Concordia University

Department of Computer Science and Software Engineering

# COMP 6411-AA: Comparative Study of Programming Languages

## Assignment 1 on Prolog

**Dr. Constantinos Constantinides, P.Eng.**

constantinos.constantinides@concordia.ca

May 10, 2021

## 1 General information

**Date posted**: Monday, 10 May, 2021.

**Date due**: Friday 21 May, 2021, by 23:59.

**Weight**: 30% of the overall mark.

## 2 Introduction and ground rules

1. This is a team assignment. Each team should designate a leader who will submit the assignment electronically.

2. You may not seek any assistance while expecting to receive credit. You must work strictly within your team and seek no assistance for this project (from the instructor, the teaching assistants, fellow classmates and other teams or external help). Failure to do so will result in penalties or no credit.

3. **All team members are expected to work relatively equally on each part of the assignment. The team leader has the responsibility to ensure that the team does not violate this rule. Failure to do so will result in penalties. In your submission, you must include only the names of those people who contributed to the assignment. Accommodating someone who did not contribute will result in penalties to each team member and a heavier penalty for the team leader**.

4. If there is any problem in the team (such as lack of contribution, etc.), the team leader must contact the instructor as soon as the problem appears.

5. No late submissions will be accepted.

## 3 Problem statement

Consider the following Java program:

```
public interface Behavior {
  public String act();
  public String reason();
}

public class Human implements Behavior {
  public String type = "HUMAN.";
  public boolean empathy = true;
  public String act() {
    return "I am an human and I can act.";
  }
  public String reason() {
    return "I am a human and I can reason.";
  }
  public boolean hasEmpathy() {return empathy;}}

public class Bladerunner extends Human {
  public String type = "BLADERUNNER.";
  public String rank;
  Bladerunner(){}
  Bladerunner (String r) {
    this.rank = r;
    String rank = "OFFICER.";
    System.out.println(rank);
  }
  public String reason() {
    return "I am bladerunner and I can reason.";
  } }
```

```
public abstract class Machine implements Behavior {
  public static String type = "MACHINE.";
}

public class Android extends Machine {
  public int version;
  Android (int version) {
    this.version = version;
  }
  public String whatIhave() {
    return "I have physical power.";
  }
  public static String whatIneed() {
    return "I need more time.";
  }
  public String act() {
    return "I am an android and I can act.";
  }
  public String reason() {
    return "I am an android and I can reason.";
  }
}

public interface Behavior2 {
  public boolean empathy = true;
  public boolean memories = true;
  public boolean hasEmpathy();
  public boolean hasMemories();
}
```

```
public class Android2 extends Android implements Behavior2 {
  Android2 (int version) {
    super(version);
  }
  Android2() {
    super(8);
  }
  public String whatIhave() {
    return "I have an infinite time.";
  }
  public boolean hasEmpathy() {
    return empathy;
  }
  public boolean hasMemories() {
    return memories;
  }
}
```

# 4 Your assignment

Your assignment consists of the following activities:

## 4.1 Construct a declarative database

Transform the Java program into a declarative database (`facts.pl`), with the following **facts**:

- `class(Type)`: `Type` is defined as a class.
- `construction(Type, Signature)`: `Type` is instantiated by a constructor defines by `Signature`. In the case of a default constructor, use the keyword `default`. Consider the following example:

  ```
  ?- construction('Bladerunner', ConstructorSignature).
  ConstructorSignature = default ;
  ConstructorSignature = '(String)' ;
  false.
  ```

- `interface(Type)`: `Type` is defined as an interface.
- `defines(Type, FeatureName, Kind, AccessType)`: Class `Type` defines feature `FeatureName` with access control `AccessType`, where `Kind` can be either one of `method`, or `attribute`.
- `extends(Subclass, SuperClass)`: Class `Subclass` directly extends class `SuperClass`.
- `implements(Type, Interface)`: `Type` directly implements `Interface`.

