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

/**
 * FILL THIS IN FOR EVERY PROJECT.  Include a class description, name, and
date (for version)
 * @author Emma Vu
 * @version 4/12/2020
 *
 */
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() {

            this.holder = new int[COINTYPES];

      }

      /**
       * Take the coinType and return its name in holder array.
       * This helper will do all the work of if-else statements that can
be used later in getter, setter and remove method.
       * @param coinType the denomination of coin. 1, 5, 10, 25 are valid
       * @return the name of the given coinType
       */

      private int coinTypeName(int coinType){
            int typeName;
            if(coinType == PENNY_VALUE){
                  typeName = PENNY;
            }
            else if(coinType == NICKEL_VALUE){
                typeName = NICKEL;
            }
            else if(coinType == DIME_VALUE){
                typeName = DIME;
            }
            // coinType == QUARTER_VALUE
            else{
                typeName = QUARTER;
            }
            return typeName;

        }

        /**
         * a helper method to get number of coins given the coinType
         * @param type the denomination of coin. 1, 5, 10, 25 are valid
         * @return the number of coins
         */

        private int getNumCoins(int type){
            return this.holder[coinTypeName(type)];
        }

        /**
         * 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 this.getNumCoins(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 (numCoins >= 0 && isBankable(coinType)) {

                this.setNumCoins(numCoins, coinType);
```

```
		}

	}


	/**
	 * helper method to set the number of coins given the coinType
	 * @param numToSet the amount of coins want to set
	 * @param type the denomination of coins. 1, 5, 10, 25 are valid
	 */
	private void setNumCoins(int numToSet, int type){
		this.holder[coinTypeName(type)] = numToSet;
	}

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

                removedCoins = this.calculateRemovedCoins(coinType, 
requestedCoins);

           }
           return removedCoins;

      }

      /**
       * calculate the coins been removed given the type and the amount want 
to be removed
       * @param type the denomination of coins. 1, 5, 10, 25 are valid
       * @param coinsRequest the amount want to be removed
       * @return the coins that are removed
       */

      public int calculateRemovedCoins(int type, int coinsRequest){
           int coinsBefore;
           int coinsAfter;
           coinsBefore = this.get(type);
           coinsAfter = this.numLeft(coinsRequest, coinsBefore);
           this.set(type, coinsAfter);
           return coinsBefore - coinsAfter;
      }


      /**
       * 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;
      }
}

----------------------------CoinBankTest------------------------

/**
 * 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 of each coinType
	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 content
	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 just enough coins for valid coin type. The remains 
for that coin type should be 0
	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 an invalid coin type is requested.
It returns 0 and bank contents don't change
      public void testRemove_invalidCoin()
      {
            Coinbank c = makeBank(4,1,3,0);
            assertEquals(0,c.remove(3,1));
            String expected = "The bank currently holds $0.39 consisting
of \n4 pennies\n1 nickels\n3 dimes\n0 quarters\n";
            assertEquals(expected, c.toString());
      }

      // New additional tests

      @Test
      // insert invalid coin type so the bank don't recognize. The contents
don't change
      public void testInsertInvalid(){
            Coinbank c = makeBank(4,1,3,0);
            assertFalse(c.insert(3));
            assertFalse(c.insert(0));
            assertFalse(c.insert(-4));
            assertEquals(-1,c.get(3));
            assertEquals(-1,c.get(0));
            assertEquals(-1,c.get(-4));
            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 more coins than bank have so bank remove all the coins of
the type asked for
      public void testRemoveMoreThanHave(){
            Coinbank c = makeBank(0,2,15,1);
            assertEquals(15,c.remove(10,20));
            String expected = "The bank currently holds $0.35 consisting
of \n0 pennies\n2 nickels\n0 dimes\n1 quarters\n";
            assertEquals(expected,c.toString());

      }

      @Test
      //remove zero coin in bank so the return will be 0. The contents don't
change
```

```
      public void testRemoveNoCoins(){
            Coinbank c = makeBank(4,1,3,2);
            assertEquals(0,c.remove(10,0));
            String expected = "The bank currently holds $0.89 consisting
of \n4 pennies\n1 nickels\n3 dimes\n2 quarters\n";
            assertEquals(expected,c.toString());


      }

      @Test
      //remove fewer coins than bank have so the bank remove that requested
amount and still have remaining in bank
      public void testRemoveFewerThanHave(){
            Coinbank c = makeBank(10,2,5,6);

            assertEquals(3,c.remove(1,3));
            String expected = "The bank currently holds $2.17 consisting
of \n7 pennies\n2 nickels\n5 dimes\n6 quarters\n";
            assertEquals(expected,c.toString());



      }

      @Test
      //insert multiple times of same coin type. The number of coins should
+1 after each insertion.
     //New contents are updated
      public void testInsertMultipleSameCoins(){
            Coinbank c = makeBank(3,1,9,2);
            c.insert(5);
            c.insert(5);
            c.insert(5);

            assertEquals(4,c.get(5));
            String expected = "The bank currently holds $1.63 consisting
of \n3 pennies\n4 nickels\n9 dimes\n2 quarters\n";
            assertEquals(expected,c.toString());
      }

      @Test
     //insert multiple times of different coin types. The number of coins
should +1 after each insertion
     //New contents are updated
     public void testInsertMultipleDifferentCoins(){
           Coinbank c = makeBank(1,4,0,5);
           c.insert(1);
           c.insert(1);

           c.insert(10);
           c.insert(10);
```

```
          c.insert(10);

          c.insert(25);

          assertEquals(3,c.get(1));
          assertEquals(4,c.get(5));
          assertEquals(3,c.get(10));
          assertEquals(6,c.get(25));

          String expected = "The bank currently holds $2.03 consisting of
\n3 pennies\n4 nickels\n3 dimes\n6 quarters\n";
          assertEquals(expected,c.toString());
    }

     @Test
     //get invalid coinType, then return -1
     public void testGetInvalidCoin(){
           Coinbank c = makeBank(1,2,3,4);
           assertEquals(-1,c.get(7));
           assertEquals(-1,c.get(0));
           assertEquals(-1,c.get(-13));
     }

     @Test
    //test toString of valid bank with existed coins
    public void testToStringValidBank(){
        Coinbank c = makeBank(3,2,0,1);
        String expected = "The bank currently holds $0.38 consisting of
\n3 pennies\n2 nickels\n0 dimes\n1 quarters\n";
        assertEquals(expected,c.toString());
    }

    @Test
    //test toString of a new empty bank with zero coin of every types
    public void testToStringNewBank(){
          Coinbank c = new Coinbank();
          String expected = "The bank currently holds $0.0 consisting of
\n0 pennies\n0 nickels\n0 dimes\n0 quarters\n";
          assertEquals(expected,c.toString());
    }

    @Test
    //test remove negative number of coins. The bank should return 0, the
contents don't change
    public void testRemoveNegativeNumCoins(){
        Coinbank c = makeBank(4,1,3,5);
        assertEquals(0,c.remove(25,-2));
        String expected = "The bank currently holds $1.64 consisting of
\n4 pennies\n1 nickels\n3 dimes\n5 quarters\n";
        assertEquals(expected,c.toString());
    }
```

```
}
```