Program Organization and Execution

• All code in Java is organized into classes and interfaces
  – One public class or interface per file.
    • Not strictly true, but accept this as true for now.
    • Also, don’t worry about what an interface is for now.
  – Must match file name.
  – Classes can be organized into “packages”.
    • Again, more on that later.

• Java compiler (javac) produces “byte code”.

• Java interpreter (java) executes program
  – Need to specify a specific class name as the “main class”.
  – Interpreter starts execution with the method “main”
    • Must be defined with the signature:
      public static void main(String[] args)
AverageHeightApp (1/2)

• Open a plain text editor.
• Create a file called AverageHeightApp.java
• Enter the following text:

```java
public class AverageHeightApp {
    public static void main(String[] args) {
        double[] heights = readHeightData();
        double sum_of_heights = 0.0;
        for (int i=0; i<heights.length; i++) {
            sum_of_heights += heights[i];
        }
        double avg_height = sum_of_heights / heights.length;
        System.out.println("The average height is: " + avg_height + " inches");
    }

    static double[] readHeightData() {
        double[] height_data = {66.0, 72.0, 69.5, 68.2, 75.0, 64.5, 63.0};
        return height_data;
    }
}
```
AverageHeightApp (2/2)

• At a terminal command line (console for you Windows people), execute the following commands:
  > javac AverageHeightApp.java
  > java AverageHeightApp
AverageHeightApp Notables

• Demonstrates:
  – Use of javac and java from the command line
  – Appropriate declaration of main
  – Using a for loop to iterate over an array
  – Using System.out.println for console output
    • See reference documentation for System class

  – Automatic type conversion from integer to string
  – Contextual interpretation of + operator
You might want to drop the class if...

• ... you didn’t recognize the idea of an array of values and/or the notion of retrieving a value from the array by its index.
• ... you can’t recognize what in the program is acting as a variable.
• ... you are unsure of how a *for* loop works.
• ... you don’t grock the notion of defining a function and/or calling a function.
• ... you can’t write a program of similar complexity in some other language if not Java.
• The above are necessary conditions for success in this class, but may not be sufficient conditions for success.
  – Trust your own judgment, don’t be afraid of a challenge.
  – Programming takes practice.
AverageHeightApp – take 2

• Same as before, but with Eclipse.
  – Eclipse Workspace
  – Creating new project
  – Creating a new package
  – Creating new class

• More about using Eclipse in recitation next week.
Comments

• Single line comments:
  // This is a comment.

• Multiple line comments:
  
  /*
   All of these lines.
   Are commented.
  */
A Word About Types

Value Types
• Integers
• Real numbers
• Booleans
• Character

Reference Types
• String
• Array
• Objects typed by their class
• Classes themselves

Values types are defined entirely by their value.

Reference types are structures in memory.
• Address in memory uniquely identifies them.
  • The “value” of a variable that holds a reference type is this address.
Value Types

- **Integers**
  - byte, short, int, long
  - Difference is in size (1, 2, 4, or 8 bytes)
  - No “unsigned” version
  - Decimal (255), hexadecimal (0xff), and binary (0b11111111) literal formats

- **Real Numbers**
  - float, double
  - Difference is in precision.

- **Characters**
  - char
  - Characters in Java are 16-bit Unicode values
  - Literals use single-quote: ‘c’
  - Unicode escape sequence: ‘\u####’ where # is hex digit.
    - Example: ‘\u00F1’ for ñ

- **Logical**
  - boolean
  - Literals are true and false
Packages, classes, and methods, oh my!

- A package is a collection of classes.
  - Defined by a “package” statement at the beginning of a source code file.
    - Example:
      ```java
      package lec02.ex01;
      ```
    - All classes defined in the file belong to that package.

- A class is a collection of functions (for now).
  - Defined by “class” keyword followed by a block of code delimited by curly braces.
  - Example:
    ```java
    public class {
      /* Class definition. */
    }
    ```

- A method is just another name for a function or procedure and is a named sequence of Java statements.
  - Defined within a class.
  - Can be “called”, or “invoked” with parameters
  - Syntax: a method header or signature followed by a block of code delimited by curly braces.
Method Signature

• Almost everything you need to know about a method is in its *signature*.
  – 5 parts to a method signature
    • Access modifier
      – public, private, protected
      – If unspecified, then “package” access.
    • Method type
      – static or default (i.e., not specified)
      – The keyword static indicates that this is a “class method”.
      – If the keyword static is not present, then this is an “instance method”.
    • Return type
      – The type of value returned by the method as a result.
      – If there is no result, then this is indicated by the keyword *void*.
    • Method name
      – Must start with a letter, $, or _
      – Can contain letters, numbers, $, or _ (no spaces or other punctuation)
    • Parameter list
      – In parenthesis, comma-separated list of typed parameter names.
        » If the method has no parameters, then just: ()
      – Each parameter variable name is preceded by a type declaration.
Method Signature Examples

public static void main(String[] args)

int foo (int a, MyType b, double c)

protected static void bar() 

static String toUpperCase(String s)

static private Secret my_secret()
Until we know a little more...