## 4.2 Executing queries (12 pts)

Execute the following queries and record your interaction into a file:

1. What classes, if any, are present in the program?
2. What methods, if any, does class 'Human' define?
3. Does class 'Android' define an attribute named 'version'?
4. Does class 'Machine' define a method named 'hasMemories()'?
5. Which class, if any, extends class 'Human'?
6. Which type, if any, implements interface 'Behavior2'?

For each query, write down its type (ground, non-ground).

## 4.3 Extend the database with rules (48 pts)

Extend your database by preparing a separate file to contain all rules described below. The first line should contain the statement

```
:- consult(facts).
```

This will enable your rules file to read (import) your facts file while keeping the two separate.

1. `empty_class/1`: Succeeds when a type defines no features.
2. `lazy_class/1`: Succeeds when a type defines only one method.
3. `data_type/1`: Succeeds when a type defines attributes, but does not define any methods.
4. `child/1` Succeeds by finding a set of direct subtype-supertype pairs.
5. `child/2`: Succeeds when it finds a subtype relationship
6. `ancestor/2`: Succeeds when it finds all ancestors of a given type (Hint: Uses `child/2`).
7. `state_of/2`: Succeeds by obtaining the state of a given type. Recall that the state of a class consists of all attributes defined or inherited.
8. `interface_of/2` Succeeds when `List` contains a list of all messages (method calls) that make up the interface of class `Type`.
9. `siblings/1`: Succeeds by obtaining a pair of sibling types.
10. `instantiated_polymorphically /1`: Succeeds when a type is instantiated using polymorphism.
11. `root/1`: Succeeds by finding a type with no ancestors. Uses `is_type/1` which succeeds when a type is either a class or an interface.
12. `provides_interface/2`: Succeeds by obtaining a list of all classes that implement a given interface.

## 5 Expected output

```
?- empty_class(Empty).
false.


?- lazy_class(Lazy).
Lazy = 'Bladerunner' ;
false.


?- data_type(Type).
Type = 'Machine' ;
false.


?- child(Set).
Set = [['Bladerunner', 'Human'], ['Android', 'Machine'],
       ['Android2', 'Android'], ['Human', 'Behavior'],
       ['Machine', 'Behavior']] .


?- child('Bladerunner', 'Machine').
false.


?- child('Android', Parent).
Parent = 'Machine' ;
false.


?- ancestor(A, 'Android2').
A = 'Android' ;
A = 'Behavior2' ;
A = 'Machine' ;
A = 'Behavior' ;
```

```
false.

?- state_of('Bladerunner', State).
State = [type, rank, type, empathy] .

?- interface_of('Android2', Interface).
Interface = ['whatIhave()', 'hasEmpathy()', hasMemories(),
             'whatIneed()', 'act()', 'reason()'] .

?- siblings(ListofSiblings).
ListofSiblings = [['Human', 'Machine'], ['Machine', 'Human']] .

?- instantiated_polymorphically('Human').
false.

?- root(R).
R = 'Behavior' ;
R = 'Behavior2' ;
false.

?- provides_interface('Behavior2', L).
L = ['Android2'] ;
false.

?- provides_interface('Behavior', L).
L = ['Human', 'Machine', 'Bladerunner', 'Android', 'Android2'] ;
false.
```

## 6 What to submit

You must submit a zip file containing the following two files:

1. File `README.txt` that contains the names and id's of all contributing members of your team.
2. A text file (named after your team, e.g. `team1-interaction.txt`), containing the interaction with the system while executing the queries of Section 4.2.
3. A prolog file (named after your team, e.g. `team1-rules.pl`) with your rules defined in 4.3.

Name the zip file after your team e.g. `team1.zip` and submit it at the Electronic Assignment Submission portal

(`https://fis.encs.concordia.ca/eas`)

under **Programming Assignment 1**.

**END OF ASSIGNMENT**

---