CS224 – Lab 3

Purpose: Polymorphism and inheritance in Java requires the methods to be dynamically dispatched during runtime. Java always performs dynamic dispatch (also called late binding, or dynamic binding, etc) when invoking a method on an object. It looks up the runtime type of the actual object and then, if the method has been overridden one or more times in the inheritance hierarchy, it invokes the most specific method definition.

In this lab we will model this process of dynamic dispatch by implementing a recursive search for the desired method through the inheritance hierarchy.

Knowledge: This lab will help you become familiar with the following content knowledge:

- The mechanics of dynamic dispatch

Task: Follow the steps in this lab carefully to complete the assignments.

Assignment 1:
Now that you understand the `getClass` method we can write a recursive search to find the appropriate method for dynamic dispatch. Use the following steps to implement the `callMethodFromClassElseFromSuper` method in the `DynamicDispatch` class:

1. Since we′ll be using recursion, we need to have a base case. We′ll be recursing up the inheritance hierarchy with `Class.getSuperClass()`. If you do this iteratively we eventually end up with the class `Object`, which is the parent of all classes in Java. If you call `getSuperClass()` on `Object`, it returns `null`.

   In your method, first check to see if the Class argument is `null`. If it is `null` we know our search failed to find any method to invoke and you should create a new `NoSuchMethodException`, giving `message1` to the constructor and throw this exception.

2. Otherwise, use the static method `log` with `message2` indicating that you are now trying to find the method on the current class. Get the declared methods of the class (using `getDeclaredMethods()`) and loop over them using a for-each loop.

3. For each `Method` object, check if its name (returned by `getName()`) is the name of the method you are looking for. If it is, further check that its parameter types (returned by `getParameterTypes()`) are compatible with the arguments you are given. Do this with the provided static method `parameterTypesMatchArguments`.

4. If the name matches and the argument types are compatible, we′ve found our method! Use the log method on `message3` to state you found a suitable method. Try to return the result of invoking the method on the Object o with the provided arguments. Do this by returning the result of the following with `m` being the current method.

```
m.invoke(Object obj, Object?? args)
```

You will need to do this inside a `try` and you need to catch any Exceptions and throw a new `RuntimeException`, giving the constructor the exception you caught.

5. If you looped through all the declared methods on the given class, and none of them matched the desired method, then it′s time to continue the search in the parent class, to see if they have a definition for it. Return the result of recursively calling `callMethodFromClassElseFromSuper` with the super class of the class argument, and the same parameters otherwise.

Criteria for Success: When you uncomment `dynamicDispatchExercise` in the TestCode you should see the correct results for executing methods on the string `s`

Assignment 2:
Add additional tests in the TestCode for dispatching the method `toString` on the object `ac` (defined as in the previous code), and for dispatching the `sum` method on the objects `bb` and `cc`.

Criteria for Success: You get the same results as in the previous TestCode.

Submit your entire DynamicDispatch directory as a single zip archive in Canvas for grading.