Exercise 1

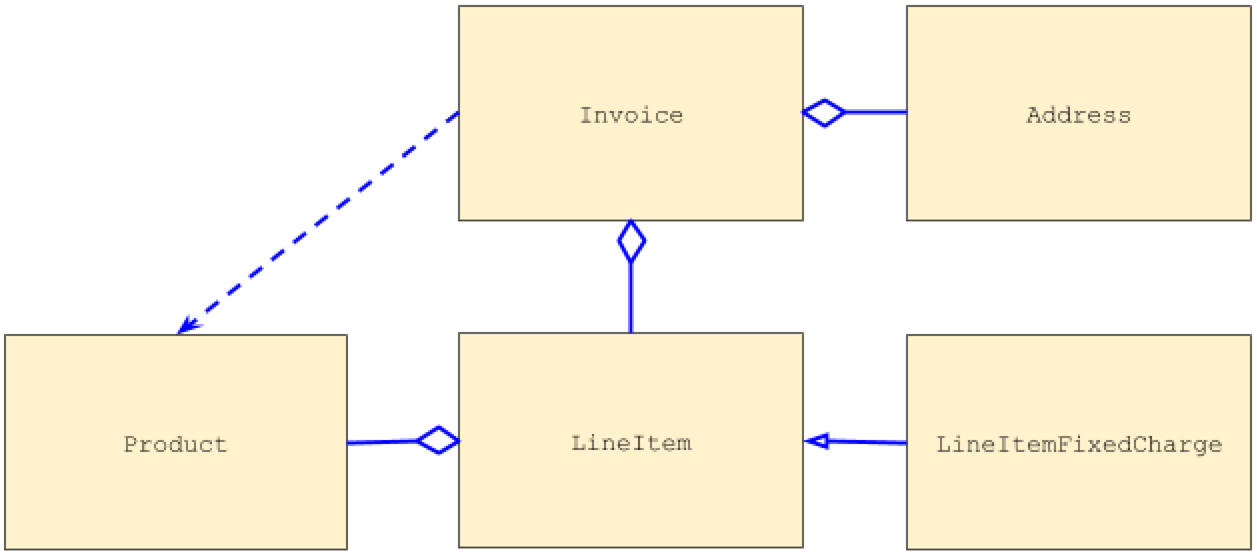
```
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 1 of Chapter 12.
	Describes a mailing address.
*/
public class Address
{
	private String name;
	private String street;
	private String city;
	private String state;
	private String zip;

	/**
		Constructs a mailing address
		@param aName the recipient name
		@param aStreet the street
		@param aCity the city
		@param aState the two-letter state code
		@param aZip the ZIP postal code
	*/
	public Address(String aName, String aStreet, String aCity, String aState, String aZip)
	{
		name = aName;
		street = aStreet;
		city = aCity;
		state = aState;
		zip = aZip;
	}

	/**
		Formats the address.
		@return the address as a string with three lines
	*/
	public String format()
	{
		return name + "\n" + street + "\n" + city + ", " + state + " " + zip;
	}
}

/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 1 of Chapter 12.
	Describes a product with a description and a price.
*/
public class Product
{
	private String description;
	private double price;

	/**
		Constructs a product from a description and a price.
		@param aDescription the product description
		@param aPrice the product price
	*/
	public Product(String aDescription, double aPrice)
	{
		description = aDescription;
		price = aPrice;
	}

	/**
		Gets the product description.
		@return the description
	*/
	public String getDescription()
	{
		return description;
	}

	/**
		Gets the product price
		@return the unit price
	*/
	public double getPrice()
	{
		return price;
	}
}

/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 1 of Chapter 12.
	Describes a quantity of an article to purchase.
*/
public class LineItem
{
```

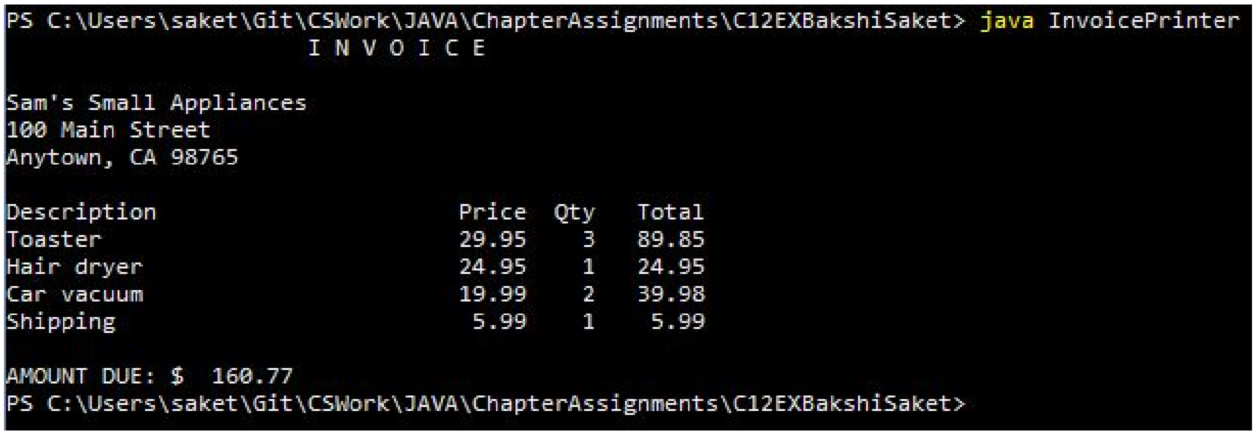
```
private int quantity;
private Product theProduct;

/**
	Constructs a blank LineItem
*/
public LineItem()
{
	theProduct = new Product("", 0);
	quantity = 0;
}

/**
	Constructs an item from the product and the quantity.
