Alesia Razumova
CS 680 HW19: Pseudo Code for Decorator Design Pattern (Note 147 Slide 43)

1. Show and explain how your code looks like if the Decorator design pattern is not used.

In case the current structure is not used, the assertThat() method of org.hamcrest.CoreMatches may become overwhelmingly complicated.

Declare string actual as “A”
Declare sting expected as “B”
asserThat if actual and expected instance of BigDecimal and actual is equal to expected, return true
else if actual and expected instance of BigDecimal and actual is not equal to expected, return false
else if actual instance of BigDecimal and expected is not instance of BigDecimal, return false
else if expected instance of BigDecimal and actual is not instance of BigDecimal, return false
else if actual and expected instance of String and actual is equal to expected, return true
else if actual and expected instance of String and actual is not equal to expected, return false
else if actual instance of String and expected is not instance of String, return false
else if expected instance of String and actual is not instance of String, return false
else if …

2. Explain how the Decorator design pattern eliminates conditional statements. Explain why Java API designers decided to use Decorator in java.io.

The Decorator design pattern can eliminate conditional statements because it allows the user to choose from a variety of options without having to iterate a sequence of options each time. This allows the code to be cleaner and more efficient as an intricate set of conditional statements can quickly become hard to maintain. As a result, Java API designers chose this pattern because it aids future development in the case a piece is updated/ refactored (this can be done since the code is decoupled) or something needs to be deprecated. This pattern preserves the original intention for the code without having to rewrite it entirely in case of a change, which is something that may happen with nested conditional statements.