© Manfred Kerber
Alexandros Evangelidis

School of Computer Science
University of Birmingham
29 October 2019

Extra Work 2

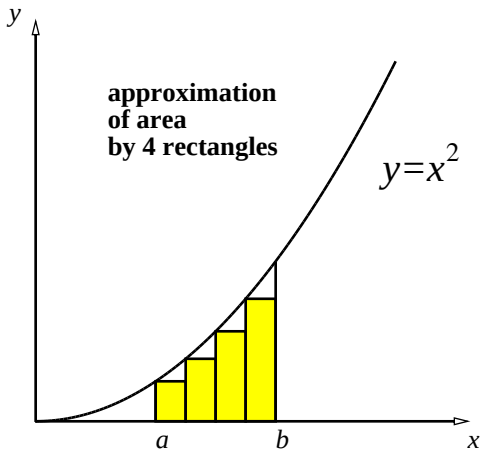
MSc/ICY Software Workshop

Exercise 1: (Basic, 30%) Let the `BankAccount` class from the examples of Week 2 be given (that is, a class with the three fields `private int accountNumber`, `private String accountName`, and `private int balance` and the corresponding constructor and getters and setters). Furthermore assume that the bank stores all the bank accounts in a variable `allAccounts` of type `ArrayList<BankAccount>`. Write a function `public static final Function<ArrayList<BankAccount>,Integer> totalAmount` that computes for an `ArrayList<BankAccount>` the total amount of money in all the bank accounts.
For instance with the two bank accounts
`BankAccount mary = new BankAccount(1, "Mary");` with `mary.payIn(100)` and
`BankAccount john = new BankAccount(2, "John");` with `john.payIn(99)` and
`BankAccount[] array = {mary, john};`
`ArrayList<BankAccount> allAccounts = new ArrayList<BankAccount>(Arrays.asList(array));`
the function call `totalAmount.apply(allAccounts)` should result in `199`.

Exercise 2: (Medium, 30%)
Definite integration means to determine the area under a function f on a particular interval $[a, b]$. It can be numerically approximated by the combined area of a number n of rectangles of equal width and of heights determined by f. Concretely, the area is the sum of the n rectangles of width $(b-a)/n$ where a rectangle i with $0 \leq i < n$ has a height given by $f(a + i \cdot (b-a)/n)$. To the left you see an example with the function f being the square function and an approximation with 4 rectangles. That is, the area is approximated, with $\delta = (b-a)/4$, by

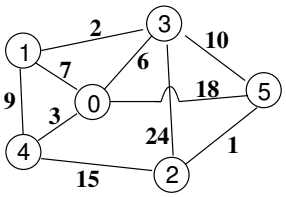
$$\delta \cdot \Big(f(a) + f(a+\delta) + f(a+2\cdot\delta) + f(a+3\cdot\delta) \Big)$$

With a bigger number of rectangles (e.g., $n = 1000$) you should get a better approximation.



Write in a class `Integral` a method `public static double integral(Function<Double,Double> f, double a, double b, int n)` that approximates the integral of an arbitrary function `f` on the interval `[a, b]` (with `a <= b`) by the sum of the areas of a positive number `n` of rectangles.
Test your method with four functions: $f(x) = x^2$, $f(x) = \sin(x)$, $f(x) = e^x$, and $f(x) = x^3 + x^2 + x + 1$ on the interval $[0, 1]$ with $n = 1000$. The results should approximately be 0.33283349999999995, 0.45927692033131423, 1.7174228307349666, and 2.0818337500000004, respectively.

Exercise 3: (Advanced, 30%) Graphs can be used in different ways, for instance, in order to describe the streets in a particular area. See the following example (graph to the left, corresponding two-dimensional array, the so-called adjacency matrix, to the right):




```
int[][] adjacencyMatrix =
 {{ 0, 7,-1, 6, 3,18},
  { 7, 0,-1, 2, 9,-1},
  {-1,-1, 0,24,15, 1},
  { 6, 2,24, 0,-1,10},
  { 3, 9,15,-1, 0,-1},
  {18,-1, 1,10,-1, 0}}
```

The adjacency matrix contains the information of the lengths of the direct routes between any two vertices. Obviously, the distance from a vertex to itself is `0`. If two vertices are not directly connected we record this by a `-1`.
Let an interface **`GraphInterface`** be given with the two method headers **`public boolean connected();`**, which is to compute whether a graph is connected, and **`public int totalLength();`**, which is to return the total length corresponding to the values in the graph.

Give an implementation of the interface in a class **`Graph`** with the field variables **`private int numberOfNodes`** and **`private int[][] adjacencyMatrix`**. In addition to the standard constructor, there should also be a constructor **`public Graph(String filename)`** that reads in a file that contains data of the form as displayed to the right (the example represents the graph above). That is, the first number defines the number of nodes in the graph, all other lines consist of the lines of integers corresponding to the adjacency matrix.

```
6
 0  7 -1  6  3 18
 7  0 -1  2  9 -1
-1 -1  0 24 15  1
 6  2 24  0 -1 10
 3  9 15 -1  0 -1
18 -1  1 10 -1  0
```

In order to determine whether a graph is connected you may use the algorithm described on `https://en.wikipedia.org/wiki/Connectivity_(graph_theory)#Computational_aspects`:

- Begin at any arbitrary node of the graph, `G`.
- Proceed from that node using either depth-first or breadth-first search, counting all nodes reached.
- Once the graph has been entirely traversed, if the number of nodes counted is equal to the number of nodes of `G`, the graph is connected; otherwise it is disconnected.

