# Summer 2022: Final Exam

**COMP-2120 Object-Oriented Programming Using Java**

**Thursday, August 18, 2022**

**School of Computer Science**

**University of Windsor**

**<u>READ THE FOLLOWING INSTRUCTION BEFORE ANSWERING THE QUESTIONS</u>**

- The time limit is 3 hours. The exam is out of 100 and will be 50% of the final exam.
- Before starting the exam, make sure that you download and complete the Signature page file, if you haven't done so, and include it with your Java files at the time of submission on Blackboard.
- Use an IDE on your local computer. **I recommend you NOT USE ONLINE COMPILERS OR NOMACHINE.**
- **Note that this course is about developing Object-Oriented programming and not just writing procedural code with a non-object-oriented programming language like C. Therefore, you must create the required Java classes for the concepts you have been asked for. Otherwise, you will lose a significant part of the mark assigned to each question.**
- For every question:
  - You should download its corresponding Java tester program to test your code with it.
  - **FIRST, THOROUGHLY AND CAREFULLY READ THE DESCRIPTION OF THE QUESTION.**
  - Develop the required Java class(es) by creating a separate java file for each required class.
  - Compile your java file(s), and make sure your code has no compile errors.
  - 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 in your final submission.
- **If a Java file has one or more compile errors, you will lose at least half of the original mark.**
- **Do not use any non-standard libraries.**
- **Do not alter the content of the tester files.**
- **SUBMIT the Java files on Blackboard along with the completed Signature page file.**
- **Do not submit compiled files. Otherwise, you will receive zero for the corresponding question.**
- **You should submit your files before the deadline.** **<u>No late submission will be accepted.</u>**

**<u>GOOD LUCK!</u>**

**Question 1 (70 marks)**

**Part 1:**
A polynomial is an expression consisting of a variable along with some terms including 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 a **non-negative 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 the Map interface**. For instance, we can see the above polynomial as

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

**Note**: 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. For doing this, you can use the method **`reverseOrder()`** from the **`Collections`** utility class as the argument for the map you declare in the **`Polynomial`** class.

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

Provide the following methods:

- (**2 marks**) **Default constructor**: It creates an empty **`polynomial`**. For an empty **`polynomial`**, you can set its degree to -1.
- (**3 marks**) **Second constructor**: It receives two **`integers`**, power and coefficient, and creates its corresponding polynomial that has only one term.
- (**4 marks**) **Third constructor**: It receives a map of powers and coefficients and creates its corresponding polynomial.
- (**3 marks**) **A copy constructor**: It receives a **`polynomial`** object and creates a deep (separate) copy of it.
- (**5 marks**) **`add`**: It receives a **`Polynomial`** object as a parameter and adds it to the existing **`polynomial`**, i.e., the implicit parameter.
- (**3 marks**) overloaded **`add`**: It is a **`static`** method that receives two **`Polynomial`** objects as parameters, adds them, stores them in another **`Polynomial`** object, and returns it.
- (**5 marks**) **`subtract`**: It receives a **`Polynomial`** object as a parameter and subtracts it from the existing **`polynomial`**, i.e., the implicit parameter. This method can use the **`add`** method, defined above, to conduct polynomial subtraction.
- (**3 marks**) overloaded **`subtract`**: It is a **`static`** method that receives two **`Polynomial`** objects as parameters, subtracts the second one from the first one, stores in another **`Polynomial`** object, and returns it.
- (**4 marks**) **`multiply`**: It receives a **`Polynomial`** object as a parameter and returns a **`Polynomial`** object which is the multiplication of the parameter and the implicit parameter.

- (**2 marks**) **`degree`**: It returns the degree of the **`Polynomial`** object, i.e., the implicit parameter. 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 whenever needed. The degree of a polynomial is the greatest power. For instance, the degree of the polynomial above is 4.

- (**3 marks**) **`coefficient`**: It receives an **`integer`** (power) as a parameter and returns its corresponding coefficient in the **`Polynomial`** object. For instance, for the polynomial above, if the method receives 2, it should return 1, which is the coefficient of the term with a power of 2.

- (**3 marks**) **`changeCoefficient`**: It receives two **`integer`**s, power and a new coefficient as parameters and changes the corresponding coefficient of the power given to the new coefficient in the **`Polynomial`** object. For instance, for the polynomial above, if the method receives 2 and 4, it should change the coefficient of $x^2$ from 1 to 4. If a term with the power given does not exist in the **`Polynomial`**, do nothing.

