SP2 — Lab sheet 1

2014

Based upon exercises from Java for Everyone, 2e, Chapters 1 through 5.
Most of these you have probably encountered during Software and Programming I.

---

1. Open a simple text editor, or create a BlueJ project, and type in the following program:

```
public class HelloPrinter {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

   Save your work in a file called `HelloPrinter.java`. Pay attention to the case of letters in your program and in the name of the file. Compile your program in BlueJ or from the console window. If you use a console window then you will type the following command:

```
javac HelloPrinter.java
```

   What files are contained in the directory after you have compiled the program?
   Run your program by typing the following command (again, if you are using a console):

```
java HelloPrinter
```

   What is the output of your program? What is contained in the `.class` file?

2. Write a program that does the following:
   (a) Create seven variables, one for each of the primitive number types in Java.
   (b) Initialize each variable with any appropriate value.
   (c) Print out the name of each variable and its value.
   (d) Modify the value of each variable with an assignment statement and print out the names of the variables and their new values.
   (e) Create seven constants, one for each of the primitive number types in Java.
   (f) Print the name of the constant and its value.
   (g) What happens if you try to assign a value to a constant?

3. Execute the program shown below. Each invocation of `println` outputs an arithmetic expression. The first two `println` commands are followed by comments that describe the operations that occur in each expression. Complete the program by adding a comment after each `println` statement that describes all the arithmetic operations that occur when evaluating the expression that is printed.

```
public class Expressions {
    public static void main(String[] args) {
        int a = 3;
        int b = 4;
        int c = 5;
        int d = 17;
        System.out.println((a + b) / c);
        // 3 and 4 are added with sum 7
        // 7 is divided by 5 with quotient 1
        System.out.println(a + b / c);
        // 4 is divided by 5 with quotient 0
        // 3 is added to 0 with sum 3
        System.out.println(a++);
        System.out.println(a--);
        System.out.println(a + 1);
        System.out.println(d % c);
        System.out.println(d / c);
        System.out.println(d % b);
        System.out.println(d / b);
        System.out.println(d + a / d + b);
        System.out.println((d + a) / (d + b));
        System.out.println(Math.sqrt(b));
        System.out.println(Math.pow(a, b));
        System.out.println(Math.abs(-a));
        System.out.println(Math.max(a, b));
    }
}
```

4. Write a program that prompts the user to enter two integers. Print the smaller of the two numbers entered. Youll need to use a `Scanner` and a `Math` method.

5. Adding (incrementing) or subtracting (decrementing) the value one from an integer variable is a common, everyday operation. To increment an `int` variable `x`, we could code

   ```
   x = x + 1;
   ```

   As an alternative, we could use the special operators `++` and `--` to increment and decrement a variable. Use the first method to increment `x` in the program below. Print the value of `x` after incrementing. Use the `++` operator to increment `y` in the program below. Print the value of `y` after incrementing.

```
public class IncrementDemo {
    public static void main(String[] args) {
        int x = 10;
        int y = -3;
        // Put your code here
    }
}
```

6. What is the output of the following program and why?

```
public class AverageCalculator {
    public static void main(String[] args) {
        int age1 = 18;
        int age2 = 35;
        int age3 = 50;
        int age4 = 44;
        double averageAge = (age1 + age2 + age3 + age4) / 4;
        System.out.println(averageAge);
    }
}
```

7. Fix the program in the previous question so that it yields the correct result.

8. What is the output of the following program and why?

```
public class PercentagePrinter {
    public static void main(String[] args) {
        double probability = 8.70;
        int percentage = (int) (100 * probability);
        System.out.println(percentage);
    }
}
```

9. Fix the program from the previous question so that it displays the correct result. Remember that you can use `Math.round` to convert a floating-point value to its *closest* integer.

10. The `if` statement is used to implement a decision. The simplest form of an `if` statement has two parts: a condition and a body. If the *condition* is true, the *body* of the statement is executed. The body of the `if` statement consists of a statement block.

    Consider the following code:

