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

/**
 * Models a plastic roll-a-coin bank from the 1980s. Bank holds pennies, nickles, dimes and quarters.
 * Refactored version of original/Coinbank which is more readable, reusable and robust.
 *
 * @author Neil Datearo
 * @version 9/22/24
 * @Note 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;

    // how many coins should be in the starting bank?
    private final int INITIALCOINCOUNT = 0;
    private int[] holder;

    /**
```

```
     * Default constructor
     */
    public Coinbank() {
        holder = new int[COINTYPES];
        holder[PENNY] = INITIALCOINCOUNT;
        holder[NICKEL] = INITIALCOINCOUNT;
        holder[DIME] = INITIALCOINCOUNT;
        holder[QUARTER] = INITIALCOINCOUNT;

    }

    /**
     * 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) {
        int coinTypeIndex = getCoinIndex(coinType);

        if (isValidCoinIndex(coinTypeIndex)) { return holder[coinTypeIndex]; }
        else{ return coinTypeIndex; }
    }

    /**
     * setter
     * @param coinType denomination of coin to set
     * @param numCoins number of coins
     */
    private void set(int coinType, int numCoins) {
        int coinTypeIndex = getCoinIndex(coinType);

        if (isValidCoinIndex(coinTypeIndex)) { holder[coinTypeIndex] = numCoins; }

    }

    /**
     * Private method to get coin index in holder.
```

```
Reusable logic for getter and setters.
     * @param coinType
     * @return coinType index. If -1, it is an invalid denomination.
     */
    private int getCoinIndex(int coinType) {
        if (isBankable(coinType)) {
            switch (coinType) {
                case PENNY_VALUE:
                    return PENNY;
                case NICKEL_VALUE:
                    return NICKEL;
                case DIME_VALUE:
                    return DIME;
                case QUARTER_VALUE:
                    return QUARTER;
                default:
                    return -1; //in event that denomination is invalid
            }
        }
        else{
                return -1;
            }

    }

    /**
     * Private helper method to check if a coin index is valid. Makes code more readable
     * @param coinIndex Index to check coin in holder.
     * @return True if valid index and false if not.
     */
    private boolean isValidCoinIndex(int coinIndex) {
        return coinIndex != -1;
    }

    /**
```

```
     * 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 {
            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 coinTypeIndex = getCoinIndex(coinType);
        if (isValidCoinIndex(coinTypeIndex) && requestedCoins > 0) {
            int coinsWeHave = get(coinType);
            int coinsLeft = numLeft(requestedCoins, coinsWeHave);
            int coinsRemoved = coinsWeHave - coinsLeft;
            set(coinType, coinsLeft); //update coin holder
            return coinsRemoved;
        }

        return 0; //If invalid coin type, no coins will be removed.

    }


    /**
     * 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) / 100.00;

        String formattedTotal = String.format("%.2f", total);


        String toReturn = "The bank currently holds $" + formattedTotal + " 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;
    }
}

```

```java
package common;
import java.util.Random;

import original.Coinbank;

public class Client {

    public static void main(String[] args) {
        Coinbank bank = new Coinbank();

        // insert 10 random coins into the bank and print it
        for (int i=0; i<10; i++) {
            bank.insert(getRandomCoin());
        }
        System.out.println(bank);

        // try to remove from 0-5 coins of a random type
        int toRemove = (int)(Math.random() * 6);
        int denom = getRandomCoin();
        int got = bank.remove(denom,toRemove);
        System.out.println("\nTried to remove " + toRemove + " " + denom + "-cent coins and got " + got + "\n");
        System.out.println(bank);
    }

    /**
     * Gets a random coin denomination
     * @return 1, 5, 10, or 25
     */
    public static int getRandomCoin() {
        Random r = new Random();
        int num = r.nextInt(4);
        switch(num) {
        case 0: return Coinbank.PENNY_VALUE;
        case 1: return Coinbank.NICKEL_VALUE;
        case 2: return Coinbank.DIME_VALUE;
        case 3: return Coinbank.QUARTER_VALUE;
        default: return 0;
```

```
            }
        }

}

```

```java
package common;
/**
 * 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 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 // 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));
    }

    @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);
        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
    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));
    }

    @Test //Testing valid cointype but an attempt to remove a negative number of coins is invalid.
    public void testRemove_negativeCoinWithdrawal()
    {
        Coinbank c = makeBank(4,1,3,5);
        assertEquals(0, c.remove(25, -4) );
    }

