Basic Java Syntax, cont'd

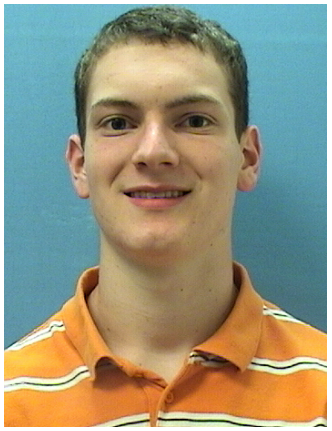
COMP 401, Fall 2014

Lecture 3

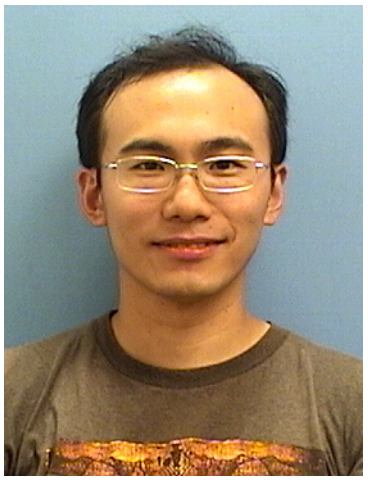
8/26/2014

Our TAs

- Grad Students:
  - Brian Cristante



  - Xufeng Han



UTAs

- Shashank Adepu
- Jade Enns
- Jennifer Hines
- Catherine Jarrett
- Henry Ji
- KJ Moon
- Nate Weatherly







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

```
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

```
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.

Lecture 2 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.
  - A value can be also be an expression that results in a value.
  - 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;`

lec02.ex5.Example5

- Calling methods
- Compound expressions as part of method call to provide parameter value.
- Returning from middle of method
- Unreachable code error
- Calling method in same/different class, same/ different package

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:

```
import package.*;
```

- Map a specific name from a package:

```
import package.name;
```

Example5OtherRevisited

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

Java Execution Model

- Your program is always executing within the context of some method.
  - Starts off in the "main" class method defined in whatever class you told the program to start with.
  - Execution context includes:
    - Local variable names that are in scope.
    - Parameter names.
      - Act just like local variables.

Java Execution Model

- When you call a method:
  - Current context is saved.
  - A new context is created.
    - Method parameter names are mapped to values provided when method was called.
    - Local variables in scope in the called method.
- When a method returns:
  - Current context is destroyed.
  - Context where method call occurred is restored.
  - Value returned is substituted as result of method call.
- Lecture 03, Example 1
- This is why recursion works.
  - Lecture 03, Example 2

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 3, Example 3

Useful String methods

- length()
- charAt()
- equals()
- substring()
- trim()
- indexOf()
- Lecture 3, Example 4

Strings are immutable

- Once created, can't change.
  - Some other languages treat strings as a type of 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.
    - Side effect of this is that it may *appear* that == can be used to test equality, but you should not do that.
    - Lecture 3, Example 5

Arrays

- Another object type.
  - A little different because it is a type that combines with another type.
    - The array 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 type name of individual elements followed by []
- As a literal list of values.
  - Comma-separated, in curly braces
  - Example:

```
int[] iarray = {1, 2, 3};
String[] sarray = {“one”, “two”, “three”};
```

- Using the new operator

type`[] vname = new` *type*`[`*length*`];`

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

Indexing Arrays

- 0-based indexing
- Length is provided by *length* field
  - Note, for String objects, length() was a method
  - Here, length is a field
- Size of array can not change once created, but individual elements may change.
- Lecture 3, Example 6

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.

Assignment 1