```
if (n > 10) System.out.print("*****");
if (n > 7)  System.out.print("****");
if (n > 4)  System.out.print("***");
if (n > 1)  System.out.print("**");
System.out.println("*");
```

    How many `*` will be printed when the code is executed

(a) with n = 6?

(b) with n = 20?

(c) with n = 2?

(d) with n = 1?

11. An alternate form for an if statement has multiple parts: a condition that evaluates to true or false, a statement that is executed if the condition is true, the word else, and finally a statement that is executed when the condition is false. Each statement can be a simple statement consisting of a single Java instruction, a compound statement (such as another if statement) or a block statement (matching braces that surround one or more Java statements). We suggest using the brace notation in every case. Consider the code below that prompts the user to input a value for x and for y. It then prints the smallest value contained in the variables x and y.

```
import java.util.*; // very bad practice  why?

public class SmallestInt {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a value for x:");
        int x = scan.nextInt();
        System.out.println("Enter a value for y:");
        int y = scan.nextInt();
        if (x <= y) {
            System.out.println("The smallest value was " + x);
        } else {
            System.out.println("The smallest value was " + y);
        }
    }
}
```

Modify the code above so that it prompts the user to enter a third value for a variable z. Rewrite the logic so that the program prints out the smallest value contained in x, y, and z.

12. The code below is a more efficient solution to the previous problem.

```
import java.util.*;
public class SmallestInt2 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a value for x:");
        int x = scan.nextInt();
        int smallest = x; // x is the smallest value so far

        System.out.println("Enter a value for y:");
        int y = scan.nextInt();
```

```
        if (y < smallest) {
            smallest = y;   // Update smallest if necessary
        }
        System.out.println("Enter a value for z:");
        int z = scan.nextInt();
        if (z < smallest) {
            smallest = z;   // Update smallest if necessary
        }
        System.out.println("The smallest value was " + smallest);
    }
}
```

Modify the code so that it prompts the user for four integers (w, x, y, and z) and prints the smallest value contained in those variables. How hard would it be to modify the version of the program you wrote in the previous question to solve the four-variable problem?

13. In the code below, the if statement evaluates the condition `x < 10` and assigns the variable `color` either the value `"red"` or `"blue"`. The condition is first examined and the corresponding alternative is taken. The strategy in this code is to wait until we know exactly which alternative to take before assigning a colour.

```
String color = "";
if (x < 10)
   color = "red";
else
   color = "blue";
```

Often an alternate strategy (lets call it *"act first, decide later"*) can be used to simplify the logic. If the actions in the true and false statements are reversible, we can go ahead and execute the false (or true) statement and then code an `if` statement that determines whether that action was correct. If the action was incorrect we can reverse it. We solve the problem posed above using this alternative strategy:

```
String color = "blue";
if (x < 10)
   color = "red";
```

We "act first" by assuming blue is the right color to assign to the variable color. We correct it if that was wrong. The logic is simpler and involves coding one less alternative in the `if` statement.

Rewrite the code above again, but this time start by setting the color variable to "`red`". How does that change the condition?

14. The relational operators in Java are `==`, `!=`, `<`, `>`, `<=`, and `>=`. Assume `x` and `y` are integers. Using relational operators, formulate the following conditions in Java:

(a) `x` is positive

(b) `x` is zero or negative

(c) `x` is at least `10`

(d) `x` is less than !10!

(e) `x` and `y` are both zero

(f) `x` is even

15. Copy and run the following program.
    Explain how the program compares the two strings.
    How can you modify the program so that `str2` and `str3` are equal when they are compared?

