Lab 1. Exercises on Java Basics

Writing Good Programs

The only way to learn programming is program, program and program. Learning programming is like learning cycling, swimming or any other sports. You can't learn by watching or reading books. Start to program immediately. On the other hands, to improve your programming, you need to read many books and study how the masters program.
It is easy to write programs that work. It is much harder to write programs that not only work but also easy to maintain and understood by others – I call these good programs. In the real world, writing program is not meaningful. You have to write good programs, so that others can understand and maintain your programs.
Pay particular attention to:

1. **Coding style**:
   - Read Java code convention: "Java Style and Commenting Guide".
   - Follow the Java Naming Conventions for variables, methods, and classes STRICTLY. Use CamelCase for names. Variable and method names begin with lowercase, while class names begin with uppercase. Use nouns for variables (e.g., radius) and class names (e.g., Circle). Use verbs for methods (e.g., getArea(), isEmpty()).
   - **Use Meaningful Names**: Do not use names like a, b, c, d, x, x1, x2, and x1688 - they are meaningless. Avoid single-alphabet names like i, j, k. They are easy to type, but usually meaningless. Use single-alphabet names only when their meaning is clear, e.g., x, y, z for co-ordinates and i for array index. Use meaningful names like row and col (instead of x and y, i and j, x1 and x2), numStudents (not n), maxGrade, size (not n), and upperbound (not n again). Differentiate between singular and plural nouns (e.g., use books for an array of books, and book for each item).
   - Use consistent indentation and coding style. Many IDEs (such as Eclipse / NetBeans) can re-format your source codes with a single click.
2. **Program Documentation**: Comment! Comment! and more Comment to explain your code to other people and to yourself three days later.
3. The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)).

1 Debugging/Tracing Programs using a Graphic Debugger

1.1 Factorial

The following program computes and prints the factorial of n ($= 1 * 2 * 3 * \cdots * n$). The program, however, has a logical error and produce a wrong answer for $n = 20$ ("The Factorial of 20 is -2102132736" – negative?!).

Use the graphic debugger (of Eclipse/NetBeans/IntelliJ) to debug the program by single-step through the program and tabulating the values of i and factorial at the statement marked by (*).

You should try out debugging features such as "Breakpoint", "Step Over", "Watch variables", "Run-to-Line", "Resume", "Terminate", among others. (Read "Eclipse for Java" or "NetBeans for Java" for details).

```java
/**
* Print factorial of n
*/
public class Factorial {
  public static void main(String[] args) {  // Set an initial
     ↪ breakpoint at this statement
    int n = 20;
    int factorial = 1;

    // n! = 1*2*3...*n
    for (int i = 1; i <= n; i++) {  // i = 1, 2, 3, ..., n
      factorial = factorial * i;    // *
    }
    System.out.println("The Factorial of " + n + " is " + factorial);
  }
}
```

2 Getting Started Exercises

2.1 CheckPassFail

Write a program called **CheckPassFail** which prints "PASS" if the int variable "mark" is more than or equal to 50; or prints "FAIL" otherwise. The program shall always print "DONE" before exiting.

Hints
Use $\geq$ for greater than or equal to comparison.

```
/**
* Trying if-else statement.
*/
public class CheckPassFail {  // Save as "CheckPassFail.java"
  public static void main(String[] args) {  // Program entry point
    int mark = 49;   // Set the value of "mark" here!
    System.out.println("The mark is " + mark);

    // if-else statement
    if ( ..... ) {
      System.out.println( ..... );
    } else {
      System.out.println( ..... );
    }
    System.out.println( ..... );
  }
}
```

Try mark = $0, 49, 50, 51, 100$ and verify your results.
Take note of the source-code **indentation**!!! Whenever you open a block with '{', indent all the statements inside the block by 3 (or 4 spaces). When the block ends, un-indent the closing '}' to align with the opening statement.

2.2 CheckOddEven

Write a program called **CheckOddEven** which prints "Odd Number" if the int variable "number" is odd, or "Even Number" otherwise. The program shall always print "Bye!" before exiting.

