Introductory Java Programming

Queen Mary University of London
School of Electronic Engineering and Computer Science

Course Code: EBU4201

Lab Sheet 8: Utility Classes and Exceptions

---

1. Consider the program you wrote in *Lab Sheet 6 – Q1* (**`ParityBitAdder.java`**). Originally, your program checked for valid user input using an **`if`** statement. Now, apart from ensuring that the user has provided two arguments when running your program (e.g., **`java ParityBitAdder 0100011 0`**), we also want to ensure that appropriate actions, using *exceptions*, are taken by the program if the user attempts to run the program in any of the following ways:

   i) Check if the data bits entered for the input binary number are indeed binary (i.e., they are either the digit **`0`** or **`1`**) <u>and</u> that the input number is exactly 7 bits long. If the user attempts to run the program with a non-binary value, then the exception **`NonBinaryValue`** should be thrown. Create this exception class, for which the exception message is:

   ```
   "Error: The first input to this program must be a 7-bit binary
   number. Please try again!"
   ```

   ii) If the user attempts to enter a value other than **`0`** or **`1`** for the parity bit argument (i.e., for the program's second argument), then the exception **`IllegalParityValue`** should be thrown. Create this exception class, for which the exception message is:

   ```
   "Error: The program's parity bit input (the second argument)
   must be either 0 or 1. Please try again!"
   ```

   <u>Your improved program should be named **`ParityBitAdder_v2.java`**</u>[1]

2. The program you wrote in *Q1* of this lab sheet requires two parameters (i.e., a 7-bit binary value and an integer representing the parity bit) as the command line arguments, stored by your program in **`args[0]`** and **`args[1]`**. Now we want to write a new version of this program that explicitly prompts the user to input these two values[2].

   <u>Your new program should be named **`ParityBitAdder_v3.java`**</u>

---

[1] **Note**: You should also have written two *exception classes*: **`NonBinaryValue.java`** and **`IllegalParityValue.java`**.

[2] **Hint**: You could use the **`Scanner`** class to achieve this. Ideally, your new program should also display user-friendly messages when prompting the user for the inputs.

3. Chromosome manipulations are often used in Genetic Algorithms[3] for optimisation and search problems (e.g., in *network routing* or in the artificial intelligence's *n-queens problem*). While you will not be solving an optimisation problem, you will be writing part of the basic building block for using genetic algorithms. Write a Java class called **`Chromosome`** that satisfies the following requirements:

   i) It has a constructor that initialises the **`chromosomeArray`**. This constructor should take in an array of **`int`** values. It should also reuse the **`NonBinaryValue`** exception written for *Q1* of this lab sheet such that: if any value in the array is not a **`0`** or a **`1`**, then it should throw the exception and set all the values of **`chromosomeArray`** to be **`1`**.

   ii) It has a **`getFitness()`** method which should return the number of **`1`**s in the **`chromosomeArray`**.

   iii) It has a **`toString()`** method that formats the **`int chromosomeArray`** for printing purposes; the array's contents should be printed as follows: **`[1 0 1 0 1 1 1]`**.

   iv) It has a **`main()`** method that creates two *chromosomes*: one with the **`int`** array **`{1,0,1,1,1,0,1}`** and another with the **`int`** array **`{3,5,2,2}`**. The **`main()`** method should catch any exceptions thrown, print off the *stack trace* and exit.

   v) The class contains *Javadoc* comments, which are also generated.

   Your program should be named **`Chromosome.java`**

Ensure all your programs contain both internal comments and *Javadoc* comments.

[3] **Note**: See e.g., https://en.wikipedia.org/wiki/Genetic_algorithm.