Introduction to Programming

by Ultan

Introduction

Notes from the course, Introduction to Programming I took. These are notes that are relevant for a general software engineering position.

Other courses, from my degree, that are relevant for a general, entry level software engineering position are Advanced Telecommunications, Algorithms and Data Structures, Information Management II and Software Engineering. Notes for these courses are also included in ../.

These summary notes are based mainly on content from the course, Introduction to Programming, but also from the textbook Introduction to Programming Using Java by David J. Eck, linked here (license).

Coding Standard

1. All code should be indented
2. No line of code should be longer than the screen and page width
3. No function should be longer than one page in length where possible
4. Duplication of code should be avoided
5. Function parameters should not be used as local variables. Function parameters should only be altered when they are being used to return results
6. The break statement should only be used within a switch statement
7. Constant values should be declared as constants using the final declaration
8. All name (for variables, constants, etc.) must be self explanatory
9. Name conventions:
    1. Constants should be all capitals with underscores
    2. Local variables should be in lowerCamelCase
    3. Function name and parameters should be in lowerCamelCase

Imperative Programming

Hello World

```
import java.util.Scanner;

public class HelloName {

    public static void main(String[] args) {

        System.out.print("What is your name?");
        Scanner input = new Scanner(System.in);
        String name = input.next();
        System.out.println("Hello " + name);
```

```
    }

}
```

Variables, Types, Assignment, Precedence

```
import java.util.Scanner;

import javax.swing.JOptionPane;

public class TriangleArea {

    public static void main(String[] args) {

        String input = JOptionPane.showInputDialog("Enter the coordinates:");
        Scanner inputScanner = new Scanner(input);
        double x1 = inputScanner.nextDouble();
        double y1 = inputScanner.nextDouble();
        double x2 = inputScanner.nextDouble();
        double y2 = inputScanner.nextDouble();
        double x3 = inputScanner.nextDouble();
        double y3 = inputScanner.nextDouble();
        inputScanner.close();

        double area = Math.abs( (x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2)) / 2.0 );

        JOptionPane.showMessageDialog(null, "The area of triangle ((" +
                x1 + "," + y1 + "), (" + x2 + "," + y2 + "), (" + x3 +
                "," + y3 + ")) is " + area );
    }

}
```

If Statements

```
import java.util.Scanner;

import javax.swing.JOptionPane;

/*
 * Write a program to determine is the year entered is a leap year or not.  A year
 * is a leap year if it is divisible by 4  (e.g. the year 2012 is a leap year)
 * unless it is divisible by 100 (e.g. the year 2100 is not a leap year) unless it
 * is divisible by 400 (e.g. the year 2000 is a leap year).
 */
public class LeapYear {

    public static void main(String[] args) {

        String input = JOptionPane.showInputDialog("Enter year:");
        Scanner scanner = new Scanner(input);
        int year = scanner.nextInt();

        boolean leapYear = false;
        if (year % 400 == 0)
        {
```

```
            leapYear = true;
        }
        else if (year % 100 == 0)
        {
            //leapYear = false;
        }
        else if (year % 4 == 0)
        {
            leapYear = true;
        }
        //else
        //{
        //  leapYear = false;
        //}

        JOptionPane.showMessageDialog(null, "" + year + " is " +
                    (leapYear ? "" : "not ") + "a leap year.");

    }

}
```

For Loops

```
import java.util.Scanner;
import javax.swing.JOptionPane;

/*
 * Write a program to calculate factorial of some number (i.e. number!).
 * For example 4! = 1*2*3*4 = 24
 */
public class Factorial {

    public static void main(String[] args) {

        String input = JOptionPane.showInputDialog("Enter an integer to compute its fact
        Scanner scanner = new Scanner(input);
        int number = scanner.nextInt();

        int factorial = 1;
        for (int i = 2; (i <= number); i++)
        {
            factorial = factorial * i;
        }

        JOptionPane.showMessageDialog(null, "The factorial of " + number +
                                        " is " + factorial );

    }

}

import java.util.Random;
import javax.swing.JOptionPane;

/*
 * Write a program to simulate the toss of a coin.  For 10,000 tosses
 * determine how many heads and how many tails are tossed.  Also indicate what the
```

```
 * last toss was (tails or heads).
 *
 */
public class TossCoin {

    public static final int NUMBER_OF_THROWS = 10000;
    public static final int HEADS = 1;
    public static final int TAILS = 0;

    public static void main(String[] args) {

        boolean headsShownOnCoin = false;
        int headsCount = 0;
        Random  generator = new Random();

        for (int throwCount=0; throwCount < NUMBER_OF_THROWS; throwCount++)
        {
            headsShownOnCoin = (generator.nextInt(2) == HEADS);
            headsCount += (headsShownOnCoin) ? 1 : 0;
        }

        JOptionPane.showMessageDialog(null, "Heads: " + headsCount +
                "\nTails: " + (NUMBER_OF_THROWS-headsCount) +
                "\nLast: " + ((headsShownOnCoin) ? "Heads" : "Tails"));

    }

}
```