	@param aProduct the product
	@param aQuantity the item quantity
*/
public LineItem(Product aProduct, int aQuantity)
{
	theProduct = aProduct;
	quantity = aQuantity;
}

/**
	Sets the quantity of the product
*/
public void setQuantity(int aQuantity)
{
	quantity = aQuantity;
}

/**
	Sets the product name
*/
public void setProduct(Product aProduct)
{
	theProduct = aProduct;
}

/**
	Computes the total cost of this line item.
	@return the total price
*/
```

```
	public double getTotalPrice()
	{
		return theProduct.getPrice() * quantity;
	}

	/**
		Formats this item.
		@return a formated string of this line item
	*/
	public String format()
	{
		return String.format("%-30s%8.2f%5d%8.2f", theProduct.getDescription(),
theProduct.getPrice(), quantity, getTotalPrice());
	}
}

/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 1 of Chapter 12.
	Describes an article, that only comes as a single charge, like shipping.
*/
public class LineItemFixedCharge extends LineItem
{
	/**
		Creates a fixed charge product.
	*/
	public LineItemFixedCharge(Product aProduct)
	{
		setQuantity(1);
		setProduct(aProduct);
	}
}

import java.util.ArrayList;
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 1 of Chapter 12.
	Describes an invoice for a set of purchased products.
*/
public class Invoice
{
	private Address billingAddress;
	private ArrayList<LineItem> items;

	/**
```

```
		Constructs an invoice
		@param anAddress the billing address
	*/
	public Invoice(Address anAddress)
	{
		items = new ArrayList<LineItem>();
		billingAddress = anAddress;
	}

	/**
		Adds a charge for a product to this invoice.
		@param aProduct the product that the customer ordered
		@param quantity the quantity of the product
	*/
	public void add(Product aProduct, int quantity)
	{
		LineItem anItem = new LineItem(aProduct, quantity);
		items.add(anItem);
	}

	/**
		Adds a charge for a fixed price product to this invoice.
		@param aProduct the fixed charge of the customer's order
	*/
	public void add(Product aProduct)
	{
		LineItemFixedCharge anItem = new LineItemFixedCharge(aProduct);
		items.add(anItem);
	}

	/**
		Formats the invoice
		@return the formatted invoice
	*/
	public String format()
	{
		String r = "                    I N V O I C E\n\n" + billingAddress.format() +
String.format("\n\n%-30s%8s%5s%8s\n", "Description", "Price", "Qty", "Total");
		for(LineItem item : items)
		{
			r = r + item.format() + "\n";
		}
		r = r + String.format("\nAMOUNT DUE: $%8.2f", getAmountDue());

		return r;
	}

	/**
		Computes the total amount due.
		@return the amount due
	*/
	public double getAmountDue()
	{
		double amountDue = 0;
		for(LineItem item : items)
		{
			amountDue = amountDue + item.getTotalPrice();
		}
		return amountDue;
	}
}

/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 1 of Chapter 12.
	This program demonstrates the invoice classes by printing a simple invoice.
*/

public class InvoicePrinter
{
	public static void main(String[] args)
	{
		Address samsAddress = new Address("Sam's Small Appliances", "100 Main
Street", "Anytown", "CA", "98765");

		Invoice samsInvoice = new Invoice(samsAddress);
		samsInvoice.add(new Product("Toaster", 29.95), 3);
		samsInvoice.add(new Product("Hair dryer", 24.95), 1);
		samsInvoice.add(new Product("Car vacuum", 19.99), 2);
		samsInvoice.add(new Product("Shipping", 5.99));

		System.out.println(samsInvoice.format());
	}
}
```

UML Diagram





```
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket> java InvoicePrinter
                      I N V O I C E

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Description                       Price  Qty   Total
Toaster                           29.95    3   89.85
Hair dryer                        24.95    1   24.95
Car vacuum                        19.99    2   39.98
Shipping                           5.99    1    5.99

AMOUNT DUE: $  160.77
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket>
```

Exercise 3

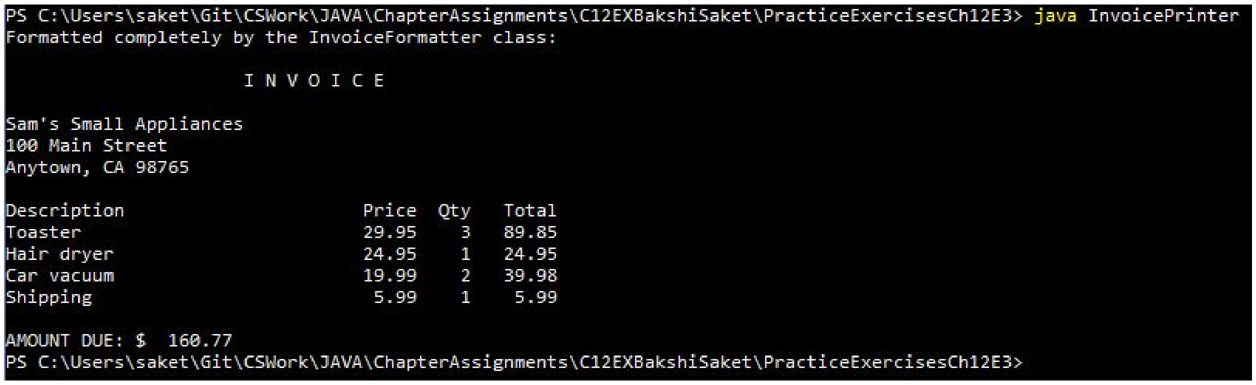
```
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 3 of Chapter 12.
