```java
 * JUnit test class.  Use these tests as models for your own.
import org.junit.*;
import org.junit.rules.Timeout;
import static org.junit.Assert.*;

import proj1.Coinbank;

public class CoinbankTest {

    @Rule // a test will fail if it takes longer than 1/10 of a second to run
    public Timeout timeout = Timeout.millis(100);

    /**
     * Sets up a bank with the given coins
     * @param pennies number of pennies you want
     * @param nickels number of nickels you want
     * @param dimes number of dimes you want
     * @param quarters number of quarters you want
     * @return the Coinbank filled with the requested coins of each type
     */
    private Coinbank makeBank(int pennies, int nickels, int dimes, int quarters) {
        Coinbank c = new Coinbank();
        int[] money = new int[]{pennies, nickels, dimes, quarters};
        int[] denom = new int[]{1,5,10,25};
        for (int index=0; index<money.length; index++) {
            int numCoins = money[index];
            for (int coin=0; coin<numCoins; coin++) {
                c.insert(denom[index]);
            }
        }
        return c;
    }

    @Test // bank should be empty upon construction
    public void testConstruct() {
        Coinbank emptyDefault = new Coinbank();
        assertEquals(0, emptyDefault.get(1));
        assertEquals(0, emptyDefault.get(5));
        assertEquals(0, emptyDefault.get(10));
        assertEquals(0, emptyDefault.get(25));
    }


    @Test // inserting penny should return true & one penny should be in bank
    public void testInsertPenny_return()
    {
        Coinbank c = new Coinbank();
        assertTrue(c.insert(1));
        assertEquals(1,c.get(1));
    }

    @Test // inserting nickel should return true & one nickel should be in bank
    public void testInsertNickel_return()
    {
        Coinbank c = new Coinbank();
        assertTrue(c.insert(5));
        assertEquals(1,c.get(5));
    }

    @Test // inserting dime should return true & one dime should be in bank
    public void testInsertDime_return()
    {
        Coinbank c = new Coinbank();
        assertTrue(c.insert(10));
        assertEquals(1,c.get(10));
    }
    @Test // inserting quarter should return true & one quarter should be in bank

    public void testInsertQuarter_return()
    {
        Coinbank c = new Coinbank();
        assertTrue(c.insert(25));
        assertEquals(1,c.get(25));
    }

    @Test // inserting invalid coin should return false & no coins should be in bank
    public void testInsertInvalid_return()
    {
        Coinbank c = new Coinbank();
        assertFalse(c.insert(3));
        assertEquals(-1,c.get(3));
    }

    @Test // getter should return correct values
    public void testGet()
    {
        Coinbank c = makeBank(0,2,15,1);
        assertEquals(0,c.get(1));
        assertEquals(2,c.get(5));
        assertEquals(15,c.get(10));
        assertEquals(1,c.get(25));
        assertEquals(-1, c.get(3));
    }

    @Test // getter should not alter the bank
    public void testGet_contents()
    {
        Coinbank c = makeBank(0,2,15,1);
        c.get(1);
        c.get(5);
        c.get(10);
        c.get(25);
        c.get(3);
        String expected = "The bank currently holds $1.85 consisting of \n0 pennies\n2
nickels\n15 dimes\n1 quarters\n";
        assertEquals(expected,c.toString());
    }

    @Test //test of remove removing to many coins
    public void testRemove_toMany() {
        Coinbank c = makeBank(2,1,0,3);
        assertEquals(1, c.remove(5, 3));
        String expected = "The bank currently holds $0.77 consisting of \n2 pennies\n0
nickels\n0 dimes\n3 quarters\n";
        assertEquals(expected,c.toString());
    }

    @Test //test of remove removing less coins
    public void testRemove_less() {
        Coinbank c = makeBank(1,2,3,3);
        assertEquals(2, c.remove(10, 2));
        String expected = "The bank currently holds $0.96 consisting of \n1 pennies\n2 nickels\n1 dimes\n3 quarters\n";
        assertEquals(expected,c.toString());
    }

    @Test //test of remove removing invalid
    public void testRemove_invalid() {
        Coinbank c = makeBank(1,2,3,3);
        assertEquals(0, c.remove(6, 2));
        String expected = "The bank currently holds $1.16 consisting of \n1 pennies\n2 nickels\n3 dimes\n3 quarters\n";
        assertEquals(expected,c.toString());
    }

    @Test // test of remove
    public void testRemove_justEnough()
    {
        Coinbank c = makeBank(4,1,3,5);
        assertEquals(5,c.remove(25,5));
        String expected = "The bank currently holds $0.39 consisting of \n4 pennies\n1 nickels\n3 dimes\n0 quarters\n";
        assertEquals(expected,c.toString());
    }

    @Test // remove should not do anything if a 3-cent coin is requested
    public void testRemove_invalidCoin()
    {
        Coinbank c = makeBank(4,1,3,5);
        assertEquals(0,c.remove(3,1));
    }
}
```

```java
package proj1;  // Don't change the package name.  Gradescope expects this.