- (**2 marks**) **`removeTerm`**: It receives an **`integer`** (power) as a parameter and removes the term with the corresponding power from the **`Polynomial`**. If a term with that power does not exist, do nothing.

- (**3 marks**) **`evaluate`**: It receives a **`double`** value as a parameter and evaluates the **`Polynomial`** object using the value. For instance, if this method is called with 2 for the polynomial above, it should return 49. $(\, p(2) = 3 * 2^4 + 2^2 - 4 * 2 + 5 = 49.$

**Exception Handling**: If a negative value has been sent for the power for a given term, your program should not accept it. This means you must properly handle the exception by showing a message and ignoring that term. 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**)

Override the following standard methods inherited from the **`Object`** class:

- (**3 marks**) **`equals`**: Two **`Polynomial`** objects are equal if they have the same number of terms, with the same powers and corresponding coefficients.

- (**4 marks**) **`toString`**: It should return a **`string`** representation of the **`Polynomial`** object. Note that **<u>if the coefficient of a term is 1, or if the power of a term is 1, you should not print them</u>**. For instance, for the above polynomial, it should return: "P(x) = 3x4 + x2 – 4x + 5"

We would also like to be able to compare two polynomials as follows:

- First, make the Polynomial class **`Comparable`**, and then implement the required method based on the rules below: (**3 marks**)

  - For two polynomials $p(x)$ , as the implicit parameter, and $q(x)$ , as the explicit parameter:
    - The method should return 1 if $p(0) > q(0)$
    - The method should return -1 if $p(0) < q(0)$
    - The method should return 0 if $p(0) = q(0)$

  - For instance, p(x) = 3x4 + x2 – 4x + 5 > q(x) = 5x4 + 2x3 + 1 because p(0) = 5 > q(0) = 1

**Part 2:**

In this part, you must create another Java class, **Quadratic**, which is a subclass of **Polynomial**, with the following features:

- A quadratic polynomial is in the format of $q(x) = ax^2 + bx + c$ , in which $a, b, and\ c$ are integers.
- A quadratic polynomial has either two, one, or no roots, based on the value of $delta = b^2 - 4ac$.

Quadratic class has two extra instance variables, root1 and root2. (**1 mark**)

Provide the following methods:

- (**4 marks**) **Constructor**: It creates a **`quadratic polynomial with three coefficients`**. Note: In this constructor, you must use the existing public methods you have already developed in the superclass. Remember to calculate the roots of the quadratic as well.
- (**1 mark**) **Getter methods**: Two getter methods for the two roots.
- (**4 mark**) **roots**: This method should calculate the roots of the quadratic polynomial and update the corresponding instance variables. **Hint:** Return **true** if the quadratic has roots and **false** otherwise.

The tester class, **`PolynomialTester.java`**, has been provided to help you with the development as well as executing and comparing the expected and actual outputs.

The outputs of the **`PolynomialTester`** class using your code for **`Polynomial.java`** should be similar to those on the next page.

**ONLY SUBMIT Polynomial.java and Quadratic.java.**

**DO NOT ALTER PolynomialTester.java and DO NOT SUBMIT it.**

```
Creating P1(x) with the terms (0,-9), (2,-2), (6,8), and (4,6) ...

P1(x) = 8x6 + 6x4 - 2x2 - 9

Degree of P1(x) = 6

Coefficient of x^2 in P1(x) = -2

P1(x) after changing coefficient of term 6 and removing term 4

P1(x) = 3x6 - 2x2 - 9

Creating P2(x) with the terms (-2,5), (0,4), (2,1), and (1,3) ...

java.lang.IllegalArgumentException: Power of a term can't be negative. The term ignored.

P2(x) = x2 + 3x + 4

Create a copy of P2(x) ...

Copy of P2(x) = x2 + 3x + 4

Adding P2(x) to P1(x)...

P1(x) = P1(x) + P2(x) = 3x6 - x2 + 3x - 5

Subtracting P1(x) from P2(x) and store in P2(x)...

P2(x) = P2(x) - P1(x) =  - 3x6 + 2x2 + 9

Multiplying P1(x) by P2(x) and store it into Q(x) ...

Q(x) = P1(x) * P2(x) =  - 9x12 + 9x8 - 9x7 + 42x6 - 2x4 + 6x3 - 19x2 + 27x - 45

P2(5) = -46816.0

P1(x) = 3x6 - x2 + 3x - 5

P2(x) =  - 3x6 + 2x2 + 9

P1(x) is not equal to P2(x)

Add P1(x) and P2(x) and store it into P3(x) ...

P3(x) = x2 + 3x + 4

Subtracting P1(x) from P2(x) and store it into P4(x) ...

P4(x) =  - 6x6 + 3x2 - 3x + 14

P3(x) is less than P4(x)

Q(x) = 2x2 + 5x - 3

Roots of quadratic Q(x)=5x^2+10x+3: Root1=     0.500 , Root2=    -3.000

Q(x) = 10x2 + 5x + 3

This quadratic polynomial has no real roots. (Delta < 0)
```