	Describes a mailing address.
*/
public class Address
{
	private String name;
	private String street;
	private String city;
	private String state;
	private String zip;

	/**
		Constructs a mailing address
		@param aName the recipient name
		@param aStreet the street
		@param aCity the city
		@param aState the two-letter state code
		@param aZip the ZIP postal code
	*/
	public Address(String aName, String aStreet, String aCity, String aState, String aZip)
	{
		name = aName;
		street = aStreet;
		city = aCity;
		state = aState;
		zip = aZip;
	}

	/**
		Returns the name.
		@return the name
	*/
	public String getName()
	{
		return name;
	}

	/**
		Returns the street.
		@return the street
	*/
	public String getStreet()
	{
		return street;
	}

	/**
		Returns the city.
		@return the city
	*/
	public String getCity()
	{
		return city;
	}

	/**
		Returns the state.
		@return the state
	*/
	public String getState()
	{
		return state;
	}

	/**
		Returns the zip.
		@return the zip
	*/
	public String getZip()
	{
		return zip;
	}
}

/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 3 of Chapter 12.
	Describes a product with a description and a price.
*/
public class Product
{
	private String description;
	private double price;
	/**
		Constructs a product from a description and a price.
		@param aDescription the product description
		@param aPrice the product price
	*/
	public Product(String aDescription, double aPrice)
	{
		description = aDescription;
		price = aPrice;
	}

	/**
		Gets the product description.
		@return the description
	*/
	public String getDescription()
	{
		return description;
	}

	/**
		Gets the product price
		@return the unit price
	*/
	public double getPrice()
	{
		return price;
	}
}

/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 3 of Chapter 12.
	Describes a quantity of an article to purchase.
*/
public class LineItem
{
	private int quantity;
	private Product theProduct;

	/**
		Constructs a blank LineItem
	*/
	public LineItem()
```

```
{
	theProduct = new Product("", 0);
	quantity = 0;
}

/**
	Constructs an item from the product and the quantity.
	@param aProduct the product
	@param aQuantity the item quantity
*/
public LineItem(Product aProduct, int aQuantity)
{
	theProduct = aProduct;
	quantity = aQuantity;
}

/**
	Sets the quantity of the product
*/
public void setQuantity(int aQuantity)
{
	quantity = aQuantity;
}

/**
	Sets the product name
*/
public void setProduct(Product aProduct)
{
	theProduct = aProduct;
}

/**
	Computes the total cost of this line item.
	@return the total price
*/
public double getTotalPrice()
{
	return theProduct.getPrice() * quantity;
}

/**
	Returns the product.
```

```
		@return the product
	*/
	public Product getProduct()
	{
		return theProduct;
	}

	/**
		Returns the product.
		@return the product
	 */
	public int getQuantity()
	{
		return quantity;
	}
}

/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 3 of Chapter 12.
	Describes an article, that only comes as a single charge, like shipping.
*/
public class LineItemFixedCharge extends LineItem
{
	/**
		Creates a fixed charge product.
	*/
	public LineItemFixedCharge(Product aProduct)
	{
		setQuantity(1);
		setProduct(aProduct);
	}
}

import java.util.ArrayList;
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 3 of Chapter 12.
	Describes an invoice for a set of purchased products.
*/
public class Invoice
{
	private Address billingAddress;
	private ArrayList<LineItem> items;
```

```
/**
        Constructs an invoice
        @param anAddress the billing address
*/
public Invoice(Address anAddress)
{
        items = new ArrayList<LineItem>();
        billingAddress = anAddress;
}

/**
        Adds a charge for a product to this invoice.
        @param aProduct the product that the customer ordered
        @param quantity the quantity of the product
*/
public void add(Product aProduct, int quantity)
{
        LineItem anItem = new LineItem(aProduct, quantity);
        items.add(anItem);
}

/**
        Adds a charge for a fixed price product to this invoice.
        @param aProduct the fixed charge of the customer's order
*/
public void add(Product aProduct)
{
        LineItemFixedCharge anItem = new LineItemFixedCharge(aProduct);
        items.add(anItem);
}

/**
        Computes the total amount due.
        @return the amount due
*/
public double getAmountDue()
{
        double amountDue = 0;
        for(LineItem item : items)
        {
                amountDue = amountDue + item.getTotalPrice();
        }
        return amountDue;
```

```
	}

	/**
		Returns the mailing address.
