# COMP 348: Principles of Programming Languages

# Assignment 1 on Java and Prolog

Summer 2021, sections AA and AB

May 8, 2021

## Contents

# 1 General Information

**Date posted:** Tuesday May 11$^{th}$, 2021.

**Date due:** Tuesday May 25$^{th}$, 2021, by 23:59[1].

**Weight:** 6% of the overall grade.

# 2 Introduction

This assignment targets two programming paradigms: 1) Object-Oriented and Functional Programming using Java, 2) Logical Programming with PROLOG.

# 3 Ground rules

You are allowed to work on a team of 3 students at most (including yourself). Each team should designate a leader who will submit the assignment electronically. See Submission Notes for the details.

ONLY one copy of the assignment is to be submitted by the team leader. Upon submission, you must book an appointment with the marker team and demo the assignment. All members of the team must be present during the demo to receive the credit. Failure to do so may result in zero credit.

This is an assessment exercise. You may not seek any assistance from others while expecting to receive credit. **You must work strictly within your team).** Failure to do so will result in penalties or no credit.

# 4 Your Assignment

Your assignment is given in two parts, as follows. 1) Object-Oriented and Functional Programming using Java, 2) Logical Programming with PROLOG.

[1] see Submission Notes

## 4.1 Object-Oriented and Functional Programming using Java

**Q 1.** Define the following classes and interfaces:

1. Shape (interface)

   - a method to return shape name. Provide a default implementation for this particular method that returns the Runtime class name.
   - a method to return shape's perimeter.
   - a method to return shape's area.

2. Rectangle (class, implementing Shape)

   - : the two sides as doubles; define the attributes and standard getters and setters.
   - : `toString()` implementation to return the shape name (by calling the corresponding method in `Shape` interface), followed by the two values for sides (separated by comma).
   - : A `static parse()` method that receives an input string and returns an instantiated `Rectangle` whose sides are initialized with the values in the input string. The input string is in comma separated format,i.e.:

     "`Rectangle,2,3.5`". The method returns the object as `Rectangle`.
   - Implement the corresponding perimeter and area methods of the `Shape` interface.

3. Circle (class, implementing Shape)

   - : radius as double; define the attribute and standard getter and setter.
   - : `toString()` implementation to return the shape name (by calling the corresponding method in `Shape` interface), followed by the value of the radius (separated by comma).
   - : A `static parse()` method that receives an input string and returns an instantiated `Circle` whose radius is initialized with the value in the input string. The input string is in comma separated format,i.e.:

     "`Circle,1`". The method returns the object as `Circle`.
   - Implement the corresponding perimeter and area methods of the `Shape` interface.
   - Re-implement the shape name method and make sure the returned name is in ALL-CAPS.

**Q 2.** Using the above, write a java program that:

- reads a file containing at least 10 shapes (each shape is provided in a single line, in comma separated format); `Arrays.stream` may be used to convert an array into stream.
- sorts and displays the shapes by shape name and area;
- sorts and displays the shapes by perimeter only;
- displays a summary information of averages per shapes;
- displays the average perimeter, average area, and the total number of shapes at the end.

**Implementation Requirements**

- You should strictly use the classes defined in 1. You may not defined additional classes.
- To implement the above functions, you must strictly use the stream API. Using loops such as `for`, `while`, etc. are not allowed.
- Use try-with-resources to open the file. The input file must be given by the user.
- You should strictly use functional programming and java API for sorting and displaying the shape objects. Use `Arrays.sort()` or `Collections.sort()`. No additional classes or interfaces are allowed. The Shape and its sub-classes may not implement the `comparable` interface.
- Use Java stream API to process the input file. You may use `String.spilt()` to transform the input lines as line array, and eventually into an array of shapes.
- Use Java stream API to display the summary information. No explicit loops may be used.
- Use at least one "method-reference" in your code.
- **IMPORTANT:** It is recommended not to use explicit `throws` clauses in method declarations. Throwing a `RuntimeException` might be useful. In any case, make sure all [runtime] exceptions are eventually caught in the `main()` method.

