Java Basics, cont’d
Strings and Arrays

COMP 401, Spring 2016
Lecture 4
1/21/2016
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);
while example

```java
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.
lec02.ex4.Example4

- 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.
String, our first object

• In Java, a string is an immutable sequence of characters.
  – Strings are objects.

• Objects have a type.
  – The name of the class that defines them.
  – Example: String

• Objects have methods
  – Dereferenced using the “.” operator
  – Example:
    String s = “This is a string”;
    int length = s.length();

• Objects have fields
  – Properties that can be directly accessed as values.
  – Accessed via the “.” operator like methods
    reference.field
Creating Strings

• As a literal.
  – Enclosed in double quotes.
  – Escape sequences for common non-printable or untypeable characters.
    • ", \, \t, \n, \u####

• Using the “new” operator.
  – Generally almost never need to do this.

• As the result of a string concatenation operator

• Lecture 4, Example 1
Useful String methods

• length()
• charAt()
• equals()
• substring()
• trim()
• indexOf()
• Lecture 4, Example 2
Strings are immutable

• Once created, can’t change.
  – Some other languages treat strings as an array of characters. Not Java.

• Any operation that manipulates a string is creating a new string.

• Why immutability?
  – If the same string occurs twice, can simply reuse the same object.
    • This is an optimization that Java performs automatically if it can.
    • It may appear that == can be used to test character-by-character equality, but you should never do that.
      – Always use .equals() method of one string, passing the other as a parameter.
  • Lecture 4, Example 3
Arrays

• Arrays hold an indexed sequence of values
  – Indices start at 0

• Another object type ... with a twist
  – A little different because it is a type that combines with another type.
    • The array structure itself is of type Array, but the type of the individual elements must also be specified.
      – Can’t have an array of different types mixed together.
    – Also different from other objects in its creation syntax.

• Arrays are fixed length.
  – Must be specified when created.
  – Once created, can not be resized.
Creating / Initializing Arrays

• Type indicator for an array is the type name of the individual elements followed by []

• Using the new operator

  \[\text{type}[] \text{ vname} = \text{new} \text{ type}[\text{length}];\]

  – Array will be created, and initialized with default values.
    • For numeric types and char: 0
    • For boolean: false
    • For reference types: null

• Example:

  ```java
  String[] names = new String[3];
names[0] = “Alice”;
names[1] = “Bob”;
names[2] = “Carol”;
  ```
Creating Literal Lists

• When you know the elements in advance.
  – Comma-separated, in curly braces

• Syntax if combined with variable declaration
  ```java
  int[] iarray = {1, 2, 3};
  String[] names = {
      “Abhinandan”,
      “Bhagavateeprasaad”,
      “Chaanakya”
  };
  ```

• Syntax if used to set an existing variable.
  ```java
  iarray = new int[] {4, 5, 6};
  ```
Indexing Arrays

• 0-based indexing
• Length is provided by \textit{length} field
  – Note, for String objects, \textit{length}() was a method
  – Here, \textit{length} is a field
• Size of array can not change once created, but individual elements may change.
• Lecture 4, Example 4
null

- Special value that is always valid for any reference type.
  - Indicates “no value”
  - Any reference type variable can be set to null.
  - Default value for reference type arrays.
Arrays as Reference Types

• Same reference, same array
  – Implication for arrays passed to methods
    • When an array is passed to a method, any changes that the method makes to its elements is permanent.

• Array cloning
  – Easy way to create a “shallow” copy of an array
  – Just call clone() method
    • Result will be a new array of same size with same values or references

• Lecture 4, Example 5
Multidimensional Arrays

- Multidimensional array is simply an array of arrays
  - Fill out dimensions left to right.
    ```java
    int[][] marray = new int[5][];
    for(int i=0; i<5; i++) {
        marray[i] = new int[10];
    }
    ```
- Each subarray can have an independent size.
  - Sometimes known as as a “ragged” or “uneven” array
    ```java
    int[][] marray = new int[5][];
    for (int i=0; i<5; i++) {
        marray[i] = new int[i+1];
    }
    ```
- If each sub-dimension is same size, we can create it with a single `new` statement
  ```java
  int[][] marray = new int[5][10];
  ```
Arrays utility class

- `Arrays` is a library of useful functions for manipulating arrays
  - Note “s” in `Arrays`
  - Like Math class, all methods are static
- `binarySearch`
- `sort`
- `filling and copying subranges`
- [http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html](http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html)
Scanner

- Scanner is a type of object that can be used to parse text input in a variety of ways.
- To create a Scanner object for the console keyboard:
  ```java
  java.util.Scanner s = new java.util.Scanner(System.in);
  ```

- Scanner documentation
  - [http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html](http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html)

- By default, treats input as whitespace separated tokens.
- Parse the next token and...
  - return as String with `next()`
  - interpret as in integer with `nextInt()`
  - interpret as real value with `nextDouble()`

- If you ask for a particular type (e.g., int, double) but the next token can’t be interpreted in that way sensibly, your program dies.
Lecture 4, Example 6

- Uses scanner to read input.
- Expects input to be a number indicating a size and then one of the following words:
  - integer, real, string
- Creates an array of that size of the corresponding type (i.e., int, double, or String)
- Uses a loop to read in that many of the appropriate type into the array.
- Prints the array.
- Does it all over again indefinitely.
Turning In Assignments

• Create a single JAR file.
  – With source code, not compiled classes.

• Open browser to autograder:
  – https://grade.cs.unc.edu/comp401sp16/grades.php
  – MUST be on campus or using VPN

• Autograder runs once an hour
  – 12 minutes past

• For future assignments, feedback will be hidden until after due date.