Task 1

Below is a table showing the result of Equivalence Class creation based on Boundary Value Analysis. This table will include the main combination possibilities of clothes, accessories, and bear counts that affect how discounts are applied to purchases. Not included in the table are a few extra test cases that take into consideration items with different prices as well as a savings cost without any bears to increase the savings cost. The latter test was included in the test suite to help eliminate trivial cases while the former was to help provide more combinatorial robustness. The table is sectioned off into 3 columns indicating the number of bears associated with each test (1-3). This range helps cover the boundaries of the discount range as well as what works within the range. Each row specifies a test case with a certain number of clothes and accessories for each bear. Within each intersection of columns and rows is the specific test used to test the test case. Because there is a modulo operation of sorts in terms of how discounts are applied, several combinations of clothes and accessories within, on the borders, and outside the ranges of 0 – 3 for clothes and 0 – 10 for accessories are tested to see whether discounts are applied in various combinations.

	1 Bear	2 Bears	3 Bears
0 clothes 0 accessories	oneBearNoSavings	twoBearsExpectNoSavings	threeBearsSaveOnCheapest
2 clothes 0 accessories	oneBear2clothingsNoSavings	twoBears2clothingsExpectNoSavings	threeBearsTwoClothesBearSavingsExpected
3 clothes 0 accessories	oneBear3clothingsDifferentPricesExpectSaving	twoBears3clothingsExpectClothesSavingsOnly	threeBears3ClothesExpectClothingAndBearSavings
5 clothes 0 accessories	oneBear5clothingsDifferentPricesExpectSaving	twoBears5clothingsExpectClothesSavingsOnly	threeBears5ClothesExpectClothingAndBearSavings
6 clothes 0 accessories	oneBear6clothingsDifferentPricesExpectMoreSaving	twoBears6clothingsExpectMoreClothesSavingsOnly	threeBears6ClothesExpectMoreClothingAndSameBearSavings
7 clothes 0 accessories	oneBear7clothingsDifferentPricesExpectMoreSaving	twoBears7clothingsExpectMoreClothesSavingsOnly	threeBears7ClothesExpectMoreClothingAndSameBearSavings
0 clothes 1 accessory	oneBear1noiseAccessoryNoSavings	twoBears1accessoryExpectNoSavings	threeBears1accessoryExpectBearSavings
0 clothes 9 accessories	oneBear9noiseAccessoriesNoSavings	twoBears9accessoriesExpectNoSavings	threeBears9accessoriesExpectBearSavings
0 clothes 10 accessories	oneBear10noiseAccessoriesSamePricesExpectSaving	twoBears10accessoriesExpectAccessoriesSavingsOnly	threeBears10accessoriesExpect10PercentAndBearSavings
0 clothes 11 accessories	oneBear11noiseAccessoriesSamePricesExpectSaving	twoBears11accessoriesExpectAccessoriesSavingsOnly	threeBears11accessoriesExpect10PercentAndBearSavings
0 clothes 20 accessories	oneBear20noiseAccessoriesSamePricesExpectSaving	twoBears20accessoriesExpectAccessoriesSavingsOnly	threeBears20accessoriesExpect10PercentAndBearSavings
0 clothes 21 accessories	oneBear21noiseAccessoriesSamePricesExpectSaving	twoBears21accessoriesExpectAccessoriesSavingsOnly	threeBears21accessoriesExpect10PercentAndBearSavings
10 clothes 0 accessories	oneBear10clothesExpectNo10PercentSaving	twoBears10clothesExpectNo10PercentSavings	threeBears10clothesExpectNo10PercentSaving
12 clothes 0 accessories	oneBear12clothesExpectNo10PercentSaving	twoBears12clothesExpectNo10PercentSavings	threeBears12clothesExpectNo10PercentSaving
14 clothes 0 accessories	oneBear14clothesExpectNo10PercentSaving	twoBears14clothesExpect10PercentSavingsAndClothesSavings	threeBears14clothesExpect10PercentSaving
3 clothes 8 accessories	oneBear3clothes8accessoriesExpectClothesAndAccessoriesSavings	twoBears3clothes8accessoriesExpectClothesAndAccessoriesSavings	threeBears3clothes8accessoriesExpectClothesAndAccessoriesSavings

Based on the above tests generated through Boundary Value Analysis, the following can be said about the 5 implementations of calculateSavings():

1. Only the fifth implementation (index 4) adheres to the specifications of calculateSavings(). Every other implementation encountered some error during testing.
2. Here is a list of errors from each implementation and what could be going wrong.

a. Implementation 0

    i. During instances of applying a buy 2 get 1 free discount (whether it be for clothes or bears), implementation 0 encountered difficulties in handling items with different prices. This indicates that savings may have been applied, but not based on the item with the lowest cost. These indicators are tied to several test cases, including threeBearsSaveOnCheapest and oneBear3clothingsDifferentPricesExpectSaving. These two tests (especially the first) are specific enough that, in the context of other existing test cases, the described error is the most likely culprit.

    ii. Another possible cause for error lies in the calculation of the 10% discount. However, no errors occur when calculating the discounts of purchases involving fewer than 3 bears. This means that the calculation incorrectly factors in the removal of a bear. This is indicated by the failure of tests like threeBears14clothesExpect10PercentSavings and the success of tests like threeBears12clothesExpectNo10PercentSavings. Because the latter test does not remove a bear price as a discount while the former does, this acts as a hint that the calculation of 10% for each bear includes the price of the discounted bear when it should not.

b. Implementation 1

    i. Errors for implementation 1 indicate that whenever there is any clothing item or accessory on a bear, the savings will not be calculated correctly. Bear discounts work correctly as evidenced by threeBearsSaveOnCheapest, which has three bears that are completely undecorated. There is some issue with the way this

implementation tracks clothes and accessories, which would affect the application of discounts when those conditions are met.

c. Implementation 2
    i. oneBear2clothingsNoSavings indicates that this implementation calculates the indicates a fundamental issue with the calculation of clothes discounts. While tests with naked bears like threeBearsSaveOnCheapest pass, others like threeBearsTwoClothesBearSavingsExpected do not, likely due to an incorrect inclusion of a clothes discount. However, the failure of test oneBear2clothingsNoSavings indicates that any inclusion of clothes, even if they do not provide a discount, throw off the discount calculation.
    ii. Another discount that appears to be applied incorrectly is the 10% discount. Tests like threeBears9accessoriesExpectBearSavings, which include naked bears with accessories, pass since they do not pass the condition to apply the 10% discount. Additionally, tests like twoBears10accessoriesExpect10PercentSavings also pass despite meeting the conditions. However, they fail when factoring 3 bears for the buy 2 get 1 free discount. This means that, while the 10% discount can be applied in certain circumstances, the implementation fails to factor in the bear discount.

d. Implementation 3
    i. oneBearNoSavings is the test that demonstrates that the calculateSavings method for this implementation is fundamentally broken. Even when there are no possible reasons for savings to be calculated based on the description, some calculation occurs. There can be a number of reasons for this to happen, including the possibility that all savings are calculated despite not meeting the

required conditions. This requires further investigation with testing tailored to this specific implementation.

Task 2