Hints
n is an even number if $(n \,\%\, 2)$ is 0; otherwise, it is an odd number. Use == for comparison, e.g., $(n \,\%\, 2) == 0$.

```
/**
* Trying if-else statement and modulus (%) operator.
*/
public class CheckOddEven {   // Save as "CheckOddEven.java"
  public static void main(String[] args) {  // Program entry point
    int number = 49;       // Set the value of "number" here!
    System.out.println("The number is " + number);
    if ( ..... ) {
      System.out.println( ..... );   // even number
    } else {
      System.out.println( ..... );   // odd number
    }
```

```
13        System.out.println( ..... );
      }
15    }
```

Try number = $0, 1, 88, 99, -1, -2$ and verify your results.
Again, take note of the source-code identation! Make it a good habit to ident your code properly, for ease of reading your program.

2.3 PrintNumberInWord

Write a program called **PrintNumberInWord** which prints "ONE", "TWO",... , "NINE", "OTHER" if the int variable "number" is $1, 2, ..., 9$, or other, respectively. Use (a) a "nested-if" statement; (b) a "switch-case-default" statement.

Hints

```
1   /**
    * Trying nested-if and switch-case statements.
3   */
    public class PrintNumberInWord {   // Save as "PrintNumberInWord.java"
5     public static void main(String[] args) {
        int number = 5;  // Set the value of "number" here!
7
        // Using nested-if
9       if (number == 1) {   // Use == for comparison
          System.out.println( ..... );
11      } else if ( ..... ) {
          .....
13      } else if ( ..... ) {
          .....
15        .....
          .....
17      } else {
          .....
19      }

21      // Using switch-case-default
        switch(number) {
23        case 1:
            System.out.println( ..... );
25          break;  // Don't forget the "break" after each case!
          case 2:
27          System.out.println( ..... );
            break;
29          .....
            .....
```

```
31            default:
                System.out.println( ..... );
33          }
        }
35    }
```

Try number = 0, 1, 2, 3, ..., 9, 10 and verify your results.

2.4 PrintDayInWord

Write a program called **PrintDayInWord** which prints "Sunday", "Monday", ... "Saturday" if the *int* variable "*dayNumber*" is 0, 1, ..., 6, respectively. Otherwise, it shall print "Not a valid day". Use (a) a "nested-if" statement; (b) a "switch-case-default" statement.
Try dayNumber = 0, 1, 2, 3, 4, 5, 6, 7 and verify your results.

3 Exercises on Decision and Loop

3.1 SumAverageRunningInt

Write a program called **SumAverageRunningInt** to produce the sum of 1, 2, 3, . . . , to 100. Store 1 and 100 in variables *lowerbound* and *upperbound*, so that we can change their values easily. Also compute and display the average. The output shall look like:

Command window

```
The sum of 1 to 100 is 5050
The average is 50.5
```

Hints

```
/**
* Compute the sum and average of running integers from a lowerbound
* to an upperbound using loop.
*/
public class SumAverageRunningInt {  // Save as "SumAverageRunningInt.java"
  public static void main(String[] args) {
    // Define variables
    int sum = 0;           // The accumulated sum, init to 0
    double average;        // average in double
    final int LOWERBOUND = 1;
    final int UPPERBOUND = 100;

```

```
      // Use a for-loop to sum from lowerbound to upperbound
14    for (int number = LOWERBOUND; number <= UPPERBOUND; ++number) {
        // The loop index variable number = 1, 2, 3, ..., 99, 100
16      sum += number;      // same as "sum = sum + number"
      }
18
      // Compute average in double. Beware that int / int produces int!
20    ......
      // Print sum and average
22    ......
    }
24  }
```

Try

1. Modify the program to use a "while-do" loop instead of "for" loop.

```
      int sum = 0;
2     int number = LOWERBOUND;          // declare and init loop index variable
      while (number <= UPPERBOUND) {    // test
4       sum += number;
        ++number;                       // update
6     }
```

2. Modify the program to use a "do-while" loop.

```
1     int sum = 0;
      int number = LOWERBOUND;          // declare and init loop index variable
3     do {
        sum += number;
5       ++number;                       // update
      } while (number <= UPPERBOUND);   // test
```

3. What is the difference between "for" and "while-do" loops? What is the difference between "while-do" and "do-while" loops?

