Java Review

Thursday, August 20, 2015 11:35 AM

Daniel Pade

cse.sc.edu/~pade/csce146

For those w/6 an account

Go to 1D35

Ryan Austin's office

Labs	10 pts
HWs	20 pts
Exams	100 pts
Quizzes	10 pts

Outline

Java Review

Arrays

Lists (Array & Linked) ← Stacks & Queues

Big O notation

Sorting & searching

Heaps (and Heapsort)

Binary search trees

Time provided

AVL Trees

Graphs

djpade@gmail.com

Duncan Buell

Data structures using Java

Java Review

Variables

Declaration

```
int foo;
/* Guidelines
   no single letter variables
   declare one at a time
*/
```

Initialization

```
int bar = 2;
```

Basic types

- int, double, boolean, char
- String

Boolean Operations

!, &&, ||

Control flow

Conditional Execution (if statement)

```
if ( boolean) { //same line
    body
} else if ( boolean ) {
    :
} else {
    :
}
```

Loops

- While

```
while ( boolean) { //same line
    :
}
```

- do while

```
do {
    :
} while (boolean);
//always executes at least once
```

- for loop

```
for (initialization; condition; update) {
    :
}
```

- for each loop

```
for ( var : iterable ) {
    :
}
```

```
int[] numbers = {1, 2, 3, 4};
:
for (int number : numbers) {
```

```
        System.out.println(number);
    }
```

1
2
3
4

- switch statement

```
int foo;
  :
switch (foo) {
    case 0:
        ~~
        break; // put this here!
    case 1:
        :
        break;
    :
    case 5:
        ~~
        ~~
        break;
    default: //need a default
        ~~
        break;
}
```

```
if (foo == 1) {
  . . .
} else if (foo == 2) {
```

~~if (month.equals("January")) {~~

```
if (month == 1) {
    .
    .
    .
}
switch (month) {
    case 1: // Jan
```

Methods

a) static

"global", "stateless"

Math.sin(...)

System.exit(...)

(Class name).method(...)

b) non-static

"converse w/ an object"

```
String name = "Daniel";
name.length(); // name, give me your length
   static
(System.out).println(...);
// out, print ...
Scanner scan = new Scanner(...)
scan.next(); // scan, give me the next token
```

(Class instance).method()

Classes

groups data & behaviors

- Define member variables
- Constructor (instantiate member vars)
- define the methods

e.g. Triangle

```
side1()
side2()
side3()
coord1();
  " 2();
  " 3();
```

```
class Triangle {
    Point p1
    .
    .
    .
}
```

```
class Triangle {
    int side1;
    int side2;
```

Bad / Simple (Triangle)

```
public class Triangle {
    public static void main(String[] args) {
        int side1 = 3;
        int side2 = 3;
        int side3 = 3;
        int perimeter = side1 + side2 + side3;
        System.out.println(perimeter);
    }
}
```

Better Example (Triangle)

```
public class Triangle {
```

```
private int side1;
private int side2;
private int side3;
public Triangle (int side1, int side2, int side3) {
    if ( side1 <= 0 || side2 <=0 || side3 <=0) {
            System.exit (1);
    }
    : // sorts the values into mid, min, max
    if ( min + mid < max ) { // check triangle
            System. exit(1)  // inequality
    }
    this.side1 = side1;
    this.side2 = side2;
    this.side3 = side3;
}
public int perimeter ( ) {
    return side1 + side2 + side3;
}
}
public class SomeOtherClass {
    public static void main (String[] args) {
        Triangle tri = new Triangle (3,3,3);
        System.out.println(tri.perimeter());
    }
}
```

Internal Classes

old:

```
class Triangle
class Point
```

new:

```
class Triangle
  class Point
```

```
public class Triangle {
    private Point p1;
    private Point p2;
    private Point p3;
    private static class Point {
        private int xCoord;
        private int yCoord;
    }
    public Triangle (int side1, int side2, int side3) {
        // Check validity
        p1 = new Point ();
        p1.xCoord = 0;
        p1.yCoord = 0;
```

```
        p2 = new Point();
        p2.xCoord = side1;
        p2.yCoord = 0;
        :
    }
}
```

Exceptions

2 kinds:

- Runtime exceptions
- Compiletime exceptions

Both kinds extend from 'Exception' (java.lang.Exception)

Runtime extends from 'RuntimeException'

Runtime Exceptions do not need an explicit try/catch

```
try {
    arr[-1];
catch (ArrayIndexOutOfBoundsException e) {
        println(e.message());
        :
}
```

File I/O

To read in → use a Scanner

To print → use a PrintWriter

reading a file:

```
import java.io.File;                    // FileNotFound... // listed lexicographically
import java.util.Scanner;
:
Scanner fscan = null;
try {
    fscan = new Scanner(new File("name.txt"));
} catch (FileNotFoundException e) {
```

```
}
while ( fscan.hasNextLine() ) {
    println ( fscan.nextLine() );
}
```

Printing to a file

```
import java.io.FileNotFoundException
import java.io.PrintWriter;
:
PrintWriter fout = null
try {
    fout = new PrintWriter ("out.txt");
} catch (FileNotFoundException e) {
    :
}
fout.println("Hello, filesystem!");
```

Throwing exceptions (and when to do it)

```
"Daniel".indexOf('a'); //1
"Daniel".indexOf('z'); //-1

"Daniel".charAt(90);
```

String {

```
public char charAt(int index)
    throws StringIndexOutOfBoundsException {
    if (index < 0 || index > size) {
        throw new StringIndexOut...
    }
```

Recursion

A loop construct

A quick example: Factorial

$3! = 3 \cdot 2 \cdot 1 = 6$

$$x! = \begin{cases} x \cdot (x-1)! & \\ 1 & x = 0 \text{ or } x = 1 \end{cases}$$

$$fact(x) = \begin{cases} 1 & x = 0 \text{ or } x = 1 \\ x \cdot fact(x-1) & \text{otherwise} \end{cases}$$

```
public static int factorial (int num) {
    if (num <= 1) {
        return 1;
    }
    return num * factorial (num-1);
}
```

base case

recursive case

```
public static int factorial (int num) {
    int product = 1;
    for (; num >= 1; num--) {
        product *= num;
    }
    return product;
}
```