    @Test //get should return -1 for invalid coin types
    public void testGet_InvalidCoinType() {
        Coinbank c = makeBank(10,12,4,2);

        assertEquals(-1, c.get(2));  // Invalid coin type
    }

```

```
    @Test //insert should return false and not change bank contents for invalid coin types
    public void testInsert_InvalidCoinType() {
        Coinbank c = makeBank(10,12,4,2);

        assertFalse(c.insert(2)); // Invalid coin type
        String expected = "The bank currently holds $1.60 consisting of \n10 pennies\n12 nickels\n4 dimes\n2 quarters\n";
        assertEquals(expected,c.toString()); // Should not change anything in bank
    }
    @Test //insert should return true and update penny count by 1
    public void testInsert_Penny() {
        Coinbank c = makeBank(10,12,4,2);

        assertTrue(c.insert(1)); // valid coin type
        String expected = "The bank currently holds $1.61 consisting of \n11 pennies\n12 nickels\n4 dimes\n2 quarters\n";
        assertEquals(expected,c.toString()); // Should update the bank
    }
    @Test //insert should return true and update dime count by 1
    public void testInsert_Dime() {
        Coinbank c = makeBank(10,12,4,2);

        assertTrue(c.insert(10)); // valid coin type
        String expected = "The bank currently holds $1.70 consisting of \n10 pennies\n12 nickels\n5 dimes\n2 quarters\n";
        assertEquals(expected,c.toString()); // Should update the bank
    }
    @Test //insert should return true and update quarter count by 1
```

```java
    public void testInsert_Quarter() {
        Coinbank c = makeBank(10,12,4,2);

        assertTrue(c.insert(25)); // valid coin type
        String expected = "The bank currently holds $1.85 consisting of \n10 pennies\n12 nickels \n4 dimes\n3 quarters\n";
        assertEquals(expected,c.toString()); // Should update the bank
    }


}

```

```java
package original;

/**
 * Models a plastic roll-a-coin bank from the 1980s
 *
 * @author Chris Fernandes
 * @version 8/29/17
 *
 */
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;

    // number of coins the bank is holding
    private int pennies;
    private int nickels;
    private int dimes;
    private int quarters;

    /**
     * Default constructor
     */
    public Coinbank() {
        pennies = 0;
        nickels = 0;
        dimes = 0;
        quarters = 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){
```

```java
        if (isBankable(coinType)) {
            if (coinType == PENNY_VALUE) {
                return pennies;
            }
            else if (coinType == NICKEL_VALUE) {
                return nickels;
            }
            else if (coinType == DIME_VALUE) {
                return dimes;
            }
            else {  // coinType == QUARTER_VALUE
                return quarters;
            }
        }
        else {
            return -1;
        }
    }

    /**
     * setter
     * @param coinType denomination of coin to set
     * @param numCoins number of coins
     */
    private void set(int coinType, int numCoins) {
        if (numCoins >= 0 && isBankable(coinType
    )) {
            if (coinType == PENNY_VALUE) {
                pennies = numCoins;
            }
            else if (coinType == NICKEL_VALUE) {
                nickels = numCoins;
            }
            else if (coinType == DIME_VALUE) {
                dimes = numCoins;
            }
            else {  // coinType == QUARTER_VALUE
                quarters = numCoins;
            }
        }
    }
```

```java

    /**
     * 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 coin 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 {
            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 removedCoins = 0;
        if (isBankable(coinType) && requestedCoins >= 0) {

            int coinsWeHave;  // coins we have before removal
            int coinsLeft;    // coins we have after removal

            if (coinType == PENNY_VALUE) {
                coinsWeHave = pennies;
                coinsLeft = numLeft(requestedCoins, pennies);
                pennies = coinsLeft;
            }
            else if (coinType == NICKEL_VALUE) {
                coinsWeHave = nickels;
                coinsLeft = numLeft(requestedCoins, nickels);
                nickels = coinsLeft;
            }
            else if (coinType == DIME_VALUE) {
                coinsWeHave = dimes;
                coinsLeft = numLeft(requestedCoins
```

```
, dimes);
                dimes = coinsLeft;
            }
            else {  //coinType == QUARTER_VALUE
                coinsWeHave = quarters;
                coinsLeft = numLeft(requestedCoins, quarters);
                quarters = coinsLeft;
            }

            // how many coins actually got removed?
            removedCoins = coinsWeHave - coinsLeft;
        }
        return removedCoins;
    }


    /**
     * 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;
    }
}
```