While and Do-While Loops

```
import java.util.Scanner;
import javax.swing.JOptionPane;

/*
 * Write a program to reverse the order of the digits in a user
 * supplied integer.  For example if the user entered 395 the
 * system would output 593.
 */
public class ReverseDigits {

    public static void main(String[] args) {

        String input = JOptionPane.showInputDialog("Enter number to reverse:");
        Scanner scanner = new Scanner(input);
        int number = scanner.nextInt();

        int reversedNumber = 0;
        int remainingNumber = number;
        while (remainingNumber > 0)
        {
          reversedNumber = reversedNumber*10 +
                     remainingNumber%10;
          remainingNumber = remainingNumber/10;
        }

        JOptionPane.showMessageDialog(null, "The reverse of " + number + " is " + revers
    }
```

```
}
```

Exception Handling

```
import java.util.Scanner;

public class SimpleExceptionHandling {

    public static void main(String[] args) {
        try
        {
            System.out.print("Enter a number: ");
            Scanner inputScanner = new Scanner( System.in );
            int number = inputScanner.nextInt();
            int result = 100 / number;
            System.out.println( "Result is " + result );
        }
        catch (ArithmeticException exception)
        {
            System.err.printf ( "\nException thrown: %s\n", exception );
        }
    }

}
```

Switch Statements

```
switch (currentNumber%10)
{
   case 1:
       outputString+="st";
       break;
   case 2:
       outputString+="nd";
       break;
   case 3:
       outputString+="rd";
       break;
   default:
       outputString+="th";
       break;
}
```

Functions

```
import java.util.Scanner;
import javax.swing.JOptionPane;

/*
 * Write a program which, given two integers (from the user) will compute the
 * Greatest Common Divisor (GCD) of those numbers.  As part of your solution
 * write and use the following functions:
 *  - getGreatestCommonDivisor which takes two integers and returns their
 *    greatest common divisor.
 *  - getNextDivisor() which takes a number and a divisor (of that number)
 *    and returns the next highest divisor of the number.  If there is no
```

```
 *    such divisor -1 is returned.
 * Ensure that you handle all possible errors.
 */
public class GCD {

    public static void main(String[] args) {

        try
        {
            String input = JOptionPane.showInputDialog("Enter first number:");
            Scanner scanner = new Scanner(input);
            int number1 = scanner.nextInt();
            input = JOptionPane.showInputDialog("Enter second number:");
            scanner = new Scanner( input );
            int number2 = scanner.nextInt();
            if ((number1 <= 0) || (number2 <= 0))
                throw new NumberFormatException();

            int gcd = getGreatestCommonDivisor ( number1, number2 );

            JOptionPane.showMessageDialog(null, "The GCD of " + number1 +
                    " and " + number2 + " is " + gcd);
        }
        catch (NullPointerException exception)
        {
        }
        catch (java.util.NoSuchElementException exception)
        {
            JOptionPane.showMessageDialog(null, "No number entered.",
                    "Error", JOptionPane.ERROR_MESSAGE);
        }
        catch (NumberFormatException exception)
        {
            JOptionPane.showMessageDialog(null, "Numbers must be greater than 0.",
                    "Error", JOptionPane.ERROR_MESSAGE);
        }
    }


    // Returns largest number which is evenly divisible into two provided numbers.
    public static int getGreatestCommonDivisor ( int number1, int number2 )
    {
        int gcd = 1;
        int divisor = 1;
        while ((divisor != -1) && (divisor < number2))
        {
            if (number2%divisor == 0)
            {
                gcd = divisor;
            }
            divisor = getNextDivisor( divisor, number1 );
        }
        return gcd;
    }

    // Given a number and a divisor, find and return the next highest
    // divisor of the number.  If there is no such divisor return -1.
    public static int getNextDivisor ( int lastDivisor, int number )
```

```
    {

        if ((lastDivisor > 0) && (lastDivisor < number) &&
                (number % lastDivisor == 0))
        {
            int divisor = lastDivisor;
            do {
                divisor++;
            } while ((divisor < number) && (number % divisor != 0));
            if (number % divisor == 0)
            {
                return divisor;
            }
        }
        return -1;
    }
}
```

