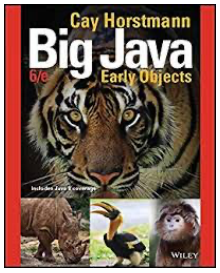
University of Windsor

FALL 2020: FINAL EXAM (Take-Home)

COMP-2120 Object-Oriented Programming Using Java

Wednesday, December 16, 2020

School of Computer Science
University of Windsor



READ THE FOLLOWING INSTRUCTION BEFORE GOING TO THE QUESTIONS

- The time limit is 3 hours.
- Before starting this exam, make sure that you download and complete the SignaturePage file, and include it with your Java files at the time of submission on Blackboard.

- For every problem:
  - You should download its corresponding Java tester program to test your code with it.
  - Use an IDE on your local computer. **DO NOT USE ONLINE COMPILERS OR NOMACHINE.**
  - **THOROUGHLY AND CAREFULLY READ THE DESCRIPTION OF THE QUESTION.**
  - Develop the required Java class by creating a separate java file for each required class.
  - **Note that this course is about developing Object-Oriented programming and not just writing a procedural code that you do in a non-object-oriented programming language like C. Therefore, you must create required Java classes for the concepts you have been asked for. Otherwise, you will lose the significant part of the mark assigned to the problem.**
  - Compile your java file(s). Then using the corresponding tester class provided, test your code. Compare the output of your program with the one provided. If it is not similar to it, go back and fix your code and test it again.
  - After making sure about the correctness of your code, in terms of compile-time, and fatal and non-fatal run-time errors, include the Java files into your final submission.
- **If a Java file has compile-errors, you will lose at least half of the original mark.**
- **Do not use any non-standard libraries.**
- **Do not alter content of the tester files.**
- **When you are done, SUBMIT the Java files, INDIVIDUALLY, on Blackboard, along with the completed SignaturePage file.**
- **Do not submit compiled files, otherwise you will receive zero for the problem.**
- **You should submit your files before the deadline. <u>No late submission accepted.</u>**

<u>GOOD LUCK!</u>

Problem 1 (60 marks)

A polynomial is an expression consisting of a variable, coefficients and powers. For instance

$$p(x) = 3x^4 + x^2 - 4x + 5$$

is a polynomial. We can see a polynomial as a list of terms. A term contains an integer as the power of variable x, and an integer as the coefficient.

Develop a class **`Polynomial`** that stores a polynomial using its terms as some pairs of two integers for power and coefficient, using Map interface. For instance, we can see the above polynomial as

$$p(x) = (4,3), (2,1), (1,-4), (0,5)$$

Note that you must store the polynomial in its descending order, i.e., start from the term with the highest power to the term with the lowest power.

Provide required **instance variables.** (**3 marks**)

Provide the following methods:

- (**2 marks**) **Default constructor**: It creates an empty **`polynomial`**.
- (**5 marks**) **Second constructor**: It receives an **`integer`** and a **`double`**, and creates its corresponding single-term polynomial.
- (**10 marks**) **`add`**: It receives a **`Polynomial`** object and adds it to the existing polynomial.
- (**8 marks**) **`subtract`**: It receives a **`Polynomial`** object and subtracts it from the existing **`polynomial`**. This method should somehow use the method **`add`**, defined above, to conduct polynomial subtraction.
- (**10 marks**) **`multiply`**: It receives a **`Polynomial`** object and returns a **`Polynomial`** object which is the multiplication of the existing **`polynomial`** and the parameter.
- (**3 marks**) **`degree`**: It returns the degree of the **`Polynomial`** object. To have an efficient way of finding the degree of a **`polynomial`**, you must define an instance variable, **`degree`**, for the **`Polynomial`** class, and keep updating its value for the **`Polynomial`** objects whenever needed.
- (**4 marks**) **`coefficient`**: It receives an **`integer`** (power) and returns its corresponding coefficient in the **`Polynomial`** object.
- (**4 marks**) **`evaluate`**: It receives a **`double`** value and evaluates the **`Polynomial`** object using it.

Override the following methods:

- (**4 marks**) **`equals`**: Two **`Polynomial`** objects are equal if they have the same number of terms, with the same powers and corresponding coefficients.
- (**5 marks**) **`toString`**: It should return a **`string`** representation of the **`Polynomial`** object. Note that if the coefficient of a term is **`1`**, or if the power of a term is **`1`**, you should not print them. For instance, for the above polynomial, it should return

  P(x) = 3x4 + x2 – 4x + 5

Note 2: If a negative value has been sent for the power, your program should not accept it. This means, you must properly handle the exception by showing a message, skip the operation, and send back the control to the caller. The execution of the program should not be terminated in this case. You can simply throw the **`IllegalArgumentException`** in this case and catch it properly. (**2 marks**)

