Nicholas Poon

Hardekopf

CS 162 Winter 2018

30 January 2018

Assignment 1 Writeup

For the first assignment we are asked to create a program using languages that we are unfamiliar with. As a student with a background in Java through Android application development, I have decided to proceed with Javascript for this particular programming assignment. My interest in attempting this assignment with JS was also bolstered by Professor Hardekopf's statement in class: "Javascript has NOTHING to do with Java" and further explained that it was simply a marketing gimmick by its creators for a false sense of credibility. For the reasons above, I have decided to proceed this assignment with JavaScript and draw its main functioning comparisons with Java.

We were asked to choose from a list of object-oriented languages. For that reason, I started this program with creating two simple objects. The first function - **createNewPerson(name)** - serves to create a Person object with a **greeting** method that outputs a browser alert. At the first glance, this function looks very much like a Java class - we have created a new empty object and promptly returned it, the greeting method could then be called from outside the function. The second function **Person(name)**, which provides identical functionalities, shows the use of Javascript's constructor functions. While it does not explicitly create or return objects, it serves to define properties and methods of the object **Person**.

Note that for every time a new object is created through the **Person(name)** function, it contains not only the person's name defined within the function, but also the **greeting** method for each individual object. This convention promotes a wasteful allocation of memory and performance; as Javascript is an interpreted language that's directly interpreted by client-side browsers, any impact can cause significant implications on performance. To avoid this, we can define functions on the object's prototype instead. The function **People(firstname, lastname, age, gender, interests)** serves as an example of defining functions on its object prototype.

Javascript has a core distinction that separates itself from other OOP languages. It is arguably the most popular language that supports prototype-based inheritance in place of class-based inheritance that we've seen from other well-known OOP languages such as Java and C++. The function **Teacher(firstname, lastname, age, gender, interests, subject)** is an example of prototype-based inheritance. In addition, the object Teacher also has a **greeting** function that overrides the previous **greeting** function from the object People. We can see that through two separate calls to **teacher1.greeting()** - the first call is made before the function has been overridden, while the second is called afterwards.

To further show prototypical inheritance in action, I have created multiple shape objects to demonstrate function inheritance. The inheritance chain goes as such: Shape -> Quadrilateral -> Rectangle -> Square. Each of the objects listed above has one specific function that is unique to them: Shape with **move()**, Quadrilateral with **perimeter()**, Rectangle with **Area()**. After creating a Square object, I promptly retrieved the inherited parameters (of the square object) from each of the square, rectangle, and quadrilateral objects. The parameters are then properly passed down. Functions, also, are properly passed down through prototypical inheritance. Each of the **move(), perimeter(), and area()** are called through the created square object. Even though none of the functions are defined within square's prototype, they were defined through the inheritance chain and are also properly shown in the browser popups.

I've used this assignment to demonstrate the prototype-based inheritance within Javascript, which is arguably the biggest distinction between itself and many other OOP languages that features class-based inheritance. Compared to a language like Java (with the classical parent-child inheritance relationship), implementing inheritance within Javascript is much more trivial. Multiple inheritances are also much easier to implement with just a few lines of extra code - I was able to simply just copy a subset of properties from one prototype into another prototype as evident from the shape objects. It is also very easy to change behaviors at runtime by simply changing properties within prototype chains. Though prototypical inheritance in Javascript has many advantages, there are a few drawbacks. For one, encapsulation is limited with prototypical inheritance; I cannot hide specific parameters within objects. Type safety is also difficult to enforce as type-checking is verified at runtime instead of compile time in another language like Java. This can lead to mistakes being overlooked into late in development. After this assignment, I have finally come to an understanding of why Professor Hardekopf states that Java and Javascript are entirely different languages. In fact, they share barely any similarity. As an aspiring front-end stack developer, I find that this assignment gives me a good entry point to start learning Javascript and eventually move onto libraries such as React.js.