## 4.2 Logical Programming with PROLOG

### Fact Representation, Queries, Unification, and Resolution

**Q 3. Unification:** Indicate which of the following pairs of terms can be unified together? If they can't be unified, please provide the reason for it. In case of error, indicate the error. If they can be unified successfully, wherever relevant, provide the variable instantiations that lead to successful unification. (Note that '=' indicates unification)

1. `food(bread, X) = Food(Y, soup)`
2. `Bread = soup`
3. `Bread = Soup`
4. `food(bread, X, milk) = food(Y, salad, X)`
5. `manager(X) = Y`
6. `meal(healthyFood(bread), drink(milk)) = meal(X,Y)`
7. `meal(eat(Z), drink(milk)) = [X]`
8. `[eat(Z), drink(milk)] = [X, Y | Z]`
9. `f(X, t(b, c)) = f(l, t(Z, c))`
10. `ancestor(french(jean), B) = ancestor(A, scottish(joe))`
11. `meal(healthyFood(bread), Y) = meal(X, drink(water))`
12. `[H|T] = [a, b, c]`
13. `[H, T] = [a, b, c]`
14. `breakfast(healthyFood(bread), egg, milk) = breakfast(healthyFood(Y), Y, Z)`
15. `dinner(X, Y, Time) = dinner(jack, cook( egg, oil), Evening)`
16. `k(s(g), Y) = k(X, t(k))`
17. `equation(Z, f(x, 17, M), L*M, 17) = equation(C, f(D, D, y), C, E)`
18. `a(X, b(c, d), [H|T]) = a(X, b(c, X), b)`

**Q 4. Queries:** Assume we have the following database in a Prolog program:

```
course(heat_transfer, mechanical).
course(web_design, computer).
course(design_methods, fine-arts).
course(poetry, literature).
lab_number(mechanical,15).
lab_number(fine_arts,10).
lab_number(X, Z) :- course(X, Y), lab_number(Y, Z).
field(mechanical,engineering).
field(computer, engineering).
field(fine-arts, art).
field(literature, social).
field(management, business).
field(X, Y) :- course(X, Z), field(Z, Y).
student(john, heat_transfer).
student(jane, heat_transfer).
student(jake, poetry).
student(jeff, leadership).
student(alex, web_design).
student(eve, design_methods).
student(X, Y) :- field(Z, Y), student(X, Z).
student(X):- student(X,_).
```

Determine the type of each of the following queries (ground/non-ground), and explain what will Prolog respond for each of these queries (write all the steps of unifications and resolutions for each query)?

1. `? field(heat_transfer,engineering).`
2. `? lab_number(fine_arts,X).`
3. `? field(computer, literature).`
4. `? course(X,Y).`
5. `? student(jeff).`
6. `? student(john, engineering).`
7. `? student(X, engineering).`

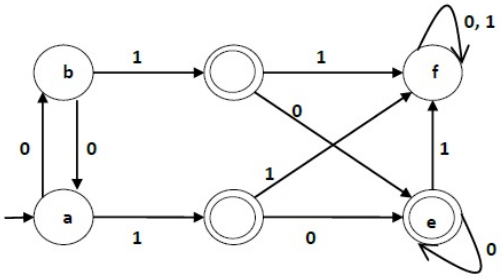
```
8. ?  student(X, fine-arts), course(fine_arts, Y).
9. ?  field(_, X).
10. ?  lab_number(_, X), field(X, Y).
11. ?  lab_number(X, 15), field(X, Y).
12. ?  student(X), !, student(X,_).  % note to cut here
13. ?  student(X), student(X,_), !.
14. ?  course(X,_), \+ student(_,X). % \+ is for negation (not)
```

**Q 5.** Provide a knowledge-base of clauses specifying you and your team's courses in PROLOG.

- In your database include your student information (name and id) as well as all courses that each member of the team is taking this semester. The courses include course name and course number.
- Write a query to return the list of courses taken by each member.
- Write a query to return the team size.
- Write a query to return the unique courses taken by the whole team.
- Use `sort/2` to sort the result of the previous query.
- Unify the expression `[A,B|C]` with the above result. Provide the values for $A$, $B$, and $C$.