/**
 * This is the Coin bank class it holds different coins and allows a person to insert or remove a coin
 * and tells the amount of money and coins in the bank
 * @author Matthew Caulfield
 * @version 9/20/17
 *
 * I affirm that I have carried out the attached academic endeavors with full academic honesty, in
 * accordance with the Union College Honor Code and the course syllabus.
 */
public class Coinbank {

    // Denominations
    public static final int PENNY_VALUE = 1;
    public static final int NICKEL_VALUE = 5;
    public static final int DIME_VALUE = 10;
    public static final int QUARTER_VALUE = 25;

    // give meaningful names to holder array indices
    private final int PENNY = 0;
    private final int NICKEL = 1;
    private final int DIME = 2;
    private final int QUARTER = 3;

    // how many types of coins does the bank hold?
    private final int COINTYPES = 4;

    private int[] holder;

    /**
     * Default constructor
     */
    public Coinbank() {
        holder = new int[COINTYPES];
        for(int i = 0; i < COINTYPES; i++) {
            holder[i] = 0;
        }
    }

    /**
     * getter
     * @param coinType denomination of coin to get. Valid denominations are
     * 1,5,10,25
     * @return number of coins that bank is holding of that type, or -1
     * if denomination not valid
     */
    public int get(int coinType){
        if(isBankable(coinType)) {
            return holder[getCoinIndex(coinType)];
        }
        else {
            return -1;
        }
    }

    /**
```

```
     * setter
     * @param coinType denomination of coin to set
     * @param numCoins number of coins
     */
    private void set(int coinType, int numCoins) {
        if(isBankable(coinType)) {
            holder[getCoinIndex(coinType)] = numCoins;
        }
    }

    /**
     * takes the value of a coin and returns its index in the holder array
     * the coin value must be a valid value 1, 5, 10, 25
     * @param coinType
     * @return Constant that is the index of the coin in the holder array
     */
    private int getCoinIndex(int coinType) {
        if(coinType == PENNY_VALUE) {
            return PENNY;
        }
        else if (coinType == NICKEL_VALUE) {
            return NICKEL;
        }
        else if (coinType == DIME_VALUE) {
            return DIME;
        }
        else{
            return QUARTER;
        }
    }

    /**
     * Return true if given coin can be held by this bank.  Else false.
     * @param coin penny, nickel, dime, or quarter is bankable. All others are not.
     * @return true if bank can hold this coin, else false
     */
    private boolean isBankable(int coin){
        switch (coin) {
        case PENNY_VALUE: case NICKEL_VALUE:
        case DIME_VALUE: case QUARTER_VALUE:
            return true;
        default:
            return false;
        }
    }

    /**
     * insert valid coin into bank.  Returns true if deposit
     * successful (i.e. coin was penny, nickel, dime, or quarter).
     * Returns false if coin not recognized
     *
     * @param coinType either 1, 5, 10, or 25 to be valid
     * @return true if deposit successful, else false
     */
    public boolean insert(int coinType){
        if (!isBankable(coinType)) {
            return false;
        }
        else {
```

```java
            set(coinType, get(coinType)+1);
            return true;
        }
    }

    /**
     * returns the requested number of the requested coin type, if possible.
     * Does nothing if the coin type is invalid.  If bank holds
     * fewer coins than is requested, then all of the coins of that
     * type will be returned.
     * @param coinType either 1, 5, 10, or 25 to be valid
     * @param requestedCoins number of coins to be removed
     * @return number of coins that are actually removed
     */
    public int remove(int coinType, int requestedCoins) {
        int coinsHave = get(coinType);
        int coinsLeft = numLeft(requestedCoins, coinsHave);
        if(requestedCoins >= 0 && isBankable(coinType)) {
            set(coinType, coinsLeft);
            if(coinsLeft > 0) {
                return requestedCoins;
            }
            else{
                return coinsHave;
            }
        }
        else {
            return 0;
        }
    }


    /**
     * returns number of coins remaining after removing the
     * requested amount.  Returns zero if requested amount > what we have
     * @param numWant number of coins to be removed
     * @param numHave number of coins you have
     * @return number of coins left after removal
     */
    private int numLeft(int numWant, int numHave){
        return Math.max(0, numHave-numWant);
    }

    /**
     * Returns bank as a printable string
     */
    public String toString() {
        double total = (get(PENNY_VALUE) * PENNY_VALUE +
                get(NICKEL_VALUE) * NICKEL_VALUE +
                get(DIME_VALUE) * DIME_VALUE +
                get(QUARTER_VALUE) * QUARTER_VALUE) / 100.0;

        String toReturn = "The bank currently holds $" + total + " consisting of \n";
        toReturn+=get(PENNY_VALUE) + " pennies\n";
        toReturn+=get(NICKEL_VALUE) + " nickels\n";
        toReturn+=get(DIME_VALUE) + " dimes\n";
        toReturn+=get(QUARTER_VALUE) + " quarters\n";
        return toReturn;
    }
```

```
}
```