Java Programming 2 – Lecture #4 – Jeremy.Singer@glasgow.ac.uk

Type Conversions

Since Java is a statically typed language[1], a declared variable can only hold values of a single, specific type. In order to store a value of type t_1 in a variable of type t_2, the value must be converted to type t_2 before the assignment occurs. Some type conversions are implicit, i.e. the programmer does not need to indicate the conversion in the source code. These are generally *widening* conversions, where little or no information is lost. Example widening conversions include `byte` to `long`, or `int` to `double`.

When a type conversion would result in significant potential loss of information, e.g. `double` to `float` or `int` to `short`, this is known as a *narrowing* conversion. In such cases, the conversion must be made explicit using a type-cast operator which specifies the target type in brackets. For example:

```
int i = 1025;
byte b = (byte)i;   // b has value 1
```

Floating-point to integer type conversions use the *round to zero* convention if the floating-point value is in the representable range of the target integer type. For –ve and +ve numbers that are too large in magnitude to represent, the `MIN_VALUE` or the `MAX_VALUE` of the integer type is selected, respectively. The Java Language Specification gives full details of the type conversion rules[2].

It is worth mentioning one other kind of conversion, from `String` objects to primitive values. Each of the primitive wrapper classes has a static method to convert from a String to a primitive value of that type. For instance, `Integer.parseInt(“42”)` will return the value 42 of type `int`. Consider the program below, which takes a sequence of integers from the program arguments and sums the positive integers until a 0 value appears, or there are no more arguments.

```
public static void main(String [] args) {
  int i, sum = 0;
  try {
    for (i=0; i<args.length; i++) {
      int value = Integer.parseInt(args[i]);
      if (value==0) break;
      sum += value;
    }
    System.out.printf(“Sum of first %d args is %d\n”, i, sum);
  }
  catch (NumberFormatException e) {
    // error in parsing
  }
}
```

[1] In contrast to dynamically typed languages, such as Python, Ruby and Javascript.
[2] http://docs.oracle.com/javase/specs/jls/se7/html/jls-5.html

Constant Values

The Java `final` modifier indicates that the relevant entity (for now, just class variables and local variables) is a constant. Constant class variables are useful values for general calculations, for example `Math.E` and `Math.PI`. (Note that constant class variables generally have all-caps identifiers.) Constant local variables are useful to indicate values that should not change after their initial assignment, e.g. the length of an array or a `String` can be stored in a `final` variable. The use of `final` is encouraged[3] because it makes source code easier to read and also to optimize.

Math Library Methods

The Java Math library has some useful static methods for numeric calculations. These include trigonometric functions like `Math.sin()`, simple utility functions like `Math.pow()`, etc. Check out the full documentation at http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html. Note that most of these methods operate on double values.

One particularly useful method is `Math.random()` which returns a pseudo-random `double` value, in the range [0.0, 1.0). Values are uniformly distributed in this range. So, to get an `int` value in the range [1,100], the code would look like:

```
    int i = (int)(1 + Math.random()*100);
```

Fun Task

Write a Java program that computes a 'secret' random number in the range 1 to 20. The program prompts the user for guess values. For each guess, the program outputs HIGHER, LOWER or CORRECT. Until the guess is CORRECT, the user is prompted for another guess.

Hints: use the random number generation method above. Use `Scanner.nextInt()` on `System.in` to acquire user input. Use `System.out.println()` to output messages to the user. A `while` loop (or a `do/while`) is appropriate for the control flow.

Question

1) Should method parameters be marked as `final`? If so, why? If not, why not?

[3] http://www.javapractices.com/topic/TopicAction.do?Id=23