• All of the methods in my examples today are going to be public class methods.
  – This means their signatures will include the words:
    • public
    • static
Inside a method

• The body of a method is a sequence of statements.
• A statement ends in a semi-colon
  – Types of statements:
    • Declarations of local variables
    • Assignment
    • Conditional
    • Loop
    • Method call
    • Return statement

• Blocks
  – Zero or more statements enclosed in curly braces { }
  – Allowed anywhere a single statement is allowed.
    • And vice versa
Inside a method

• The body of a method is a sequence of *statements*.
• A statement ends in a semi-colon
  – Types of statements:
    • Declaration of local variables
    • Assignment
    • Conditional
    • Loop
    • Method call
    • Return statement

• Statement Blocks
  – One or more statements enclosed in curly braces { }
  – Allowed anywhere a single statement is allowed.
    • And vice versa
Local variable declaration

• Syntax:
  type name;
  type name1, name2, name3;
  type name = value;

• A local variable is valid within the block of statements where the declaration occurs.
  – This is known as the scope of the variable.
  – The declaration must occur before the variable is used.

• Parameter names are local variables that are in scope for the entire method body.
Variable Names

- Variable names *should* start with a letter and can contain letters, digits, $, or _
  - Cannot start with digit
  - Cannot contain whitespace or punctuation other than $ or _
    - In general, use of punctuation in variable names is discouraged.
  - Case sensitive
  - Cannot be a keyword (e.g. for, while, if, ...)
- Legal:
  - foo, bar, a_variable, var123
- Legal but not considered good:
  - var_with_$, _badness
- Illegal:
  - 1var, while, break, this has whitespace
Inside a method

• The body of a method is a sequence of *statements*.
• A statement ends in a semi-colon
  – Types of statements:
    • Declarations of local variables
    • Assignment
    • Conditional
    • Loop
    • Method call
    • Return statement

• Blocks
  – Zero or more statements enclosed in curly braces `{ }`
  – Allowed anywhere a single statement is allowed.
    • And vice versa
Assignment

• Syntax:
  \[ variable = expression; \]

• Note: single equals for assignment.

• Left hand side must be some sort of variable name that can be assigned.

• Expression must produce a value that matches the type of the variable.
Expressions

• A sequence of symbols that can be evaluated to produce a value.
  – Can be used wherever a value is expected.

• Types of expressions:
  – Literal values: 123, ‘c’, “a string”, true
  – Named value: a_variable
  – Value retrieved from an array: my_array[3]
  – Class/object fields: Math.PI
  – Value as a result of a method call: foo()
  – Compound expression of operators: 4+3*2
Operators

• Arithmetic: +, -, /, *, %
• Relational: ==, !=, >, >=, <, <=
• Boolean: &&, ||, !
• Ternary: ?:
  – expression ? if_true : if_false
• Bitwise: ~, <<, >>, >>>, &, |, ^
• Be aware of precedence
  – Can be controlled by explicitly grouping with ()
• Be aware of context
  – Some operators do different things depending on the types of the values they are applied to.
Assignment and Unary Operators

• Most numeric operators have an “assignment” form.
  – Easy syntax for applying the operator to a variable and assigning the result back to the same variable.
  – Examples:
    • a += 3 // Equivalent to a = a + 3
    • b *= 4 // Equivalent to b = b * 4

• Unary operators ++ and --
  – Used with integer typed variables to increment and decrement.
  – Usually used as a statement unto itself:
    • a++; // Equivalent to a = a + 1;
    • b--; // Equivalent to b = b – 1;
Importing JAR files into a project

• Save JAR file somewhere.
• Create the project in Eclipse if not already extant.
• Right click the src folder in the project and choose, “Import…”
• Choose the type General->Archive and click Next
• Browse for and select the JAR file.
• Click Finish
lec02.ex2.Example2

• Variable declarations for value types
• Integer math vs. Real math
• Ternary operator
• Operator precedence
• Boolean operator shortcut
Inside a method

• The body of a method is a sequence of *statements*.
• A statement ends in a semi-colon
  – Types of statements:
    • Variable declaration
    • Assignment
    • **Conditional**
    • Loop
    • Method call
    • Return statement

