interactive interface
  - Shape plotter w/GUI
- What are you going to build next?

12 End Notes

- Thank you! (from Maryam too)
- Fill out the course evaluation! Win your iPod
http://oracle.seas.columbia.edu/wces/
- Please also remember to rate your TAs *(you can rate any TA in this class, not just your lab instructor!)*
- OH changes - check bboard