```
public class StringEqual {
    public static void main(String[] args) {
        String str1 = "abcd";
        String str2 = "abcdefg";
        String str3 = str1 + "efg";
        System.out.println("str2 = " + str2);
        System.out.println("str3 = " + str3);
        if (str2 == str3)
            System.out.println("The strings are equal");
        else
            System.out.println("The strings are not equal");
    }
}
```

16. Write a program that prompts the user to enter three strings. Compare the `String` objects lexicographically and print the *middle-valued string*. For example, if the three strings were `"abcd"`, `"wxyz"`, and `"pqrs"`, the program would print `"pqrs"`.

    Limit yourself to simple, nested `if` statements that **dont** use the `Boolean` operators `&&` or `||`. Be sure to test your code by providing input data that tests every path through your code.

    Make a list of values for `str1`, `str2`, and `str3` that would thoroughly test the code.

17. Rewrite the previous program using the `Boolean` operator `&&` to simplify the logical structure.

18. Programmers take many visual cues from the indenting in a program, so it is imperative that the indentation we provide reflects the logic of the program. Consider the program below, which is extremely difficult to read because it is so badly indented. Take the program and indent it properly so that the indents reflect the logical structure of the program.

    Some programmers adopt a style for `if` statements in which each alternative is always represented as a block of code surrounded by `{}`. There are a couple of advantages to this style:

- the braces clearly indicate the true and false alternatives, and
- the program is easier to maintain if you need to add more lines within one of the alternatives in the future.

After indenting the following code, add `{}` to all the alternatives.

```
public class BadIfs {
     public static void main(String[] args) {
    int x = 9;
    int y = 3;
  int z = 7;
  if (x < y){System.out.println("aaa"); if (x < z)
  System.out.println("bbb"); } else System.out.println("ccc");
System.out.println("ddd");
if (y > z)if (z > x) System.out.println("eee");
    else System.out.println("fff"); else
  System.out.println("ggg"); }
}
```

Here is the output of the program if you run it as listed:

```
ccc ddd ggg
```

Make sure that the program still produces the same output when you have indented it properly.

19. According to the following program, what colour results when using the following inputs?

    (a) `Y N Y`

    (b) `Y Y N`

    (c) `N N N`

```
import java.util.Scanner;

public class ColorMixer {
    public static void main(String[] args) {
        String mixture = "";
        boolean red = false;
        boolean green = false;
        boolean blue = false;

        Scanner in = new Scanner(System.in);
        System.out.print("Include red in mixture? (Y/N) ");
        String input = in.next();

        if (input.toUpperCase().equals("Y"))
            red = true;

        System.out.print("Include green in mixture? (Y/N) ");
```

```
        input = in.next();

        if (input.toUpperCase().equals("Y"))
            green = true;

        System.out.print("Include blue in mixture? (Y/N) ");
        input = in.next();

        if (input.toUpperCase().equals("Y"))
            blue = true;

        if (!red && !blue && !green)
            mixture = "BLACK";
        else if (!red && !blue)
            mixture = "GREEN";
        else if (red)
            if (green || blue)
                if (green && blue)
                    mixture = "BLACK";
                else if (green)
                    mixture = "YELLOW";
                else
                    mixture = "PURPLE";
            else
                mixture = "BLACK";
        else if (!green)
            mixture = "BLUE";
        else
            mixture = "WHITE";
        System.out.println("Your mixture is " + mixture);
    }
}
```

20. Loops provide a mechanism for repeating a block of code called *the loop body*. Many loops are controlled with a single variable, which we will refer to as *the loop control variable* or *the loop index*.

    Consider the code below. What is the output the program produces? (Decide this **without** executing the code.)

```
/**
    A simple program that prints a loop control variable.
*/
public class SimpleLoop {
    public static void main(String[] args) {
        int i = 0;
        int limit = 6;
        while (i < limit) {
            System.out.println("i = " + i);
```

```
            i++;
        }
    }
}
```

21. Consider the code shown below.

```
/**
A simple program that prints a loop control variable.
*/
public class SimpleLoop2 {
    public static void main(String[] args) {
        int i = 0;
        int limit = 6;
        while (i < limit) {
            System.out.println("i = " + i);
            i++;
        }
    }
}
```

    What happens if you comment out the line that increments `i`?
    Will the program ever stop looping?
    (Decide this **without** executing the code.)

22. Manipulating the *loop control variable* is a critical skill in learning to write code with loops. Modify the program in question 20 so that it produces the following output:

```
i=6 i=8 i = 10 i = 12 i = 14 i = 16 i = 18 ...
i = 98
```

23. There is a famous story about a primary school teacher who wanted to occupy his students time by making the children compute the sum of `1 + 2 + 3 + ... + 100` by hand. As the story goes, the teacher was astounded when one of the children immediately produced the correct answer: `5050`. The student, a child prodigy, was *Carl Gauss*, who grew up to be one of the most famous mathematicians of the eighteenth century.

    Repeat Gausss remarkable calculation by writing a loop that will compute and print the above sum. After you have the program working, rewrite it so you can compute `1 + 2 + ... + n` where `n` is any positive integer.

24. Java provides three types of loops: `while`, `for`, and `do` (also called *do-while*). Theoretically, they are interchangeable – any program you write with one kind of loop could be rewritten using any of the other types of loops. As a practical matter, though, it is often the case that choosing the right kind of loop will make your code easier to produce, debug, and read.

    It takes time and experience to learn to make the best loop choice, so this exercise gives you some of that experience. Rewrite the program from question 23 using a `for` loop. Repeat the exercise again but this time use a `do while` loop.

Which form of loop seems to work best and why?

25. Write a program that uses a `while` loop. During each iteration of the loop, prompt the user to enter a number – positive, negative, or zero. Keep a running total of the numbers the user enters and also keep a count of the number of entries the user makes. The program should stop whenever the user enters `q` to quit.

    When the user has finished, print the grand total and the number of entries the user typed.

26. Use nested `for` loops to produce the following output:

```
X
XX
XXX
XXXX
XXXXX
```

    The outer loop can control the number of rows that will be printed. The inner loop can control the number of `X`s that print on a single line. The trick is to notice that there is a relationship between the row number and the number of `X`s in the row. This relationship allows you to use the outer loop control variable to control the inner loop.

27. One loop type might be better suited than another to a particular purpose. The following usages are idiomatic:

`for`	Known number of iterations
`while`	Unknown number of iterations
`do`	At least one iteration

    Convert the following `while` loop to a `do` loop.

```
import java.util.Scanner;

public class PrintSum {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int sum = 0;
        int n = 1;
        while (n != 0) {
            System.out.print("Please enter a number, 0 to quit: ");
            n = in.nextInt();
            if (n != 0) {
                sum = sum + n;
                System.out.println("Sum = " + sum);
            }
        }
    }
}
```

28. Is the `do` loop in the previous question an improvement over the `while` loop? Why or why not?

29. Convert this `while` loop to a `for` loop.

```
import java.util.Scanner;
/**
    Program to compute the first integral power to which 2 can be
    raised that is greater than that multiple of a given integer.
*/
public class CountPowerOf2 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Please enter a number, 0 to quit: ");

        int n = in.nextInt();
        int i = 1;

        while (n * n > Math.pow(2, i))
            i++;

        System.out.println("2 raised to " + i
                           + " is the first power of two greater than " + n + " squared");
    }
}
```

30. Convert this `for` loop to a `while` loop:

```
public static void main(String[] args) {
   for (int i = 1; i <= 10; i++)
      System.out.println(i + " squared equals " + i * i);
}
```

31. Write a static method called `AreaOfRectangle` that is passed two float-point values for the *length* and *width* of a rectangle. The method returns the product of the length and width as a `double`.

    Comment the method using `javadoc` conventions. Write a `main` method that creates the following variables to describe the sides of a rectangle:

```
double length = 3.4;
double width = 8.4;
```

    The `main` method should print the length, width, and area of the rectangle.

32. Run the following code. The `sum` method in this program violates a design principle (that methods should not try to modify an argument) when it assigns `5` to `a`, and `6` to `b`. The values have changed; this can be seen by examining the value the `sum` method returns. What about arguments `x` and `y`? Are their values changed, too? In other words: Do the assignments made in the method body have side effects in the main program?