• **Blocks**
  – Zero or more statements enclosed in curly braces `{}`
  – Allowed anywhere a single statement is allowed.
    • And vice versa
Conditional Execution: if-else if-else

if (expression) {
    // Block of statements
} else if (expression) {
    // Block of statements
} else {
    // Block of statements
}
if (score > 90) {
    System.out.println("You got an A!");
} else if (score > 80) {
    System.out.println("You got a B.");
} else if (score > 70) {
    System.out.println("You got a C? ");
} else if (score > 60) {
    System.out.println("You got a D :-(");
} else {
    System.out.println("You failed");
}
Conditional Execution: switch

switch (expression) {
    case value:
        statements
        break;
    case value:
        statements
        break;
    ...
    default:
        statements
}

- Works with basic value data types
- Works with String as of Java 7
- Execution starts at first matching case value
  - or default if provided and no case matches
- Continues until break statement or end of switch.
Switch Example

```java
switch (c) {
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
        System.out.println("Vowel");
        break;
    default:
        System.out.println("Consonant");
}
```
lec02.ex3.Example3

• if and switch demo
• Variables scoped within block
• Style hint:
  – Don’t test boolean expression against true/false
• Testing real numbers with epsilon bounds
Inside a method

• The body of a method is a sequence of *statements*.
• A statement ends in a semi-colon
  – Types of statements:
    • Variable declaration
    • Assignment
    • Conditional
    • Loop
    • Method call
    • Return statement
• Blocks
  – Zero or more statements enclosed in curly braces `{ }`
  – Allowed anywhere a single statement is allowed.
    • And vice versa
Loops: while

while (expression) {
    block
}

do {
    block
} while (expression);
```java
while example

int sum = 0;
int n = 1;
while (n < 11) {
    sum += n;
    n++;
}
System.out.println("The sum of 1 to 10 is: " + sum);
```
Loops: for

for (init; test; update) {
    block
}

for example

```java
int sum = 0;
for(int n=1; n<11; n++) {
    sum += n;
}
System.out.println("The sum of 1 to 10 is: "+ sum);
```

• Note that variable \( n \) is declared as part of init expression in for loop.
  – This is a common programming idiom if loop variable is only needed for the loop.
  – Scope of variable is limited to the loop block.
Loop odds and ends

• To skip to next iteration of loop body use “continue” statement.
• To break out of the loop body use “break” statement.
Example 4

- while and for
- while and for equivalence
- scope of for loop variable
- break / continue
Inside a method

• The body of a method is a sequence of *statements*.
• A statement ends in a semi-colon
  – Types of statements:
    • Variable declaration
    • Assignment
    • Conditional
    • Loop
    • Method call
    • Return statement

• Blocks
  – Zero or more statements enclosed in curly braces { }
  – Allowed anywhere a single statement is allowed.
    • And vice versa
Calling Methods

- Calling a class method defined in the same class:
  ```
  methodName(parameters);
  ```

- Calling a class method defined in a different class (same package):
  ```
  ClassName.methodName(parameters);
  ```

- Calling a class method defined in a different package:
  ```
  PackageName.ClassName.methodName(parameters)
  ```

- In the above “parameters” is a comma separated list of values.
  - Must match in number and type according to method’s signature.

- A method call that returns a value (i.e., not a “void” method) can be part of an expression.
  ```
  int max_times_min = max(a, b, c) * min(a, b, c);
  ```
Inside a method

- The body of a method is a sequence of *statements*.
- A statement ends in a semi-colon
  - Types of statements:
    - Variable declaration
    - Assignment
    - Conditional
    - Loop
    - Method call
    - Return statement

- Blocks
  - Zero or more statements enclosed in curly braces `{ }`
  - Allowed anywhere a single statement is allowed.
    - And vice versa
Return

• Syntax:
  
  `return expression;`

• Ends execution of a method and returns the value of the expression as the result of the method.
  
  – Must match type declared in method signature.
  – If method return type is “void”, then simply:
    
    `return;`
• Calling methods
• Compound expressions as part of method call to provide parameter value.
• Returning from middle of method
  – Generally, try to avoid.
• Unreachable code error
• Calling method in same/different class, same/different package
  – lec02.ex5.Example5Other
Import Directive

• Maps class names from other packages into current name space.
  – Convenient if going to use one or more class names repeatedly.

• Map all names from a package:
  
  ```java
  import package.*;
  ```

• Map a specific name from a package:
  
  ```java
  import package.name;
  ```
Example5OtherRevisited

• import
• Math revisited
  – Classes in java.lang package are automatically imported.