## PROLOG Applications

**Q 6.** Given the Finite State Machine in the image below,



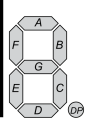

1. Represent the FSM in Prolog.

2. Write a Prolog query to determine whether the four sequences of "`0`", "`1`", "`0 1`", and "`1 0`" are accepted by the machine.

3. Run the queries and verify the answers.

**Q 7.** A 7-segment display, as illustrated in the following, may be used in electrical circuits to display a digit. As such, a Binary-Coded-Decimal (BCD) to 7-segment decoder may be used. The following diagram shows the truth table of such a decoder, where the input binary digit is represented by $A$, $B$, $C$, and $D$ and the output is represented by $a$-$g$.

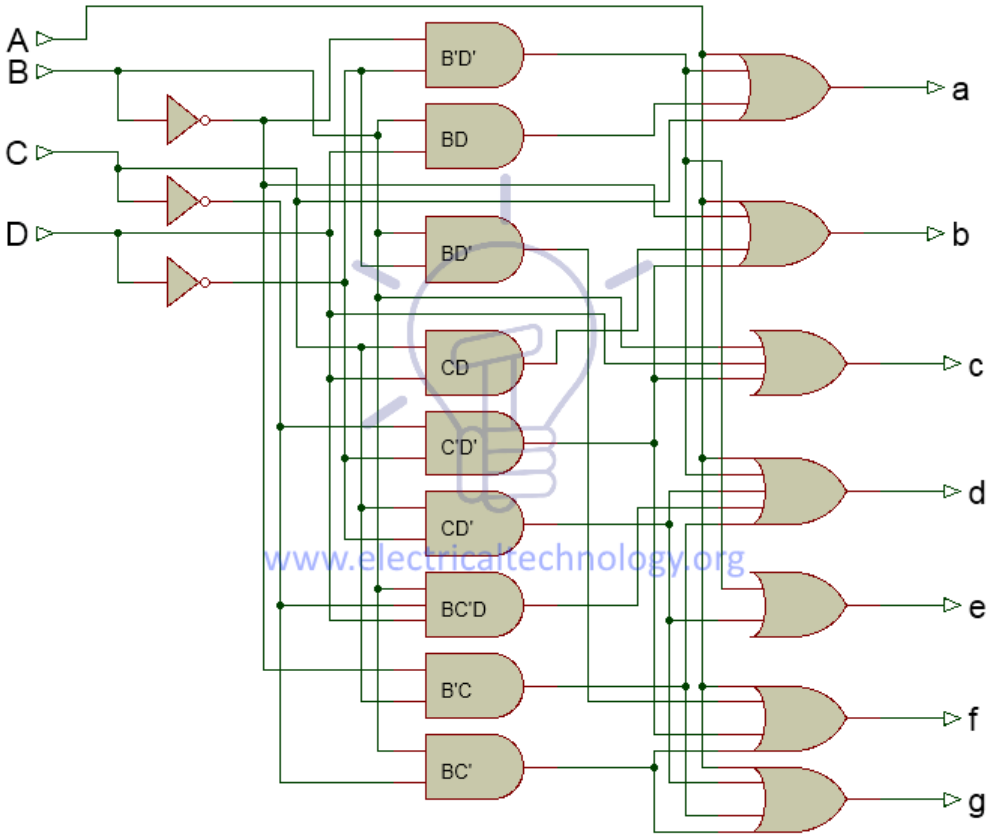
| Decimal Digit | Input lines | | | | Output lines | | | | | | | Display pattern |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | a | b | c | d | e | f | g | |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | |



The digital circuit of such a decoder is given in the following.

a) Implement the above circuit in PROLOG.

b) Write the query to calculate the outputs of the seven segments for the input 0101.




