COMS W1114 - Java Lab

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

---

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!)

---

What we are covering today

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

```java
if (condition1)
  <statement1>
  <statement2>
else if (condition2)
  <statement3>
  <statement4>
else
  <statement5>
```

```java
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");
}
```

switch

```java
switch (variable)
  case value1:
    <statement1>
    <statement2>
    break;
  case value2:
    <statement3>
    <statement4>
    break;
  default:
    <statement5>
}
```

```java
switch (args.length) {
  case 0:
    System.out.println("no input entered");
    break;
  case 1:
    System.out.println("one input entered");
    break;
  default:
    System.out.println(">1 input entered");
}
```

Methods

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

```java
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.

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.
Example of constructors

class Example {
  Example(int a, int b) {
    if (isValidDiv(a, b));
    // this is the method call!
    System.out.print(a + " / " + b + " is " + divide(a, b));
    else
      System.out.println("you tried to divide by 0");
  }
  public boolean isValidDiv(int a, int b) {
    if (b == 0) {
      return false;
    } else {
      return true;
    }
  }
  public double divide(int a, int b) {
    // we assume isValidDiv was called so we will not divide by zero
    return (double) a / (double) b;
  }
  public static void main(String[] args) {
    System.out.println("program starts here");
    new Example(Integer.valueOf(args[0].intValue(), Integer.valueOf(args[1].intValue()));
  }
}

Variable Scope

Member Variable Scope

Method Parameter Scope

Local Variable Scope

Exception-handler Parameter Scope

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
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

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.InputStreamReader;
    - import java.io.BufferedReader;
    - OR
    - import java.io.*;

Interactive Input(3)

```java
import java.io.InputStreamReader;
import java.io.BufferedReader;
public class Lab5Example{
    }
```
Interactive Input(4)

- Create a BufferedReader called *stream*…

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

- Call the constructor for BufferedReader in the API.

  ```java
  let's look at the constructor for BufferedReader ... 
  ```

- We need to send it an input stream. So, create an InputStreamReader object:

  ```java
  new InputStreamReader();
  ```

- We see from the API it needs an input stream to connect to. Use System.in (look familiar?)

  ```java
  new InputStreamReader(System.in);
  ```

- Put it all together:

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

Interactive Input(5)

```java
import java.io.InputStreamReader;
import java.io.BufferedReader;

public class Lab5Example{
    public static void main(String[] args)
        throws IOException {
        BufferedReader stream = new BufferedReader(new InputStreamReader(System.in));
    }
}
```

- Note! throws IOException
- What?
- Why?

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…
Interactive Input(7)

```java
import java.io.InputStreamReader;
import java.io.BufferedReader;

public class Lab5Example{
    public static void main(String[] args) throws IOException {
        System.out.print("Please enter your name: ");
        BufferedReader stream = new BufferedReader(new InputStreamReader(System.in));
        String input = stream.readLine();
        System.out.println("Hello "+input);
    }
}
```

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:
    ```java
double d = Double.parseDouble(stream.readLine()).doubleValue();
    ```

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)

```java
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.
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

Basic Exception Handling(2)

```
try {
    <statements>
} catch ( Exceptiontype e1) {
    statements to react and recover
} catch ( Exceptiontype e2) {
    statements to react and recover
} ... etc
```

A calls B
Something unexpected happens. An exception is raised and thrown

What is the Handler?
A “try…catch” block...

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.
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

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

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.