		@return the address
	*/
	public Address getAddress()
	{
		return billingAddress;
	}

	/**
		Returns the list of items.
		@return the address
	*/
	public ArrayList<LineItem> getItems()
	{
		return items;
	}
}

import java.util.ArrayList;
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 3 of Chapter 12.
	Formats an invoice to be printed.
*/
public class InvoiceFormatter
{
	private Address billingAddress;
	private ArrayList<LineItem> items;
	private double amountDue;

	public InvoiceFormatter(Invoice anInvoice)
	{
		billingAddress = anInvoice.getAddress();
		items = anInvoice.getItems();
		amountDue = anInvoice.getAmountDue();
	}

	public String format()
	{
		String r = "			I N V O I C E\n\n";
		r = r + billingAddress.getName() + "\n" + billingAddress.getStreet() + "\n" +
billingAddress.getCity() + ", " + billingAddress.getState() + " " + billingAddress.getZip();
		r = r + String.format("\n\n%-30s%8s%5s%8s\n", "Description", "Price", "Qty",
"Total");
		for(LineItem item : items)
		{
			Product theProduct = item.getProduct();
			r = r + String.format("%-30s%8.2f%5d%8.2f", theProduct.getDescription(),
theProduct.getPrice(), item.getQuantity(), item.getTotalPrice()) + "\n";
		}

		r = r + String.format("\nAMOUNT DUE: $%8.2f", amountDue);

		return r;
	}
}

/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 3 of Chapter 12.
	This program demonstrates the invoice classes by printing a simple invoice.
*/

public class InvoicePrinter
{
	public static void main(String[] args)
	{
		Address samsAddress = new Address("Sam's Small Appliances", "100 Main
Street", "Anytown", "CA", "98765");

		Invoice samsInvoice = new Invoice(samsAddress);
		samsInvoice.add(new Product("Toaster", 29.95), 3);
		samsInvoice.add(new Product("Hair dryer", 24.95), 1);
		samsInvoice.add(new Product("Car vacuum", 19.99), 2);
		samsInvoice.add(new Product("Shipping", 5.99));

		InvoiceFormatter formatter = new InvoiceFormatter(samsInvoice);

		System.out.println("Formatted completely by the InvoiceFormatter class: \n");
		System.out.println(formatter.format());
	}
}
```

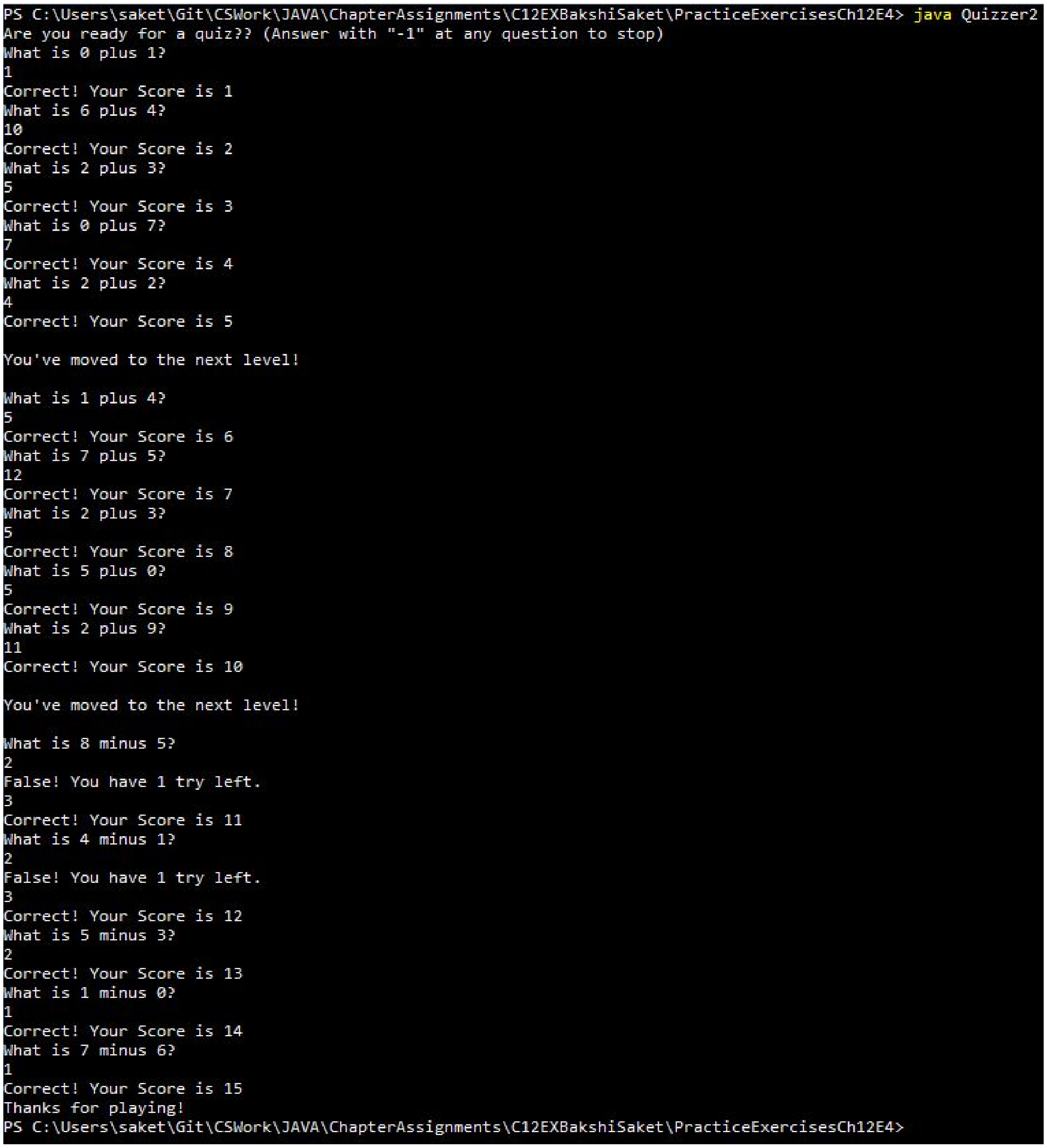


```
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket\PracticeExercisesCh12E3> java InvoicePrinter
Formatted completely by the InvoiceFormatter class:

                      I N V O I C E

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Description                      Price  Qty   Total
Toaster                          29.95    3   89.85
Hair dryer                       24.95    1   24.95
Car vacuum                       19.99    2   39.98
Shipping                          5.99    1    5.99

