William Romine

00103649

Dr. Lewis CS414

Assignment 02

Question 1:

The need for a C++-style switch construct is greatly alleviated in a purely object-oriented language by the combination of polymorphic message passing, dynamic typing and language simplicity. This is because objects in such a language can respond to messages - method calls depending on their class, without explicit conditional statements. This is polymorphic message passing, whereby objects of different classes can respond to the same message in their own way. There is no longer any need for a switch construct to handle different cases.

Consider "ColoredShapes Classes:" where their area can be computed. In a language providing for polymorphic message sending, you will be able to send an area message to any representative object of a shape, and it will be the actual class of the object that will determine which implementation is going to be executed. You wouldn't need a switch statement to handle each shape class. Thanks to dynamic typing, objects are treated as instances of their parent class, which simplifies the code. Therefore, the code can work with objects of different classes. This, combined with language simplicity that encourages a more declarative programming style, makes it easier to write concise and expressive code which focuses on the problem domain rather than implementation details.

In other words, polymorphic message passing combined with dynamic typing and simplicity of the language in a purely object-oriented language can express different cases more elegantly and expressively. The C++-style switch construct is superfluous. The natural behavior of objects and their classes can be exploited to write code that is more concise, flexible and maintainable, while being better suited to the problem domain.

Question 2:

Source: https://www.blackbox.ai/

https://tonyg.github.io/squeak-actors/testing.html

Implementation of ifAbsent:

```
!Object methodsFor: 'caseOf' stamp: ‘William Romine 2024-09-05 11:24'!

caseOf: aDictionary ifAbsent: aBlock

  | keyValue |

  keyValue := aDictionary keyAtValue: self ifAbsent: [ ^ aBlock value ].

  ^ (aDictionary at: keyValue) value

!!
```

Test Cases:

```
!CaseOfTest methodsFor: 'testing' stamp: ‘William Romine 2024-09-05 11:24'!

testSimpleCase

  | value |

  value := 1.

  self assert: (value caseOf: {

    [1] -> [ 'Value is 1' ].

    [2] -> [ 'Value is 2' ].

    [3] -> [ 'Value is 3' ].

    [4] -> [ 'Value is 4' ].

  } ifAbsent: [ 'Unknown value' ]) equals: 'Value is 1'


testMultipleCases

  | value |

  value := 3.

  self assert: (value caseOf: {

    [1] -> [ 'Value is 1' ].

    [2] -> [ 'Value is 2' ].
```

```
        [3] -> [ 'Value is 3' ].

        [4] -> [ 'Value is 4' ].

    } ifAbsent: [ 'Unknown value' ]) equals: 'Value is 3'


testNoMatch
```