Java Over/Review (Crash Course)

Instructors: Sam McCauley and Dan Barowy

February 7, 2022

Part I: Hello Java!

Hello World

```java
/*
 * Hello.java
 * Author: CS 136 staff
 * Spring 2022
 * Prints a welcome message to the terminal
 */
public class Hello {
 public static void main(String[] args) {
    System.out.println("Hello, CS136!") ;
 }
}
```

Running Java Code

- Edit/Compile/Run cycle
- Edit: Save Java source code in file Hello.java
- Compile: javac Hello.java
  - Produces Java *bytecode* file named Hello.class
- Execute: java Hello
  - Searches Hello.class for a method with *signature*
  - `public static void main(String[])`
  - Executes that method (if it exists)

Notes on Java Syntax

- Multi-line comments: `/*` and `*/`
- Single-line comments: `//`
- Code is wrapped in a *class declaration*
  - Everything is (in) a class in Java
  - File name must be same as declared class name
  - System is a Java class holding an object called out

hello syntax

```
/*
 * Hello.java
 * Author: CS 136 staff
 * Fall 2020
 * Prints a welcome message to the
     terminal
 */
public class Hello {
 public static void main(String[]
     args) {
   System.out.println("Hello,
       CS136!");
 }
}
```

- The parameter `args` is an array of `String`
  - Passed to the main method from the *command line*
  - Contains every string on the command line after java Hello
- This allows passing values into program
- args can be replaced with any other variable name…

hello syntax

```
/*
 * Hello.java
 * Author: CS 136 staff
 * Fall 2020
 * Prints a welcome message to the
     terminal
 */
public class Hello {
 public static void main(String[]
     args) {
    System.out.println("Hello,
        CS136!") ;
 }
}
```

- `System` is a Java class holding an object called `out`
- out is of class type `PrintStream`
  - PrintStreams provide many methods, including print() and println()
  - Don't need to know `PrintStream` details until later in the course

More about strings in Java

- Every array has an associated variable (instance variable) called length, which holds the size of the array
- Array indexing, as in C and Python, starts at `0`
- String, unlike int, is a class-based type, not a primitive type
    - More on this soon.…

Part II: Primitive Types, Array Types, Operators and Expressions, Control Structures

Primitive Types

- Provide numeric, character, and logical values
  - 11, -23, 4.21, 'c', false
- Can be associated with a name (*variable*)
- Variables *must* be *declared* before use
  - `int age; // A simple integer value`
  - `float speed; // A number with a 'decimal' part`
  - `char grade; // A single character`
  - `bool loggedIn; // Either true or false`

Primitive Types Cont.

- Variables *can* be *initialized* when declared
  - int age = 21;
  - float speed = 47.25;
  - char grade = ‘A’;
  - bool loggedIn = true;

Default values (Not to be used on purpose!)

- Uninitialized *instance variables* (variables that describe features of an object) of primitive type are given default values
- int age; // Initialized to `0`
- float speed; // Initialized to `0.0`
- char grade; // Initialized to `\u0000` (Unicode)
- bool loggedIn; // Initialized to false
- Uninitialized local variables declared in a method are *not* given default values
    - Always initialize a local variable when you declare it!

Strings and Primitive Types

- Often numeric data is made available as a string
- Consider a class Sum which adds two numbers provided on the command line
  - `java Sum 3 5`, for example, would return 8
- 3 and 5 are held as String values in `args[]`
- The values can be converted to int values using the `parseInt()` method of the class Integer
  - `int num1 =  Integer.parseInt(args[0]);`

Sum

```java
public class Sum {
   public static void main(String[] args) {
      if ( args.length < 2 )
         System.out.println( "Syntax: java Sum num1 num2" );
      else {
         int n0 = Integer.parseInt( args[0] );
         int n1 = Integer.parseInt( args[1] );
         System.out.println(n0 + " + " + n1 + " = " + (n0 + n1));
      }
   }
}
```

Array

- Holds a collection of values of some type
- Can be of any type
  - int[] ages; // An array of integeras
  - float[] speeds; // An array of floats
  - char[] grades; // An array of characters
  - bool[] loggedIn; // Either true or false

Array

- Arrays can be initialized when declared
  - `int[] ages = { 21, 20, 19, 19, 20 };`
  - `float[] speeds = { 47.25, 3.4, -2.13, 0.0 };`
  - `char[] grades = { 'A', 'B', 'C', 'D' };`
  - `bool[] loggedIn = { true, true, false, true };`
- Or just created with a standard default value
  - int[] ages = new int[15]; // array of 15 0s

Notes on Arrays

- Arrays are not primitive types in Java, they are class types (an array is therefore an *object* in Java)
- As a result, an uninitialized array holds the special object value null. This means
  - It is an error to attempt to index into an uninitialized array
    - int[] scores; // Uninitialized array
    - scores[0] = 100; // Error!
  - It is an error to access any instance variable or method of an uninitialized array
    - int size = scores.length; // Error!