Write a main method and check whether the graph above is connected (it is) and compute its total length (the answer should be `95`).

Exercise 4: (Debugging, 10%) A class **`Student`** has the field variables **`private String registrationNumber`** and **`private int[] marks`**. The mark array has a size of 11. Its elements represent (in this order) 7 assessed worksheets (worth 3, 3, 3, 3, 1, 1, and 1 percent of the total module mark, respectively), 2 in-class tests (worth 3 and 2 percent of the total module mark), a team project (worth 10 percent of the total module mark), and an examination result (worth 70 percent of the total module mark). Somebody tries to write a corresponding class, in particular to compute the mark by a method **`public double totalMark()`** rounded to one decimal place. If a mark of **`-1`** has been entered for one piece of assessment, then this is to mean that the student has been granted for this piece of assessment extenuating circumstances and that the mark should be discarded from the computation of the total mark. If marks with a total weight of more than 50 percent of the total mark have been waived, then the returned total mark should be **`-1`**.
Finally, write a method **`public boolean passed()`** that returns **`true`** if the total mark of the student is greater than or equal to 50 and false otherwise.
In the case of a total mark of **`-1`**, the method **`passed()`** should throw an **`IllegalArgumentException`**.

```
/**
 *  The class contains a method totalMark() that determines the total
 *  mark from a number of component marks that come with a particular
 *  weight each. In case that a student was excused for a particular
 *  assignment (indicated by a -1 entry) the corresponding assignment
 *  is taken out of the computation of the total mark. However, if
 *  more than half of the marks have been waived, a total mark of -1
 *  is returned in order to indicate that the mark cannot be
 *  determined. Correspondingly there is a method passed() that
 *  determines whether a student has passed, that is, has a total mark
 *  of at least 50. (In case that the total mark cannot be determined,
 *  an IllegalArgumentException is thrown.)
 */
public class Student {
```

```
private String registrationNumber;
private int[] marks;
//Static constant for the weights of the assignments
public static final int[] weights = {3,3,3,3,1,1,1,3,2,10,70};
/**
 *  The constructor initializes the fields
 *  @param registrationNumber The registration number of the student.
 *  @param marks The array contain the 11 component marks.
 */
public Student(String registrationNumber, int[] marks) {
    this.registrationNumber = registrationNumber;
    this.marks = marks;
}
/**
 *  Getter for the registrationNumber.
 *  @return The registration number of the student.
 */
public String getRegistrationNumber() {
    return registrationNumber;
}
/**
 *  Getter for the marks.
 *  @return The marks of the student.
 */
public int[] getMarks() {
    return marks;
}
/**
 *  Setter for the registrationNumber.
 *  @param registrationNumber The new registration number of the student.
 */
public void setRegistrationNumber(String registrationNumber) {
    this.registrationNumber = registrationNumber;
}
/**
 *  Setter for the marks.
 *  @param marks The new marks array of the student.
 */
public void setMarks(int[] marks) {
    this.marks = marks;
}
/**
 *  Setter for a single mark.
 *  @param assignmentNumber The number of the assignment as a
 *  number between 1 and 11.
 *  @param mark The new mark for the assignment with the given
 *  assignmentNumber.
 */
public void setAssignmentMark(int assignmentNumber, int mark) {
    this.getMarks()[assignmentNumber-1] = mark;
}
/**
 *  Method to compute the total module mark of a student.
 *  @return The total mark for a student is computed from the 11
 *  component marks that come with a particular weight each
 *  (weighted average) and returned. The total number is rounded
 *  to one digit after the decimal point. In case that a student
 *  was excused for a particular assignment (indicated by a -1
 *  entry) the corresponding assignment is taken out of the
 *  computation of the total mark. However, if more than half of
 *  the marks (measured by weight) have been waived, a total mark of -1
 *  is returned in order to indicate that the mark cannot be determined.
```

```
     */
    public double totalMark() {
        int marksAchieved = 0;
        int totalWeight = 0;
        /* The method iterates over all component marks and if one is
         * not waived (that is, not -1) then it is multiplied by its
         * weight and added to the total of marks achieved;
         * furthermore the weight is added to the totalWeight in this case.
         */
        for (int i = 0; i <= weights.length; i++) {
            if (getMarks()[i] !=-1) {
                marksAchieved += getMarks()[i] * weights[i];
                totalWeight += weights[i];
            }
        }
        /* If the weight of all marks going into the computation is
         * below 50, a -1 if the total mark cannot be determined.
         */
        if (totalWeight <= 50) {
            return -1;
        } else {
            /* The total mark is computed as the marks achieved
             * divided by the total weight, rounded one decimal place.
             */
            return Math.round(marksAchieved * 10.0/totalWeight)/10.0;
        }
    }
    /**
     *  Method to compute whether a student has passed the module.
     *  @return true if the student has a mark of at least 50, false
     *  else.
     *  @exception IllegalArgumentException if the student has not
     *  attempted at least 50% of the assignements measured by their weights.
     */
    public boolean passed() {
        if (totalMark() == -1) {
            throw new IllegalArgumentException();
        }
        return totalMark() == 50;
    }
    /**
     *  toString method for a student object.
     *  @return A human readable String for a student object
     *  containing the registration number, the component mark, the
     *  total mark, and an indication of whether the student has passed.
     */
    public String toString() {
        String result = getRegistrationNumber();
        // Each component mark is added to the result
        for (int el : getMarks()) {
            result += " " + el;
        }
        result += " Total: " + totalMark();
        // Possible exception from method passed caught.
        if (passed()) {
            return result + " PASSED";
        } else {
            return result + " FAILED";
        }
    }
}
```