4. Modify the program to sum from 111 to 8899, and compute the average. Introduce an *int* variable called *count* to count the numbers in the specified range (to be used in computing the average).

```java
int count = 0;   // Count the number within the range, init to 0
for ( ...; ...; ... ) {
   ......
   ++count;
}
```

5. Modify the program to find the "sum of the squares" of all the numbers from 1 to 100, i.e. $1*1+2*2+3*3+...+100*100$.

6. Modify the program to produce two sums: sum of odd numbers and sum of even numbers from 1 to 100. Also compute their absolute difference.

```java
// Define variables
int sumOdd  = 0;   // Accumulating sum of odd numbers
int sumEven = 0;   // Accumulating sum of even numbers
int absDiff;       // Absolute difference between the two sums
......

// Compute sums
for (int number = ...; ...; ...) {
   if (......) {
      sumOdd += number;
   } else {
      sumEven += number;
   }
}

// Compute Absolute Difference
if (sumOdd > sumEven) {
   absDiff = ......;
} else {
   absDiff = ......;
}

// OR use one liner conditional expression
absDiff = (sumOdd > sumEven) ? ..... : ......;
```

3.2 HarmonicSum

Write a program called **HarmonicSum** to compute the sum of a harmonic series, as shown below, where $n = 50000$. The program shall compute the sum from left-to-right as well as from the right-to-left. Are the two sums the same? Obtain the absolute difference between these two sums and explain the difference. Which sum is more accurate?

$$\text{harmonic}(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}.$$

Hints

```
/**
 * Compute the sum of harmonics series from left-to-right and right-to-left.
 */
public class HarmonicSum {   // Save as "HarmonicSum.java"
  public static void main(String[] args) {
    // Define variables
    final int MAX_DENOMINATOR = 50000;  // Use a more meaningful name
        ↪ instead of n
    double sumL2R = 0.0;          // Sum from left-to-right
    double sumR2L = 0.0;          // Sum from right-to-left
    double absDiff;               // Absolute difference between the two sums

    // for-loop for summing from left-to-right
    for (int denominator = 1; denominator <= MAX_DENOMINATOR; ++
        ↪ denominator) {
      // denominator = 1, 2, 3, 4, 5, ..., MAX_DENOMINATOR
      ......
      // Beware that int/int gives int, e.g., 1/2 gives 0.
    }
    System.out.println("The sum from left-to-right is: " + sumL2R);

    // for-loop for summing from right-to-left
    ......

    // Find the absolute difference and display
    if (sumL2R > sumR2L) {
      ......
    } else {
      ......
    }
  }
}
```

3.3 ComputePI

Write a program called **ComputePI** to compute the value of π, using the following series expansion. Use the maximum denominator (MAX_DENOMINATOR) as the terminating condition. Try MAX_DENOMINATOR of $1000, 10000, 100000, 1000000$ and compare the *PI* obtained. Is this series suitable for computing *PI*? Why?

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} + \cdots\right).$$

Hints

Add to sum if the denominator $\%\,4$ is 1, and subtract from sum if it is 3.

```
double sum = 0.0;
int MAX_DENOMINATOR = 1000;    // Try 10000, 100000, 1000000
for (int denominator = 1; denominator <= MAX_DENOMINATOR; denominator
    ↪ += 2) {
  // denominator = 1, 3, 5, 7, ..., MAX_DENOMINATOR
  if (denominator % 4 == 1) {
    sum += ......;
  } else if (denominator % 4 == 3) {
    sum -= ......;
  } else {    // remainder of 0 or 2
    System.out.println("Impossible!!!");
  }
}
......
```

Try

1. Instead of using maximum denominator as the terminating condition, rewrite your program to use the maximum number of terms (MAX_TERM) as the terminating condition.

```
int MAX_TERM = 10000;  // number of terms used in computation
int sum = 0.0;
for (int term = 1; term <= MAX_TERM; term++) {  // term = 1, 2,
    ↪  3, ..., MAX_TERM
  // term = 1, 2, 3, 4, ..., MAX_TERM
  if (term % 2 == 1) {  // odd term number: add
    sum += 1.0 / (term * 2 - 1);
  } else {              // even term number: subtract
    ......
  }
}
```

2. JDK maintains the value of π in a built-in double constant called *Math.PI* (= 3.141592653589793). Add a statement to compare the values obtained and the *Math.PI*, in percents of *Math.PI*, i.e., (*piComputed* / *Math.PI*) *100.