Example: Rolling a Die

```
import java.util.Random; // importing an external class
public class DieRoller {
  public static void main(String[] args) {
    // A random number generator
    Random rng = new Random();
    int faces = Integer.parseInt(args[0]);
    int[] counts = new int[faces]; // initialized to 0s
    int numRolls = 100*faces;    // number of tests
    // generate numRolls random values in range 0..faces-1
    for (int i = 0; i < numRolls; i++)
      counts[rng.nextInt(faces)]++;
    for (int i = 0; i < faces; i++)
      System.out.println(""+ i + ": " + counts[i]);
  }
}
```

Reminder : Importing Classes

- We've imported things like `java.util.Scanner` and `java.util.Random`
- The Java distribution has a variety of useful classes
- To use such a class, you must import it
- Unless it is in the directory of your program
- To do this, use import with the package name
- Examples
  - import java.util.Scanner;
  - import java.util.Random;
  - import structure5.*; // entire package

Operators

Java provides a number of *operators* including:

- Arithmetic operators: +, -, *, /, %
- Relational operators: ==, !=, <, <=, >, >=
- Logical operators &&, || (don't use &, |)
- Assignment operators =, +=, -=, *=, /=, ...
- Common unary operators include:
  - Arithmetic: - (prefix); ++, -- (prefix and postfix), Logical: ! (not)

Operator Gotchas!

- There is no exponentiation operator in Java.
  - The symbol `^` is the *bitwise xor* operator in Java.
  - Use something like `x * x` to calculate the square of `x`
- The *remainder* operator % is the same as the mathematical 'mod' function for *positive* arguments,
  - For **negative** arguments **it is not** : -8 % 3 = -2
- The logical operators && and || use *short-circuit evaluation* :
  - Once the value of the logical expression can be determined, no further evaluation takes place.
  - E.g.: If n is 0, then `(n != 0) && (k/n > 3)` will yield false without evaluating k/n. Very useful!

Example

```java
/* QuotientRemainder.java
 * Prints the quotient and remainder given positive integers
 * java QuotientRemainder 23 4 returns Q = 5, R = 3
 */
public class QuotientRemainder {

  public static void main(String[] args) {
    if(args.length != 2 )
      System.out.println("Usage: java QuotientRemainder int int ");
    else {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      System.out.println("Q = " + a/b + ", R = " + a % b);
    }
  }
}
```

Expressions

Expressions are either:

- literals, variables, invocations of non-void methods, or
- statements formed by applying operators to them
- An expression returns a value:
  - `3 + 2 * 5 - 7 / 4 // returns 12`
  - `x + y * z - q / w`
  - `(-b + Math.sqrt(b * b - 4 * a * c) )/(2 * a)`
  - `( n > 0) && (k / n > 2) // computes a Boolean`

Operator Precedence in Java

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Expressions

- Assignment operator also forms an expression
  - `x = 3; // assigns x the value 3 and returns 3`
  - So `y = 4 * (x = 3)` sets x = 3 and y = 12 (and returns 12)
  - Boolean expressions let us control program *flow of execution* when combined with *control structures*
  - Example
    - `if ( (x < 5) && (y !=0 ) ) {...}`
    - `while (! loggedIn) { ... }`

Two versions of a loop

```
Random rng = new Random();
int flip = rng.nextInt(2);
int count = 1;
while (flip == 0) {
   //count flips until "heads"
   flip = rng.nextInt(2);
   count++;
}
```

```
Random rng = new Random();
int flip = rng.nextInt(2);
for(int count=1; flip==0; count++){
   flip = rng.nextInt(2);
}
```

One more version of the loop

```
Random rng = new Random();
int flip = rng.nextInt(2), count =
    1;
while (flip == 0) {
   // count flips until "heads"
   flip = rng.nextInt(2);
   count++;
}
```

```
int flip, count = 0;
do {
   //count flips until "heads"
    flip = rng.nextInt(2);
   count++;
} while (flip == 0);
```