```
public class Area {
    public static void main(String[] args) {
        int x = 2;
        int y = 3;
        System.out.println("x: " + x + " y: " + y + " Sum: " + sum(x, y));
    }

    /**
     * Computes the sum of two arguments.
     *
     * @param a an int operand to be added
     * @param b another int operand
     * @return the sum of a and b
     */
    public static int sum(int a, int b) {
        a = 5;
        b = 6;
        return a + b;
    }
}
```

33. Credit card numbers contain a check digit that is used to help detect errors and verify that the card number is valid. The check digit can help detect all single-digit errors and almost all transpositions of adjacent digits.

    In this problem we will write some methods that will allow us to quickly check whether a card number is invalid. We will limit our numbers to seven digits and the rightmost digit will be the check digit.

    For example, if the credit card number is 2315778, the check digit is 8. We number the digit positions starting at the check digit, moving left. Heres the numbering for credit card number 2315778:

Position	Digit
1	8
2	7
3	7
4	5
5	1
6	3
7	2

    To verify that the card number is correct we will need to "decode" every digit. The decoding process depends on the position of the digit within the credit card number:

    (a) If the digit is in an odd-numbered position, simply return the digit,

    (b) If the digit is in an even-numbered position, double it. If the result is a single digit, return it; otherwise, add the two digits in the number and return the sum.

    For example, if we decode 8 and it is in an odd position, we return 8. On the other hand,

if 8 is in an even position, we double it to get 16, and then return 1 + 6 = 7. Decoding 4 in an odd position would return 4, and decoding it an even position would return 8.

As a first step to being able to being able to detect invalid numbers, you should write a method called `decode` that is passed an `int` for the digit and a `boolean` for the position (`true` = even position, `false` = odd position). The method should decode the digit using the method described above and return an `int`. Test your method with the `main` method below:

```
public class Luhn {
    public static void main(String[] args) {
        boolean even = false;
        System.out.println(decode(1, even));
        System.out.println(decode(2, even));
        System.out.println(decode(3, even));
        System.out.println(decode(4, even));
        System.out.println(decode(5, even));
        System.out.println(decode(6, even));
        System.out.println(decode(7, even));
        System.out.println(decode(8, even));
        System.out.println(decode(9, even));
        even = ! even;
        System.out.println(decode(1, even));
        System.out.println(decode(2, even));
        System.out.println(decode(3, even));
        System.out.println(decode(4, even));
        System.out.println(decode(5, even));
        System.out.println(decode(6, even));
        System.out.println(decode(7, even));
        System.out.println(decode(8, even));
        System.out.println(decode(9, even));
    }

    public static int decode(int digit, boolean position) {
        // Your code goes here
        return 0;
    }
}
```

34. Now that we can decode single digits, its time to build some code that will help detect errors in credit card numbers. Heres the idea:

    (a) Starting with the check digit and moving left, compute the sum of all the decoded digits.

    (b) Compute the remainder of the sum using integer division by 10. If the result is not zero, the credit card number is invalid. Otherwise, the card number is likely to be valid.

    Here are two examples:

Card number: 2315778	Card number 1234567
decode(8, false) = 8	decode(7, false) = 7
decode(7, true) = 5	decode(6, true) = 3
decode(7, false) = 7	decode(5, false) = 5
decode(5, true) = 1	decode(4, true) = 8
decode(1, false) = 1	decode(3, false) = 3
decode(3, true) = 6	decode(2, true) = 4
decode(2, false) = 2	decode(1, false) = 1
Sum = 30	Sum = 31
30 mod 10 = 0	31 mod 10 = 1
This number may be valid	This number is invalid

Write a `static` method called `checkDigits` that is passed a seven-digit credit card number and that performs the steps described above. You should reuse the `decode` method that you wrote earlier. The method should return the word `valid` if the number passes the test and `invalid` otherwise.

You should test your methods with the following `main` method:

```
public class Luhn2 {
    public static void main(String[] args) {
        int num = 2315778;
        System.out.println("CC number: " + num + " is " + checkDigits(num));
        num = 1234567;
        System.out.println("CC number: " + num + " is " + checkDigits(num));
        num = 7654321;
        System.out.println("CC number: " + num + " is " + checkDigits(num));
        num = 1111111;
        System.out.println("CC number: " + num + " is " + checkDigits(num));
    }

    public static String checkDigits(int number) {
        // Put your code here
        return "";
    }
}
```