**Schematic of BCD to 7-Segment Decoder**

### Short Programs

**Q 8.** Write a prolog procedure `every-other/2` that receives a source list as its first argument and produces a list in the second argument whose elements are the elements of odd indexes in the source list.

Examples are given in the following:

```
?- every-other([], L)
L = [].
?- every-other([1], L)
L = [1].
?- every-other([1, 2], L)
L = [1].
?- every-other([1, 2, 3], L)
L = [1, 3].
?- every-other([1, 2, 3, 4], L)
L = [1, 3].
```

Make sure your procedure is efficiently terminated.

**Q 9.** Write a Prolog query with arity 2 to return the first $n$ numbers of a Lucas sequence in a list.

`https://brilliant.org/wiki/lucas-numbers/`

The Lucas sequence has the same recursive relationship as the Fibonacci sequence, where each term is the sum of the two previous terms, except that the first two numbers in the sequence are: 2, and 1. The first few elements of the sequence are: $2, 1, 3, 4, 7, 11, 18, ...$

# 5 What to Submit

The whole assignment is submitted by the due date under the corresponding assignment box. Your instructor will provide you with more details. It has to be completed by ALL members of the team in one submission file.

## Submission Notes

Clearly include the names and student IDs of all members of the team in the submission. Indicate the team leader.

IMPORTANT: You are allowed to work on a team of 3 students at most (including yourself). Any teams of 4 or more students will result in 0 marks for all team members. If your work on a team, ONLY one copy of the assignment is to be submitted. You must make sure that you upload the assignment to the correct assignment box on Moodle. No email submissions are accepted. Assignments uploaded to the wrong system, wrong folder, or submitted via email will be discarded and no resubmission will be allowed. Make sure you can access Moodle prior to the submission deadline. The deadline will not be extended.

Naming convention for uploaded file: Create one zip file, containing all needed files for your assignment using the following naming convention. The zip file should be called a#_studids, where # is the number of the assignment, and studids is the list of student ids of all team members, separated by (_). For example, for the first assignment, student `12345678` would submit a zip file named `a1_12345678.zip`. If you work on a team of two and your IDs are `12345678` and `34567890`, you would submit a zip file named `a1_12345678_34567890.zip`.
Submit your assignment electronically on Moodle based on the instruction given by your instructor as indicated above: `https://moodle.concordia.ca`
Please see course outline for submission rules and format, as well as for the required demo of the assignment. A working copy of the code and a sample output should be submitted for the tasks that require them. A text file with answers to the different tasks should be provided. Put it all in a file layout as explained below, archive it with any archiving and compressing utility, such as WinZip, WinRAR, tar, gzip, bzip2, or others. You must keep

a record of your submission confirmation. This is your proof of submission, which you may need should a submission problem arises.

‘

# 6 Grading Scheme

Q1 10 marks

Q2 25 marks

Q3 10 marks

Q4 10 marks

Q5 10 marks

Q6 5 marks

Q7 10 marks

Q8 10 marks

Q9 10 marks

**Total:** 100 marks.

# References


1. Java Streams: `https://www.baeldung.com/java-8-streams`
2. Using `Collectors.summarizingDouble`:

   `https://www.logicbig.com/how-to/code-snippets/jcode-java-8-streams-collectors-summarizingdouble.html`
3. Lambdas & Exceptions: `https://www.baeldung.com/java-lambda-exceptions`
4. SWI-PROLOG: `https://swish.swi-prolog.org`
5. PROLOG Sort: `https://www.swi-prolog.org/pldoc/man?predicate=sort/2`
6. Lucas Sequence: `https://brilliant.org/wiki/lucas-numbers/`
7. BCD-7-Segment Decoder:

   `https://www.electricaltechnology.org/2018/05/bcd-to-7-segment-display-decoder.html`