3.4 Fibonacci

Write a program called **Fibonacci** to print the first 20 Fibonacci numbers $F(n)$, where

$$F(n) = F(n–1) + F(n–2) \quad \text{and} \quad F(1) = F(2) = 1.$$

Also compute their average. The output shall look like:

Command window

```
The first 20 Fibonacci numbers are:
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
The average is 885.5
```

Hints

```
/**
* Print first 20 Fibonacci numbers and their average
*/
public class Fibonacci {
  public static void main(String[] args) {
    int n = 3;           // The index n for F(n), starting from n=3, as
       ↪ n=1 and n=2 are pre-defined
    int fn;              // F(n) to be computed
    int fnMinus1 = 1;    // F(n-1), init to F(2)
    int fnMinus2 = 1;    // F(n-2), init to F(1)
    int nMax = 20;       // maximum n, inclusive
    int sum = fnMinus1 + fnMinus2;  // Need sum to compute average
    double average;

    System.out.println("The first " + nMax + " Fibonacci numbers are: ");
    ......

    while (n <= nMax) {  // n starts from 3
      // n = 3, 4, 5, ..., nMax
      // Compute F(n), print it and add to sum
      ......
      // Increment the index n and shift the numbers for the next iteration
      ++n;
      fnMinus2 = fnMinus1;
      fnMinus1 = fn;
    }

    // Compute and display the average (= sum/nMax).
    // Beware that int/int gives int.
    ......
  }
}
```

Try

1. Tribonacci numbers are a sequence of numbers $T(n)$ similar to Fibonacci numbers, except that a number is formed by adding the three previous numbers, i.e., $T(n) = T(n-1) + T(n-2) + T(n-3), T(1) = T(2) = 1$, and $T(3) = 2$. Write a program called **Tribonacci** to produce the first twenty Tribonacci numbers.

3.5 ExtractDigits

Write a program called **ExtractDigits** to extract each digit from an *int*, in the reverse order. For example, if the int is 15423, the output shall be "3 2 4 5 1", with a space separating the digits.

Hints
The coding pattern for extracting individual digits from an integer n is:

1. Use $(n \,\%\, 10)$ to extract the last (least-significant) digit.
2. Use $n = n/10$ to drop the last (least-significant) digit.
3. Repeat if $(n > 0)$, i.e., more digits to extract.

Take note that n is destroyed in the process. You may need to clone a copy.

```
int n = ...;
while (n > 0) {
  int digit = n % 10;  // Extract the least-significant digit
  // Print this digit
  ......
  n = n / 10;  // Drop the least-significant digit and repeat the loop
}
```

4 Exercises on Input, Decision and Loop

4.1 SumProductMinMax3

Write a program called **SumProductMinMax3** that prompts user for three integers. The program shall read the inputs as int; compute the sum, product, minimum and maximum of the three integers; and print the results. For examples,

Command window

```
Enter 1st integer: 8
Enter 2nd integer: 2
Enter 3rd integer: 9
The sum is: 19
The product is: 144
The min is: 2
The max is: 9
```

Hints

```
// Declare variables
// The 3 input integers
int number1;
int number2;
int number3;

// To compute these
int sum;
int product;
int min;
int max;

// Prompt and read inputs as "int"
Scanner in = new Scanner(System.in);  // Scan the keyboard
......
......
in.close();

// Compute sum and product
sum = ......
product = ......

// Compute min
// The "coding pattern" for computing min is:
// 1. Set min to the first item
// 2. Compare current min with the second item and update min if second item is smaller
// 3. Repeat for the next item
min = number1;          // Assume min is the 1st item
if (number2 < min) {    // Check if the 2nd item is smaller than current min
  min = number2;        // Update min if so
}
if (number3 < min) {    // Continue for the next item
  min = number3;
}

// Compute max - similar to min
......

// Print results
......
```

Try

1. Write a program called **SumProductMinMax5** that prompts user for five integers. The program shall read the inputs as *int*; compute the sum, product, minimum and maximum of the five integers; and print the results. Use five *int* variables: *number1*, *number2*, ..., *number5* to store the inputs.

