Computer Science 225
Advanced Programming
Siena College
Spring 2022

Topic Notes: Java You Know

We begin by reminding you about the Java and general programming concepts and constructs you already know.

Java/Programming Basics

By now, these topics should be second nature.

- sequential execution
- variables and primitive data types
- evaluation of arithmetic and boolean expressions
- basic `String` manipulations (*e.g.*, `length`, concatenation, `substring`, `split`, `charAt`, `toLowerCase`, `toUpperCase`, `startsWith`, `trim`)
- basic type conversions (`DecimalFormat`, `Integer.parseInt()`, `Double.parseDouble()`)
- `equals` *vs.* `==`
- conditional execution (`if`, `if` .. `else`)
- repetition (`while`, `for`, `do` .. `while`)
- methods, both writing and calling, including parameter passing and returning values
- keyboard/terminal I/O (`java.util.Scanner`s on `System.in`, `System.out`)
- basic file I/O (`java.util.Scanner`s on a `java.io.File`, `java.io.PrintWriter`s)
- random number generation (`java.util.Random` class)
- arrays
- basics of defining classes (instance variables, constructors, methods)

Intermediate Java/Programming

Let's take a look at a few things you likely learned late in your introductory Java course or in data structures. It's ok if you missed or have forgotten a few.

- two-dimensional arrays

```
int a[][] = new int[10][15];

for (int row = 0; row < 10; row++) {
    for (int col = 0; col < 15; col++) {
        a[row][col] = row*15+col;
    }
}
```

- *Javadoc*
  - begin with `/**`
  - class comment, including description, `@author` and `@version` tags
  - each method and constructor has a method description, and all appropriate `@param` and `@return` tags
  - generate API documentation with the `javadoc` command
- `static` (variables, methods)
  - a *class method*, denoted by the `static` qualifier, does not have access to the *instance variables* of its class, so can be called even if no object of that class has been instantiated
  - this is in contrast to an *instance method*, which does not have the `static` qualifier, and does have access to the instance variables of the same instance of the class on which the method is called
  - instance methods have an implicit `this` reference parameter that lets them find the correct set of instance variables
  - a class method does have access to the *class variables* of its class, which are also denoted by the `static` qualifier
  - exactly one copy of each class variable exists, no matter how many instances of the class are instantiated
  - the single copy of the class variables is accessible to all class and instance methods
  - see the `StaticStuff` example
- all classes have `equals` and `toString` methods that we can replace with customized versions
- `switch` statements
- *enhanced* `for` *loop*, sometimes called a "foreach loop"
- interfaces

  - API examples: `java.util.Iterator`, `Iterable`, `Comparable`
  - more on interfaces will be one of our first major topics

- exceptions (`try` .. `catch`, `throw`, `throws`)
  - very flexible and powerful mechanism for error handling, we will see and implement examples
- recursion
- complexity (Big O basics)
- timing (`System.currentTimeMillis`, `System.nanoTime`)

  To see how much time a code block takes to execute:

```
long startTime = System.currentTimeMillis();
// code to be timed goes here
long endTime = System.currentTimeMillis();

System.out.println("Code took " + (endTime-startTime) + " ms");
```

- `StringBuffer` and `StringBuilder`
  - see the `BuildingStrings` example
- *ternary operator*
  - this code:

    ```
    expr ? iftrue : iffalse
    ```

    will evaluate to whatever we put in place of `iftrue` if the boolean expression `expr` evaulates to `true`, and will evaulate to what's in `iffalse` otherwise
  - it can be used to eliminate some `if`...`else` conditionals

Fundamental Data Structures and Related Topics

Next, let's recall some core data structures material you know and love.

In Data Structures, you likely implemented or studied implementations of many of these. Mentioned below are some of the Java API classes and interfaces that provide the given functionality.

- wrapper classes, autoboxing, autounboxing
- lists (`java.util.List`)

  - array-based (`java.util.ArrayList`)
  - singly-linked, doubly-linked (`java.util.LinkedList`), circular

- generic structures/type parameters
- linear structures: stacks, queues (`java.util.Queue` implementations), deques (`java.util.Deque` implementations)
- restricted data structures and why they are beneficial
- ordered structures
  - linear vs. binary search
- maps/dictionaries
- binary trees
  - binary search trees
  - balanced BSTs (`java.util.TreeMap`)
- heaps
- priority queues (`java.util.PriorityQueue`)
- hashing (`java.util.HashMap`)

Other Java API classes of interest:

- `String` (it probably does more than you think)
- `java.util.Arrays`