AMOUNT DUE: $  160.77
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket\PracticeExercisesCh12E3>
```

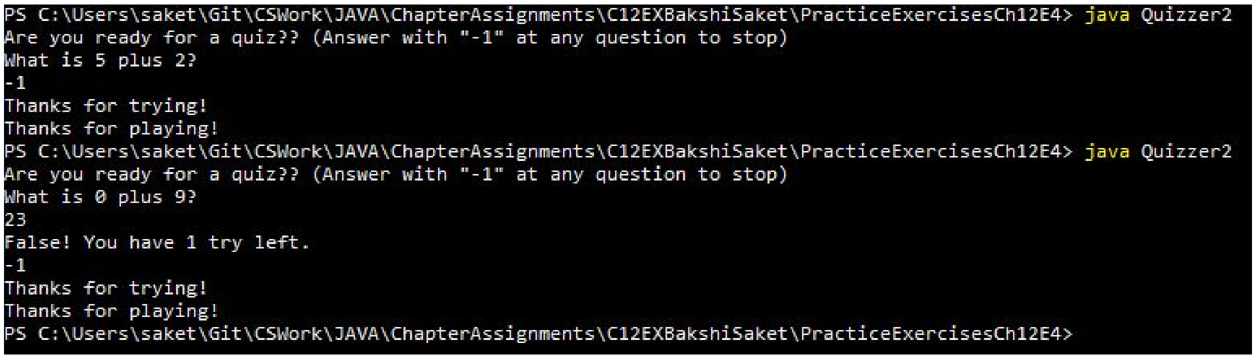
Exercise 4

```
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 4 of Chapter 12.
	Describes the level of question needed and the amount of tries taken to answer a
question. Also counts a score of correctly answered questions.
*/
public class Game
{
	private int lvl;
	private int tries;
	private int score;

	/**
		Creates an Arthimatic Game object to track and score kids as they practice math.
	*/
	public Game()
	{
		lvl = 1;
		tries = 2;
		score = 0;
	}

	/**
		Returns the current level of question.
		@return the level
	*/
	public int getLvl()
	{
		return lvl;
	}

	/**
		Returns the amount of tries that are left.
		@return the current tries left
	*/
	public int getTries()
	{
		return tries;
	}

	/**
		Returns the current score
		@return the score
	*/
	public int getScore()
	{
		return score;
	}

	/**
		Resets the amount of tries left.
	*/
	public void resetTries()
	{
		tries = 2;
	}

	/**
		Increases the score by 1.
	*/
	public void increaseScore()
	{
		score++;
	}

	/**
		Increases the level of question difficulty by 1.
	*/
	public void increaseLevel()
	{
		lvl++;
	}
}

import java.util.Random;
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 4 of Chapter 12.
	Creates questions for kids to solve.
*/
public class QuestionGenerator
{
	private int firstNumber;
	private int secondNumber;
	private int answer;
	private String question;
	/**
		Creates a question for teaching kids. Creates different questions depending on
the level of question required.
		@param lvl the level of question required
	*/
	public QuestionGenerator(int lvl)
	{
		Random rand = new Random();
		switch(lvl)
		{
			case 1: //level 1 question
				answer = rand.nextInt(11);
				firstNumber = rand.nextInt(answer);
				secondNumber = answer - firstNumber;
				question = "What is " + firstNumber + " plus " + secondNumber +
"?";
				break;
			case 2: //level 2 question
				firstNumber = rand.nextInt(10);
				secondNumber = rand.nextInt(10);
				answer = firstNumber + secondNumber;
				question = "What is " + firstNumber + " plus " + secondNumber +
"?";
				break;
			case 3: //level 3 question
				firstNumber = rand.nextInt(10);
				secondNumber = rand.nextInt(firstNumber);
				answer = firstNumber - secondNumber;
				question = "What is " + firstNumber + " minus " + secondNumber +
"?";
				break;
			default:
				// System.out.println("There is currently no level this high");
				break;
		}
	}

	/**
		Returns the number for part 1.
