CS2510 Assignment

Stefan Rudvin 51549217

Simplicity

In terms of pure simplicity Prolog takes the cake on this one. Through procedural programming and a single data type, small programs tend to be much more simple in Prolog. The term is also untyped so it can unify with other term types as needed. This makes code simpler to write (convention over configuration) - Java forces the user to specify the return and input type of methods, which makes collaboration simpler. The object oriented nature of Java forces users to use classes, which automatically adds some lines of code and can be confusing for newcomers. These features become more useful when programs grow in size.

Due to these reasons my experience with Prolog was much simpler, and I found it easier to follow the recommended conventions of the language. To be 'properly' OOP I decided to create multiple classes for Java – as each class should only have one task -, which took me longer to write and made me consider access modifiers and sending messages in between classes. I could essentially have thrown all the functions in a single class and called it a day, but I feel like this would have fought against the OOP nature of the language itself.

Orthogonality

Orthogonality is a property of a language where the operations do not affect parts of the system that they are not involved in, where everything is 'contained' so to speak. This helps even large and complex systems work seamlessly through tens or hundreds of method calls. Java provides orthogonality through access modifiers.
Prolog is considered more orthogonal due to its single data type, term, that is used to build other types of data such as atoms, variables and numbers. However, due to its nonmonotonic nature logic, facts and rules are dynamic and can be changed at any time. With the assert() and retract() functors, the program itself can be changed at runtime. This makes the language a little less orthogonal since adding new rules can essentially break the code at runtime.

When writing the code, I did not have issues about orthogonality with either language – this is probably because both projects were very small. In my Java project I did have to consider how the methods in the user class would affect e.g. the books in another class- a simple example being making the method for adding books not static. For larger projects I feel like Java would be more Orthogonal since writing classes with specific scopes makes it easier for classes to not mess with each other.

Data types

As mentioned before, all of Prologue's data types are based on the term data type. In contrast Java provides eight primitive data types (bool, byte, char, short, int, long, float, double) along with non-primitive data types. To write the bookstore code, the availability of Java's data types gave me much more flexibility in writing code. I had the option of using dictionaries or arrays to store the books for each customer. In Prolog I used facts to add books to the buy functor. Prolog does support lists too (which are represented internally as trees) which I could have used but decided not to due to their relative complexity. The problem with functors is them being very un-DRY, as

the functor must be repeated when each book is added. I did not see it as a problem with this project, but with a bookstore with hundreds or thousands of books my approach to the problem would be about as useful as building a wall to stop immigrants from entering a country. Another solution to the problem would have been to read a file with a list of books and customers (probably how things would work in the real life), but Prolog is notorious for poor I/O management and it seemed a tad excessive for a project this size.

Syntax Design

Java and Prologue's syntaxes vary wildly since they come from very different backgrounds. Java's syntax comes mainly from C++ with an object oriented twist. Classes are required for all code and everything is regarded as an object that can be referenced if the access modifiers allow it. Java does not support multiple inheritance, although it can be achieved through interfaces. With a general computing science background, I found Java's syntax easy to use and simple. This obviously comes with a fair bit of bias too since Java is based on C++ after all.

Prologue's syntax is divided into facts, rules and queries. As it uses the resolution algorithm, things must be clearly labelled: atoms must begin with a lowercase letter and variables must begin with a capital letter or an underscore symbol. I found this useful since I knew exactly what type an object was, but coming from Object Oriented Languages it was weird to be limited by something as simple as capitalization.

https://en.wikipedia.org/wiki/Java_(programming_language)

Support for abstraction

While I did not specifically use abstraction in either of the programs, as far as I know Java has better support for abstraction through OOP principles – polymorphism & interfaces. Vanilla Prologue's support for abstraction is more limited, with more options available through expansions.

Exception handling

Java provides a great exception hierarchy, where all exception classes are subtypes of the java.lang.Exception class1. The class provides users with multiple methods to manage exceptions i.e. get details of the exception and catch them appropriately. Prolog also provides similar functionality with catches and throws. Once again I did not use exception handling in my code, but its cool to know that it is available. Most of the logic in my code was fairly straight forward and did not include things like opening files, visiting a URL or fetching data where exception handling is generally required.

https://www.tutorialspoint.com/java/java_exceptions.htm

Expressivity

Much of the expressivity of both languages boils down to the background of the programmer. A coder coming from a heavy mathematical, formal logic background would find Prolog much easier to understand than Java. Personally with a background in Python and PHP I found Java to be more expressive – easier to understand and write. It shares the syntax and the way the code reads is essentially the same. From a completely objective point of view, Prolog is more expressive due to less code needing to be written and the code being easier to read. However, its major difference to other programming languages is the way you need to think – logic programming really is another can of beans for most programmers.

Type checking

As mentioned previously, Prolog is untyped, meaning that terms can be unified. This makes it simple to mess around and not worry about data types, but can make things confusing when working with many people. I found it both challenging and useful – in a way Prolog focuses in the logic of what's happening itself and does not bother itself with data types clashing and the likes.

I found Java's statically typed approach to be more specific and more academic so to speak. Coming from python it was weird specifying the type of each object, but it also helped me code better – in python I would use the toString() and int() methods to convert things around. In Java, I began to question what I was doing in an of itself and resorted to handling data types much better.