**Question 2 (30 marks)**

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

Implement a Java class, **`Airport.java`**, for this simulation, using three appropriate lists as follows:

- One for the airplanes waiting to take off using the flight id as a String. The sooner a flight comes for take-off, the sooner it will take-off, **with considering the flight id as priority for taking off**. For example, if a flight with id "A11" comes after "A12", it will take-off first because "A11" is lexicographically comes before "A12" (FIFO with Priority).

- One for the airplanes waiting to land using the flight id as a String. The sooner an airplane comes for landing, the sooner it will land (FIFO).

- One for keeping the record of all the airplanes that have already landed or taken off to print out the activity log whenever asked, such that the later a flight added to this list, the sooner it comes out from the list (LIFO).

  **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-id* | (Take off the airplane with the flight id) |
| • ***l** flight-id* | (Land the airplane with the flight id) |
| • ***n*** | (Conduct the next possible landing or taking off operation) |
| • ***p*** | (Print the current status of the two queues) |
| • ***g*** | (Print list of the airplanes already taken off or landed) |
| • ***f*** | (Print list of the airplanes already taken off or landed into an output file) |
| • ***q*** | (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-id*, 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 airplanes 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 instance variables.

- (**3 marks**) **`addTakeOff`**: It receives a *flight-id* as a **`String`** and updates the corresponding list.

- (**3 marks**) **`addLanding`**: It receives a *flight-id* as a **`String`** and updates the corresponding list.

- (**6 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.

- (**5 marks**) **`waitingPlanes`**: It returns a **`String`** showing the list of all the waiting airplanes for landing/take-off, as you can see in the expected output.

- (**3 marks**) **`log`**: It returns a **`String`** showing the list of **all the operations already conducted**, as you can see in the expected output. Follow the following order: **The sooner an airplane landed/taken off, the later it must appear in the log.**

- (**4 marks**) Overloaded **`log`**: It receives the name of a text file as a **`String`** and writes the same output of the above **`log`** function into the text file instead of storing and returning the result as a String.

The tester class, **`AirportTester.java`**, has been provided to help you with 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.

**ONLY SUBMIT Airport.java.**

**DO NOT ALTER AirportTester.java and DO NOT SUBMIT it.**

```
Runway Simulator Menu
-----------------------------------------------
> (l) add a plane for landing, followed by the flight id
> (t) add a plane for take-off, followed by the flight id
> (n) perform next action
> (p) print the planes waiting for landing/take-off.
> (g) print the planes already landed/taken off to the screen
> (f) print the planes already landed/taken off to the output file
> (q) quit the 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
> t AA111
> l DA456
> l DA123
> n
Flight DA456 is landing.
> n
Flight DA123 is landing.
> g
List of the landing/take-off activities
---------------------------------------
Flight DA123 landed.
Flight DA456 landed.

> p
Planes waiting for take-off
---------------------------
AA111
AA123

> t AA122
> l SP333
> n
Flight SP333 is landing.
> n
Flight AA111 is taking off.
> n
Flight AA122 is taking off.
> n
Flight AA123 is taking off.
> g
List of the landing/take-off activities
---------------------------------------
Flight AA123 taken-off.
Flight AA122 taken-off.
Flight AA111 taken-off.
Flight SP333 landed.
Flight DA123 landed.
Flight DA456 landed.

> f
Writing the airport log to the file AirportLog.txt
Done.
> p
No plane is in the landing and take-off queues.
> q
```