		@return the first number
	*/
	public int getFirstNumber()
	{
		return firstNumber;
	}

	/**
		Returns the number for part 2.
		@return the second number
	*/
	public int getSecondNumber()
	{
		return secondNumber;
	}

	/**
		Returns the answer.
		@return the answer
	*/
	public int getAnswer()
	{
		return answer;
	}

	/**
		Returns the question.
		@return the question
	*/
	public String getQuestion()
	{
		return question;
	}
}

import java.util.Scanner;
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for problem 4 of Chapter 12.
	Quizzes children to teach arithmetic.
*/
public class Quizzer2
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		System.out.println("Are you ready for a quiz?? (Answer with \"-1\" at any question
to stop)");

		Game game = new Game();

		boolean gameOver = false;
		int answer = 0;
		boolean wantsToPlay = true;
		while((game.getScore() != 5 || answer == -1) && wantsToPlay)
		{
			QuestionGenerator question = new QuestionGenerator(game.getLvl());
			System.out.println(question.getQuestion());
			boolean correct = false;
			while(game.getTries() > 0 && !correct && wantsToPlay)
			{
				answer = in.nextInt();
				if(answer == question.getAnswer())
				{
					game.increaseScore();
					correct = true;
					System.out.println("Correct! Your Score is " +
game.getScore());
				}
				else if(answer == -1)
				{
					System.out.println("Thanks for trying!");
					wantsToPlay = false;
				}
				else
				{
					game.decreaseTries();
					System.out.println("False! You have " + game.getTries() +
" try left.");
				}
			}
			game.resetTries();
		}
		game.increaseLevel();

		if(wantsToPlay)
			System.out.println("\nYou've moved to the next level!\n");

		while((game.getScore() != 10 || answer == -1) && wantsToPlay)
		{
			QuestionGenerator question = new QuestionGenerator(game.getLvl());
			System.out.println(question.getQuestion());
			boolean correct = false;
			while(game.getTries() > 0 && !correct && wantsToPlay)
			{
				answer = in.nextInt();
				if(answer == question.getAnswer())
				{
					game.increaseScore();
					correct = true;
					System.out.println("Correct! Your Score is " +
game.getScore());
				}
				else if(answer == -1)
				{
					System.out.println("Thanks for trying!");
					wantsToPlay = false;
				}
				else
				{
					game.decreaseTries();
					System.out.println("False! You have " + game.getTries() +
" try left.");
				}
			}
			game.resetTries();
		}
		game.increaseLevel();

		if(wantsToPlay)
			System.out.println("\nYou've moved to the next level!\n");

		while((game.getScore() != 15 || answer == -1) && wantsToPlay)
		{
			QuestionGenerator question = new QuestionGenerator(game.getLvl());
			System.out.println(question.getQuestion());
			boolean correct = false;
			while(game.getTries() > 0 && !correct && wantsToPlay)
			{
				answer = in.nextInt();
				if(answer == question.getAnswer())
				{
```

```
                    game.increaseScore();
                    correct = true;
                    System.out.println("Correct! Your Score is " +
game.getScore());
                }
                else if(answer == -1)
                {
                    System.out.println("Thanks for trying!");
                    wantsToPlay = false;
                }
                else
                {
                    game.decreaseTries();
                    System.out.println("False! You have " + game.getTries() +
" try left.");
                }
            }
            game.resetTries();
        }
        System.out.println("Thanks for playing!");

    }
}
```



```
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket\PracticeExercisesCh12E4> java Quizzer2
Are you ready for a quiz?? (Answer with "-1" at any question to stop)
What is 0 plus 1?
1
Correct! Your Score is 1
What is 6 plus 4?
10
Correct! Your Score is 2
What is 2 plus 3?
5
Correct! Your Score is 3
What is 0 plus 7?
7
Correct! Your Score is 4
What is 2 plus 2?
4
Correct! Your Score is 5

You've moved to the next level!

What is 1 plus 4?
5
Correct! Your Score is 6
What is 7 plus 5?
12
Correct! Your Score is 7
What is 2 plus 3?
5
Correct! Your Score is 8
What is 5 plus 0?
5
Correct! Your Score is 9
What is 2 plus 9?
11
Correct! Your Score is 10

You've moved to the next level!

What is 8 minus 5?
2
False! You have 1 try left.
3
Correct! Your Score is 11
What is 4 minus 1?
2
False! You have 1 try left.
3
Correct! Your Score is 12
What is 5 minus 3?
2
Correct! Your Score is 13
What is 1 minus 0?
1
Correct! Your Score is 14
What is 7 minus 6?
1
Correct! Your Score is 15
Thanks for playing!