The tester class, **`PolynomialTester.java`**, has been provided to help you for the development as well as executing and comparing the expected and actual outputs. The expected outputs of the **`PolynomialTester`** class using your code for **`Polynomial.java`** should be similar to the lines on the next page.

```
Creating P1(x) with (-10,0)...
P1(x) =  - 10
Adding (-1,1) to P1(x)...
P1(x) =  - x - 10
Adding (9,7) to P1(x)...
P1(x) = 9x7 - x - 10
Adding (5,1) to P1(x)...
P1(x) = 9x7 + 4x - 10
Creating P2(x) with (3,-2)...
java.lang.IllegalArgumentException: Power of a term can't be negative.
P2(x) =
Creating P2(x) with (3,2)...
P2(x) = 3x2
Adding (2,1) to P2(x)...
P2(x) = 3x2 + 2x
Adding (2,0) to P2(x)...
P2(x) = 3x2 + 2x + 2
Adding P2(x) to P1(x)...
P1(x) = P1(x) + P2(x) = 9x7 + 3x2 + 6x - 8
Subtracting P2(x) from P1(x)...
P1(x) = P1(x) - P2(x) = 9x7 + 4x - 10
Creating Q(x) with multiplying P1(x) by P2(x)...
Q(x) = P1(x) * P2(x) = 27x9 + 18x8 + 18x7 + 12x3 - 22x2 - 12x - 20
Degree of P1(x) = 7
Coefficient of x in P1(x) = 4
P2(3) = 35
P1(x) = 9x7 + 4x - 10
P2(x) = 3x2 + 2x + 2
P1(x) is not equal to P2(x)
Creating P3(x) with (3,2), adding (2,1), and adding (2,0)...
P3(x) = 3x2 + 2x + 2
P2(x) = 3x2 + 2x + 2
P3(x) is equal to P2(x)
```

ONLY SUBMIT the file Polynomial.java as the answer to this problem.

Do NOT ALTER the PolynomialTester.java file and do NOT SUBMIT it.

Problem 2 (40 marks)

An airport has a runway for planes landing and take-off. When the runway is busy, planes wishing to take-off or land have to wait. Landing planes get priority, and if the runway is available, it can be used.

Implement a Java class, **`Airport.java`**, for this simulation, using two appropriate lists, one for the planes waiting to take-off and one for those waiting to land. Note that the data structures you select for the two lists must be suitable for this purpose. For instance, the sooner a plane comes for landing, the sooner the plane will land. Also, you should keep the record of all the planes that have already landed or taken-off in one single list, to print out the activity log whenever asked, such that the sooner a plane landed, the later it shows in the printout. To get a clear idea, please have a close look at the expected outputs of the execution of the tester class provided.
The user enters the following commands: (The user entry has already been done in the tester class)

- ***t*** *flight-number* (for taking off the plane with the flight number *flight-number*)
- ***l*** *flight-number* (for landing the plane with the flight number *flight-number*)
- ***n*** (for conducting the next possible landing or taking off operation)
- ***p*** (for printing the current status of the two queues)
- ***g*** (for printing the list of the planes already taken-off or landed)
- ***q*** (to quit the program)

The first two commands, ***t*** and ***l***, place the indicated flight in the take-off or landing queue, respectively. Command ***n*** will conduct the current take-off or landing, print the action (take-off or land) and the *flight-number*, and enable the next one. Command ***p*** will print out the current content of the two queues. Command ***g*** will print out the list of all the planes that have already taken-off or landed. Command ***q*** will quit the program.

Provide required **instance variables.** (**3 marks**)

Provide the following methods in this class:

- (**3 marks**) The **`Airport`** class must have one default constructor to initialize the class instance variables.
- (**4 marks**) **`addTakeOff`**: It receives the *flight-number* as a **`String`** and updates the corresponding list.
- (**4 marks**) **`addLanding`**: It receives the *flight-number* as a **`String`** and updates the corresponding list.
- (**10 marks**) **`handleNextAction`**: It checks the landing and take-off lists and conducts one single operation as described above. It also returns a **`String`** showing the conducting operation, as you can see in the expected output.
- (**8 marks**) **`waitingPlanes`**: It returns a **`String`** showing the list of all the waiting planes for landing/take-off, as you can see in the expected output.
- (**8 marks**) **`log`**: It returns a **`String`** showing the list of all the operations already conducted, as you can see in the expected output.

The tester class, **`AirportTester.java`**, has been provided to help you for the development as well as executing and comparing the expected and actual outputs. The expected outputs of the **`AirportTester`** class using your code for **`Airport.java`** should be similar to the lines on the next page. Texts with `green` color are the user inputs.

```
Runway Simulator Menu
---------------------------------------------
> (l) to add a plane for landing, followed by the flight symbol
> (t) to add a plane for take-off, followed by the flight symbol
> (n) to perform next action
> (p) to print the planes waiting for landing/take-off.
> (g) print the planes already landed/taken-off.
> (q) quit to quit simulation.

> g
No activity exists.
> p
No plane is in the landing and take-off queues.
> n
No plane is waiting to land or take-off.
> t AA123
> l DA456
> l VA789
> n
Flight DA456 is landing.
> n
Flight VA789 is landing.
> g
List of the landing/take-off activities
---------------------------------------
Flight DA456 landed.
Flight VA789 landed.

> p
Planes waiting for take-off
---------------------------
AA123

> t KT429
> l SP333
> n
Flight SP333 is landing.
> n
Flight AA123 is taking off.
> n
Flight KT429 is taking off.
> g
List of the landing/take-off activities
---------------------------------------
Flight DA456 landed.
Flight VA789 landed.
Flight SP333 landed.
Flight AA123 taken-off.
Flight KT429 taken-off.

> p
No plane is in the landing and take-off queues.
> q
```

ONLY SUBMIT the file Airport.java as the answer to this problem.
Do NOT ALTER the AirportTester.java file and do NOT SUBMIT it.