4.2 CircleComputation

Write a program called **CircleComputation** that prompts user for the radius of a circle in floating point number. The program shall read the input as *double*; compute the diameter, circumference, and area of the circle in *double*; and print the values rounded to 2 decimal places. Use System-provided constant *Math.PI* for pi. The formulas are:

Command window

```
diameter = 2.0 * radius;
area = Math.PI * radius * radius;
circumference = 2.0 * Math.PI * radius;
```

Hints

```
// Declare variables
double radius;
double diameter;
double circumference;
double area;  // inputs and results - all in double
......

// Prompt and read inputs as "double"
System.out.print("Enter the radius: ");
radius = in.nextDouble();  // read input as double

// Compute in "double"
......

// Print results using printf() with the following format specifiers:
// %.2f for a double with 2 decimal digits
// %n for a newline
System.out.printf("Diameter is: %.2f%n", diameter);
......
```

Try

1. Write a program called **SphereComputation** that prompts user for the radius of a sphere in floating point number. The program shall read the input as *double*; compute the volume and surface area of the sphere in *double*; and print the values rounded to 2 decimal places. The formulas are:

   surfaceArea = 4 * Math.PI * radius * radius;
   volume = 4 /3 * Math.PI * radius * radius * radius; // But this does not work in programming?! Why?

Take note that you cannot name the variable surface area with a space or surface-area with a dash. Java's naming convention is *surfaceArea*. Other languages recommend surface_area with an underscore.

2. Write a program called **CylinderComputation** that prompts user for the base radius and height of a cylinder in floating point number. The program shall read the inputs as *double*; compute the base area, surface area, and volume of the cylinder; and print the values rounded to 2 decimal places. The formulas are:

```
baseArea = Math.PI * radius * radius;
surfaceArea = 2.0 * Math.PI * radius + 2.0 * baseArea;
volume = baseArea * height;
```

4.3 PensionContributionCalculator

Both the employer and the employee are mandated to contribute a certain percentage of the employee's salary towards the employee's pension fund. The rate is tabulated as follows:

Employee's Age	Employee Rate (%)	Employer Rate (%)
55 and below	20	17
above 55 to 60	13	13
above 60 to 65	7.5	9
above 65	5	7.5

However, the contribution is subjected to a salary ceiling of $\$6,000$. In other words, if an employee earns $\$6,800$, only $\$6,000$ attracts employee's and employer's contributions, the remaining $800 does not.

Write a program called **PensionContributionCalculator** that reads the monthly salary and age (in *int*) of an employee. Your program shall calculate the employee's, employer's and total contributions (in *double*); and print the results rounded to 2 decimal places. For examples,

```
Command window

Enter the monthly salary: $3000
Enter the age: 30
The employee's contribution is: $600.00
The employer's contribution is: $510.00
The total contribution is: $1110.00
```

Hints

```
// Declare constants
final int SALARY_CEILING = 6000;
final double EMPLOYEE_RATE_55_AND_BELOW = 0.2;
final double EMPLOYER_RATE_55_AND_BELOW = 0.17;
final double EMPLOYEE_RATE_55_TO_60 = 0.13;
final double EMPLOYER_RATE_55_TO_60 = 0.13;
final double EMPLOYEE_RATE_60_TO_65 = 0.075;
final double EMPLOYER_RATE_60_TO_65 = 0.09;
final double EMPLOYEE_RATE_65_ABOVE = 0.05;
final double EMPLOYER_RATE_65_ABOVE = 0.075;

// Declare variables
int salary;
int age;      // to be input

int contributableSalary;
double employeeContribution;
double employerContribution;
double totalContribution;
......

// Check the contribution cap
contributableSalary = ......

// Compute various contributions in "double" using a nested-if to handle 4 cases
if (age <= 55) {          // 55 and below
   ......
} else if (age <= 60) {   // (60, 65]
   ......
} else if (age <= 65) {   // (55, 60]
   ......
} else {                  // above 65
   ......
}

// Alternatively,
// if (age > 65) ......
// else if (age > 60) ......
// else if (age > 55) ......
// else ......
```