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket\PracticeExercisesCh12E4>
```



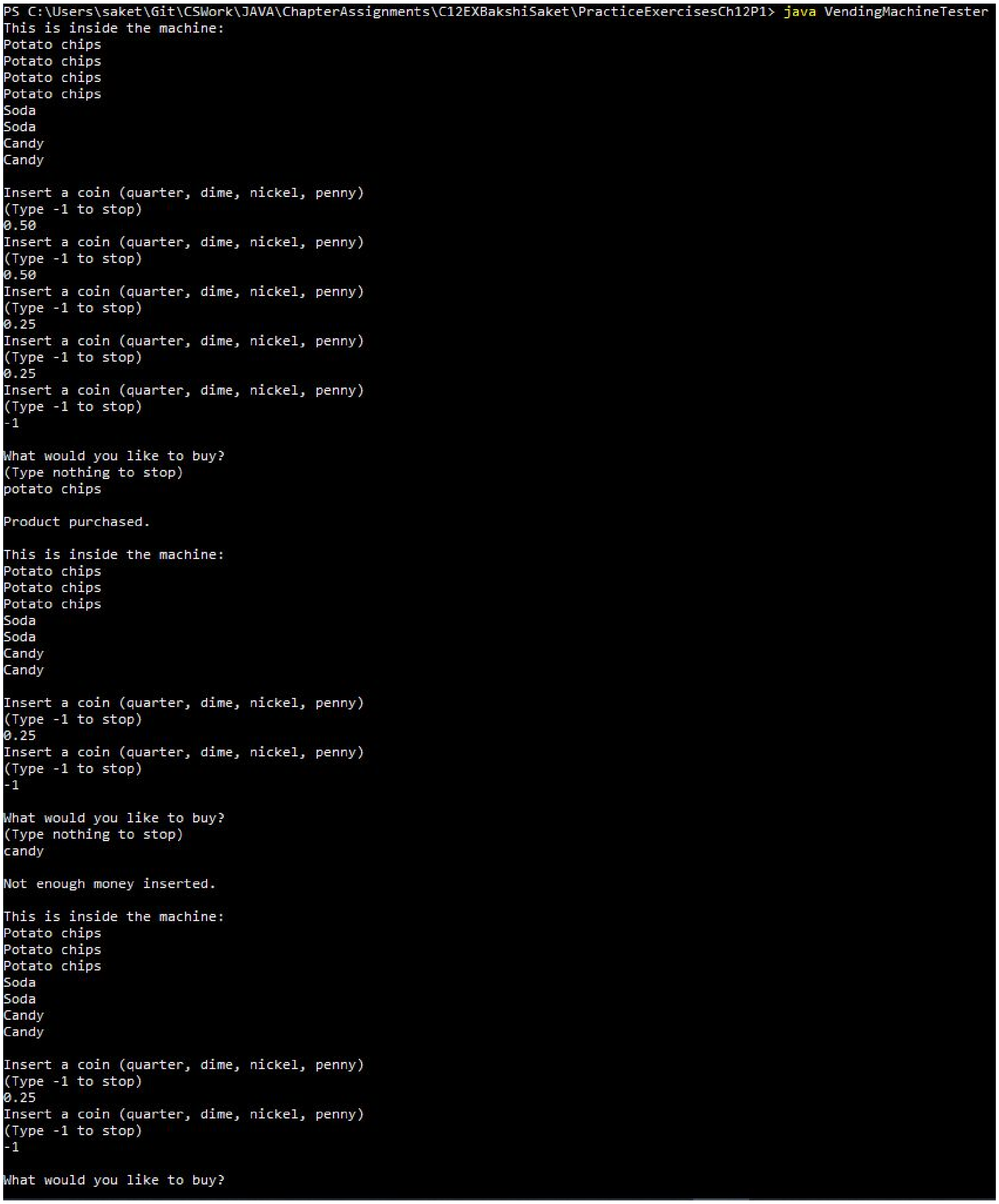
```
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket\PracticeExercisesCh12E4> java Quizzer2
Are you ready for a quiz?? (Answer with "-1" at any question to stop)
What is 5 plus 2?
-1
Thanks for trying!
Thanks for playing!
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket\PracticeExercisesCh12E4> java Quizzer2
Are you ready for a quiz?? (Answer with "-1" at any question to stop)
What is 0 plus 9?
23
False! You have 1 try left.
-1
Thanks for trying!
Thanks for playing!
