01286222 Lab 7 Name... id ....

Objective(s):

a. To understand the recursive algorithm via min-coin-change problem

b. To be able to improve a recursive algorithm with memorization technique.

```
public static void main(String[] args) {
 //ex1();
 int amount = 0;
 for (amount = 4; amount < 12; amount++) {
     println("amount " + amount + " uses " + minNumCoin(amount));
     // int [] minCoinResidual = new int[amount+1];
     // println("amount " + amount + " uses "
                          + minNumCoin(amount,minCoinResidual));
 }
}
static void ex1() {
 for (Coins c : Coins.values())
     System.out.println("Coin " + c + "s are ready ");
}

static int minNumCoin(int amount) {
 // System.out.println("call for " + amount);
 if (amount == 0) return 0; //base case

 int coinsNeeded = Integer.MAX_VALUE;

 int numCoin = 0;
 for (Coins c : Coins.values()) {
     if (c.value() <= amount) {
         numCoin = 1 + minNumCoin(amount - c.value());
         /* your code */
     }
 }
 /* equiv to below code */
 // int minP, minN, minD;
 // minP = minN = minD = 0;
 // if (amount >= Coins.PENNY.value()) {
 //     minP = 1 + minNumCoin(amount - Coins.PENNY.value());
 //     if (minP < coinsNeeded)
 //         coinsNeeded = minP;
 // }
 // if (amount >= Coins.NICKEL.value()) {
 //     minN = 1 + minNumCoin(amount - Coins.NICKEL.value());
 //     if (minN < coinsNeeded)
 //         coinsNeeded = minN;
 // }
 // if (amount >= Coins.DIME.value()) {
 //     minD = 1 + minNumCoin(amount - Coins.DIME.value());
 //     if (minD < coinsNeeded)
 //         coinsNeeded = minD;
 // }
 return coinsNeeded;
}
```

Task 1:

Given Coins.java (contains Penny, Nickel, and Dime coin type with its corresponded value (by calling .value() method). ex1() code demonstrates that a java enumeration can be traversed (each coin type in the enum). Fill in the code so that the `int minNumCoin(int amount)` produces the minimum coins required for the requested amount.

Instructions: Capture the for loop code (**in light theme**) containing your answer.

Task : 2

Memoization is a buffer (int [] residual in this case) to store previously computed minNumCoin. Later call would no longer re-compute the value of the given amount but retrieve from the buffer instead. The mechanism greatly reduces the number of recursive calls.

Instructions: Capture the for loop code (**in light theme**) containing your answer.

```
static int minNumCoin(int amount, int [] residual) {
  if (amount == 0) return 0; //base case

  int coinsNeeded = Integer.MAX_VALUE;

  int numCoin = 0;
  for (Coins c : Coins.values()) {
      if (c.value() <= amount) {
          if (residual[amount -  c.value()] > 0)
              numCoin = 1 + residual[amount - c.value()];
          else
              numCoin = 1 + minNumCoin(amount - c.value());
          if (numCoin < coinsNeeded)
              coinsNeeded = numCoin;
          /* your code */
      }
  }
  return coinsNeeded;
}
```

Task : 3

```
  static void fasterBy() {
    int amount = 59;
    long time = System.currentTimeMillis();
    print("amount = " + amount + " uses " + minNumCoin(amount));
    println(" elapse time = " + (int)(System.currentTimeMillis() - time));

    time = System.currentTimeMillis();
    int [] residual = new int [amount + 1];
    print("amount = " + amount + " uses " +
minNumCoin(amount,residual));
    println(" elapse time = " + (int)(System.currentTimeMillis() - time));
  }
```

Capture the output of fasterBy() (**in light theme**) (you may increase the amount value).

Submission: This pdf

Due date: TBA