4.4 PensionContributionCalculatorWithSentinel

Based on the previous **PensionContributionCalculator**, write a program called **PensionContributionCalculatorWithSentinel** which shall repeat the calculations until user enter -1 for the salary. For examples,

Command window

```
Enter the monthly salary (or -1 to end): $5123
Enter the age: 21
The employee's contribution is: $1024.60
The employer's contribution is: $870.91
The total contribution is: $1895.51

Enter the monthly salary (or -1 to end): $5123
Enter the age: 64
The employee's contribution is: $384.22
The employer's contribution is: $461.07
The total contribution is: $845.30

Enter the monthly salary (or -1 to end): USD-1
Bye!
```

Hints

```
// Read the first input to "seed" the while loop
System.out.print("Enter the monthly salary (or -1 to end): $");
salary = in.nextInt();

while (salary != SENTINEL) {
  // Read the remaining
  System.out.print("Enter the age: ");
  age = in.nextInt();

  ......
  ......

  // Read the next input and repeat
  System.out.print("Enter the monthly salary (or -1 to end): $");
  salary = in.nextInt();
}
```

4.5 ReverseInt

Write a program that prompts user for a positive integer. The program shall read the input as *int*; and print the "reverse" of the input integer. For examples,

Command window

```
Enter a positive integer: 12345
The reverse is: 54321
```

Hints

Use the following coding pattern which uses a while-loop with repeated modulus/divide operations to extract and drop the last digit of a positive integer.

```
// Declare variables
int inNumber;   // to be input
int inDigit;    // each digit
......

// Extract and drop the "last" digit repeatably using a while-loop with modulus/divide
    operations
while (inNumber > 0) {
  inDigit = inNumber % 10; // extract the "last" digit
  // Print this digit (which is extracted in reverse order)
  ......
  inNumber /= 10;          // drop "last" digit and repeat
}
......
```

4.6 InputValidation

Your program often needs to validate the user's inputs, e.g., marks shall be between 0 and 100.

Write a program that prompts user for an integer between $0-10$ or $90-100$. The program shall read the input as *int*; and repeat until the user enters a valid input. For examples,

Command window

```
Enter a number between 0-10 or 90-100: -1
Invalid input, try again...
Enter a number between 0-10 or 90-100: 50
Invalid input, try again...
Enter a number between 0-10 or 90-100: 101
Invalid input, try again...
Enter a number between 0-10 or 90-100: 95
You have entered: 95
```

Hints

Use the following coding pattern which uses a do-while loop controlled by a *boolean* flag to do input validation. We use a do-while instead of while-do loop as we need to execute the body to prompt and process the input at least once.

```
// Declare variables
int numberIn;      // to be input
boolean isValid;   // boolean flag to control the loop
......

// Use a do-while loop controlled by a boolean flag
// to repeatably read the input until a valid input is entered
isValid = false;   // default assuming input is not valid
do {
  // Prompt and read input
  ......

  // Validate input by setting the boolean flag accordingly
  if (numberIn ......) {
    isValid = true;   // exit the loop
  } else {
    System.out.println(......);  // Print error message and repeat
  }
} while (!isValid);
......
```

4.7 AverageWithInputValidation

Write a program that prompts user for the mark (between $0 - 100$ in *int*) of 3 students; computes the average (in *double*); and prints the result rounded to 2 decimal places. Your program needs to perform input validation. For examples,

Command window

```
Enter the mark (0-100) for student 1: 56
Enter the mark (0-100) for student 2: 101
Invalid input, try again...
Enter the mark (0-100) for student 2: -1
Invalid input, try again...
Enter the mark (0-100) for student 2: 99
Enter the mark (0-100) for student 3: 45
The average is: 66.67
```

Hints

```
// Declare constant
final int NUM_STUDENTS = 3;

// Declare variables
```

```
int numberIn;
boolean isValid;   // boolean flag to control the input validation loop
int sum = 0;
double average;
......

for (int studentNo = 1; studentNo <= NUM_STUDENTS; ++studentNo) {
  // Prompt user for mark with input validation
  ......
  isValid = false;   // reset assuming input is not valid
  do {
     ......
  } while (!isValid);

  sum += ......;
}
......
```