Recursion

Fibonacci

```
import java.util.Scanner;

import javax.swing.JOptionPane;

/*
 * Write a program to compute a particular term in the Fibonacci sequence.
 *
 *  Fibonacci( 1 ) = 0
 *  Fibonacci( 2 ) = 1
 *  Fibonacci( n ) = Fibonacci( n-1 ) + Fibonacci( n-2 )    where n > 2
 */
public class Fibonacci {

    public static final int FIBONACCI_NUMBER_1 = 0;
    public static final int FIBONACCI_NUMBER_2 = 1;

    public static void main(String[] args) {

        String input = JOptionPane.showInputDialog("Enter index of Fibonacci #:");
        Scanner scanner = new Scanner( input );
        int index = scanner.nextInt();

        int fibonacciNumber = ComputeFibonacciNumber( index );

        JOptionPane.showMessageDialog(null, "Fibonacci_number( " + index +
                " ) is " + fibonacciNumber );

    }

    public static int ComputeFibonacciNumber( int index )
    {
        int result;
        switch (index)
        {
            case 1:
                result = FIBONACCI_NUMBER_1;
```

```
                break;
            case 2:
                result = FIBONACCI_NUMBER_2;
                break;
            default:
                result = ComputeFibonacciNumber( index-1 ) +
                                ComputeFibonacciNumber(index-2);
                break;
        }
        return result;
    }
}
```

Factorial

```
// An example of divide and conquer
int fac(int n) {
    if (n == 0)
        return 1;
    else
        return n * fac(n-1);
}
```

1D Arrays

```
double[] result = new double[vector1.length];
```

2D Arrays

```
char[][] board = new char[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
```

ArrayList

- A parameterised type. This can take a type parameter

```
ArrayList<String> myList;
myList = new ArrayList<String>();
myList.size(); myList.add(obj); myList.get(N);
myList.set(N,obj); myList.clear(); myList.remove(N); myList.remove(obj); myList.indexOf
```

Using Data Types

A Foundation for Programming

1. Primitive data types and assignment statements
2. Math and text I/O
3. Conditionals and loops
4. Arrays
5. Graphics, sound and image I/O
6. Functions and modules
7. Objects

Data Type

- Set of values and operations on those values
- Primitive types are operations that translate directly to machine instructions
  - boolean - true, false, not, and, or, xor
  - int - -2^{31} to $2^{31} - 1$, add, subtract, multiply
  - double - any of 2^{64} possible reals, add, subtract, multiply
  - char (single quotes), byte, short, long, float
- Non-primitive for colours, pictures etc.
  - Integer, Double, String (double quotes)

Objects

An object holds a data type value. A variable name refers to the object. This enables us to create our own data types and define operations on them.

In Java, programs manipulate references to objects with the exception of primitive types.

Constructors and Methods

```
// Declare a variable (object name)
String s;

// Call a constructor to create an object
s = new String("Hello, World!");

// Call a method that operates on the object's value
System.out.println(s.substring(0,5));
```

Classes

```
// To use the Color and its API
import java.awt.Color;

public class Luminance {

    public static double lum(Color c) {
        int r = c.getRed();
        int g = c.getGreen();
        int b = c.getBlue();
        return .299*r + .587*g + .114*b;
    }
}
```

String Processing

Check if Palindrome

```
public static boolean isPalindrome(String s) {
    int N = s.length();
    for(int i = 0; i < N/2; i++) {
        if(s.charAt(i) != s.charAt(N-1-i))
            return false;
    }
    return true;
}
```

Extract File Name and Extension From a Command-Line Argument

```
String s = args[0];
int dot = s.indexOf(".");
String base = s.substring(0,dot);
String extension = s.substring(dot+1,s.length());
```

Print All Line in Standard Input That Contain a String Specified on the Command-Line

```
String query = args[0];
while(!StdIn.isEmpty()) {
    String s = StdIn.readLine();
    if(s.contains(query)) StdOut.println(s);
}
```

Print All the Hyperlinks in the Text File on Standard Input

```
while(!StdIn.isEmpty()) {
    String s = StdIn.readString();
    if(s.startsWith("https://") && s.endsWith(".edu"))
        StdOut.println(s);
}
```

Pointers

```
String s = "abc"
Sting s1 = "abc"
if(s == s1)
    // Comparing pointers - gives false

if(s.equals(t))
    // Comparing character sequences - gives true
```

Creating Data Types

Defining Data Types in Java

- Instance variables: Set of values
- Methods: Operations defined on those values
- Constructors: Create and initialise new objects

Client

```
public static void main(String[] args) {
    double x = Double.parseDouble(args[0]);
    double y = Double.parseDouble(args[1]);
    Charge c1 = new Charge(.51,.63,21.3);
    Charge c2 = new Charge(.13,.94,81.9);
    double v1 = c1.potentialAt(x,y);
    double v2 = c2.potentialAt(x,y);
    StdOut.println(c1); // Automatically invokes toString()
    StdOut.println(c2); // Automatically invokes toString()
    StdOut.println(v1+v2);
}
```

Instance Variables

- Declare outside any method
- Always use access modifier `private`
- Use modifier `final` with instance variables that never change

```
public class Charge {
    private final double rx,ry;
    private final double q;
}
```

Constructors

Specifies what happens when you create a new object.

