Robert Szkutak

As a programmer, every time I set out to solve a problem I am forced to determine a paradigm and choose a language with which to solve the problem. The logical paradigm of programming languages conceptually structures itself around the idea of creating rules and then querying those rules to answer a question. The object oriented paradigm of programming languages is structured around the concept of treating every element in of a program (including the program itself) as an object with a specific purpose. Solving a problem in a programming language based a logical paradigm is a completely different experience from solving the same problem in a language based upon an object oriented paradigm.

When solving a problem using a language such Prolog which is based upon a logical paradigm I first tried to understand the problem in terms of the paradigm. I created a first order logic representation of the problem which showed me frames and predicates defining how a solution could be derived in terms of the problem. I clearly defined what conditions make a property of an object true or false and what we could learn from these properties once evaluated. Because of it’s basis upon a logical paradigm, it is pleasantly trivial to interpret a first order logic representation of a problem into Prolog. Once finished, a user can easily directly ask a question to find out the properties of an object in the program and it’s relationship to other objects.

When solving a problem using a language such as Java which is based upon an object oriented paradigm I first decided what objects I would need to solve the problem. For this particular problem I knew I would need a class for courses which would define what a course is and it’s properties, a class for the course catalog which would define what conditions were needed for a course to be taken, a class for a student who is enrolling in classes and needs to know what courses from the catalog he may take, and a class for the program itself which would reveal to the user all the necessary information to solve the problem at hand. Once the objects which would be needed to solve the problem were defined I put them together like a puzzle within the main program itself. In my opinion, one of the benefits to object oriented programming is the ease with which complex code and functions can be used once they are defined. For example, it only took me a line or two of using one of my defined classes to have the class perform a complex search algorithm unique to each member of the class. Other programming paradigms would require complex functions with equally complex parameters which may sacrifice some of the function’s versatility and portability.

There were some similarities between programming in a logical language like Prolog and an object oriented language like Java. The most striking of these was the similarities between defining logical frames was similar to defining objects via a class. However, these frames don’t necessarily have members, but you still end up with multiple courses either way you program it. The primary difference between programming in a logical language versus an object oriented language was the actual usage of objects within the object oriented language. There was no “one size fits all” being applied as each object was unique to itself and did not natively share the same properties of other objects defined under the same class. However, in programming in a logical language, one frame was meant to be used to define different objects the same way giving them no individuality.

It’s impossible for me to say objectively whether it’s more efficient to program under a logical or object oriented paradigm. For this particular problem I felt rather indifferent as both paradigms could be used to clearly define and cleanly solve the problem being asked. I may have been more familiar with object oriented programming, but I also recognized the power which can be harnessed from a logical paradigm. Ultimately, as a programmer, I’ll use whichever language and paradigm I believe can get the job done in a way which is most efficient and can most easily be adapted by future programmers.