- Public access modifier
- No return type
- Constructor name is the same as the class name
- Accepts a number of arguments
- This is all called the signature
- The body usually is used to set instance variables

```
public Charge (double x0, double y0, double q0) {
    rx = x0);
    ry = y0;
    q = q0;
}
```

The constructor is invoked using the `new` operator to create a new object. The following example shows creating and initialising an object.

```
Charge c1 = new Charge(.51, .63, 21.3);
```

Methods

- Usually a public modifier (if no modifier is specified it defaults to public)
- Usually has a return type
- Given a method name
- Accepts a number of arguments
- This is all called the signature
- The body performs a number of operations and can use and modify instance variables

```
pubic double potentialAt(double x, double y) {
    double k = 8.99e09;
    double dx = x - rx;
    double dy = y - ry;
    return k * q / Math.sqrt(dx*dx + dy*dy);
}
```

The method is invoked using the dot operator

```
double v1 = c1.potentialAt(x,y);
```

Applications of Data Types

- Java objects model real-world objects
- Data types enable us to add our own abstractions

Enums

Enumerated types are an alternative to writing classes. It is a type that has a fixed list of possible values which are specified when the enum is created.

```
enum enum-typename { list-of-enum-values }
```

The definition cannot be inside a subroutine (function/method).

```
enum Season { SPRING, SUMMER, FALL, WINTER }

...

Season vacation;
vacation = Season.SUMMER;
Season.SPRING.ordinal(); // 0
```

Static Subroutines and Static Variables

- In a running program, a static subroutine is a member of the class itself
- Non-static subroutine definitions, are only there to be used when objects are created, and the subroutines themselves become members of the object
- The distinction applies to variables too

Inheritance

Extending Existing Classes

- Used when there is an existing class that can be adapted with a few changes or additions
- When `protected` is used as an access modifier to a method or member variable of a class, that member can be used in subclasses — direct or indirect — of the class in which it is defined, but it cannot be used in non-subclasses. Note the exception that a protected member can be accessed by any class in the same package as the class that contains the protected member. Thus, using the protected modifier is strictly more liberal than using no modifier at all (allows access from classes in the same package and from subclasses that are not in the same package)

```
public class subclass-name extends existing-class-name {
   .
   .   // Changes and additions.
   .
}
```

Class Hierarchy

- One class can inherit part or all of its structure and behaviour from another class
- Subclass and superclass
- A variable that can hold a reference to an object of class A can also hold a reference to an object belonging to any subclass of A

Polymorphism

- A method is polymorphic if the action performed by the method depends on the actual type of the object to which the method is applied at runtime

```
for (int i = 0; i < shapelist.length; i++ ) {
    Shape shape = shapelist[i];
    shape.redraw(); // redraw is polymorphic
}
```

Abstract Classes

- An abstract class is one that is not used to construct objects, but only as a basis for making subclasses
- An abstract class exists only to express the common properties of all its subclasses
- An abstract class contains abstract methods
- `redraw` is an abstract method. It exists to specify the common interface of all the concrete versions of `redraw()`

```
...

abstract void redraw(); // Can be left blank

...
```

Interfaces

- Multiple inheritance is not allowed in Java. However, Java does have a feature that can accomplish many of the same goals as multiple inheritance: interfaces
- A class can implement an interface by providing an implementation for each of the methods specified by the interface

Interface:

```
public interface Strokeable { // Pure abstract class
    public void stroke(GraphicsContext g);
        // Pure public, abstract method

    // Can also declare public static final variables
    int CM_PER_M = 100;
}
```

Implementing:

```
public class Line implements Strokeable {
    public void stroke(GraphicsContext g) {
        . . .
    }
    . . .
}
```

A class can extend only one other class. It can implement any number of interfaces:

```
class FilledCircle extends Circle
                        implements Strokeable, Fillable {
   . . .
}
```

An interface can extend one or more other interfaces.

Since Java 8, interfaces can contain default methods:

```
public interface Readable { // represents a source of input

    public char readChar();  // read the next ch from input

    default public String readLine() {
        StringBuilder line = new StringBuilder();
        char ch = readChar();
        while (ch != '\n') {
            line.append(ch);
            ch = readChar();
        }
        return line.toString();
    }

}
```

Even though you can't construct an object from an interface, you can declare a variable whose type is given by the interface. This variable can refer to any object that implements the interface

```
Strokeable figure;
figure = new Circle();
```