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket\PracticeExercisesCh12E4>
```

Project 1

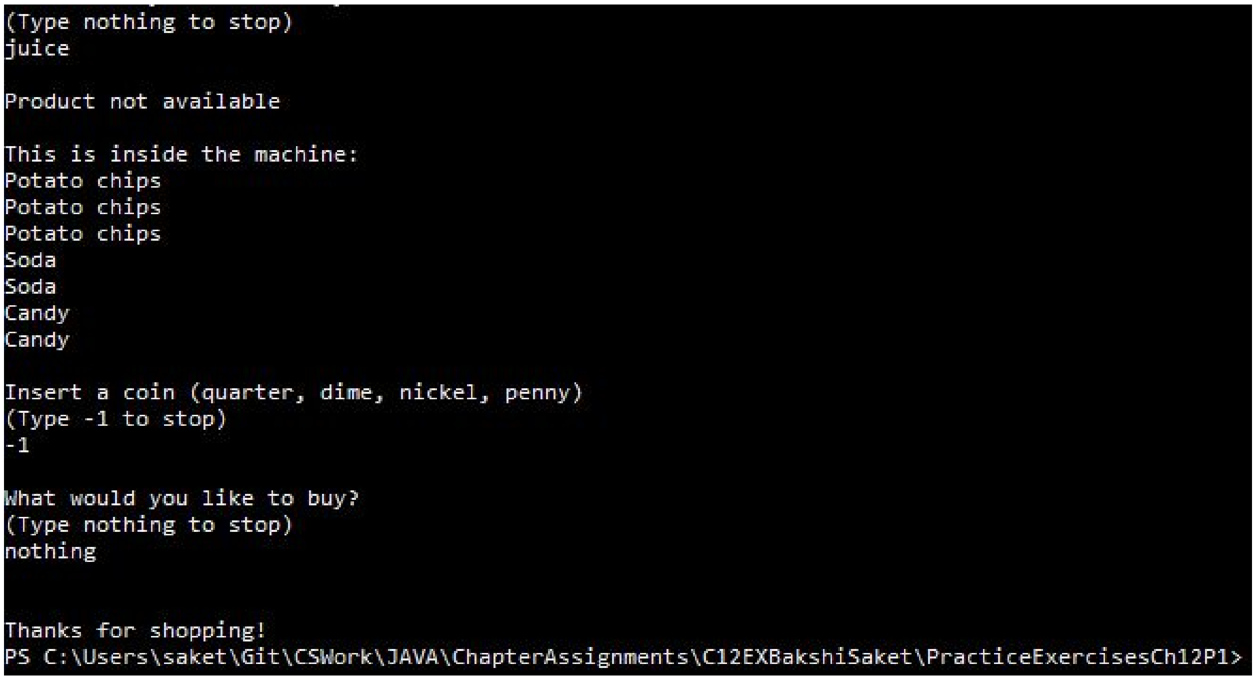
```
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for project 1 of Chapter 12.
	Makes a product with a simple description and cost.
*/
public class Product
{
	private String description;
	private double cost;

	/**
		Creates a product with a description and price.
	*/
	public Product()
	{
		description = "";
		cost = 0;
	}

	/**
		Creates a product with a description and price.
		@param aDescription the description
		@param cost the price
	*/
	public Product(String aDescription, double cost)
	{
		description = aDescription;
		this.cost = cost;
	}

	/**
		Returns the description of the product.
		@return the description
	*/
	public String getDescription()
	{
		return description;
	}

	/**
		Returns the cost of the product.
		@return the cost
	*/
	public double getCost()
	{
		return this.cost;
	}
}

import java.util.ArrayList;
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for project 1 of Chapter 12.
	Holds Products for purchase.
*/
public class VendingMachine
{
	private ArrayList<Product> items;

	/**
		Creates a vending machine.
	*/
	public VendingMachine()
	{
		items = new ArrayList<Product>();
	}

	/**
		Creates a vending machine with the items listed.
		@param items the items included
	*/
	public VendingMachine(ArrayList<Product> items)
	{
		this.items = items;
	}

	/**
		Adds products to the machine.
		@param aProduct the product to add
	*/
	public void add(Product aProduct)
	{
		items.add(aProduct);
	}

	/**
```

```
		Removes products from the machine.
		@param aProduct the product to remove
	*/
	public void remove(Product aProduct)
	{
		int placement = 0;
		for(int i = 0; i < items.size(); i++)
		{
			if(aProduct.getDescription().equals(items.get(i).getDescription()))
			{
				placement = i;
			}
		}
		items.remove(placement);
	}

	/**
		Returns all the items currently in the machine.
		@return all the items
	*/
	public ArrayList<Product> getItems()
	{
		return items;
	}

	/**
		Prints what is inside the machine.
	*/
	public void whatsInside()
	{
		for(Product item : items)
		{
			System.out.println(item.getDescription());
		}
	}
}

import java.util.ArrayList;
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for project 1 of Chapter 12.
	Handles transactions with a vending machine.
*/
public class Cashier
```

```
{
	private double valueInserted;
	private VendingMachine aMachine;

	/**
		Creates a cashier to handle transactions between a buyer and a vending
machine.
	*/
	public Cashier()
	{
		valueInserted = 0;
		aMachine = null;
	}

	/**
		Creates a cashier to handle transactions between a buyer and a vending
machine.
		@param aMachine the machine being handled
	*/
	public Cashier(VendingMachine aMachine)
	{
		this.aMachine = aMachine;
		valueInserted = 0;
	}

	/**
		Adds a coin to the cashier for purchasing.
		@param value the value of currency added
	*/
	public void addCoin(double value)
	{
		valueInserted += value;
	}

	/**
		Attempts to make a transaction with a specified product.
		@param productDescription the product wanted
		@return the message about the transaction
	*/
	public String attemptTransaction(String productDescription)
	{
		boolean hasProduct = false;
		ArrayList<Product> items = aMachine.getItems();
		int placement = 0;
		Product wantedProduct = null;
		for(int a = 0; a < items.size(); a++)
		{
if(items.get(a).getDescription().toLowerCase().equals(productDescription.toLowerCase()))
			{
				hasProduct = true;
				placement = a;
				wantedProduct = items.get(a);
				a = items.size();
			}
		}

		if(hasProduct)
		{
			if(wantedProduct.getCost() <= valueInserted)
			{
				valueInserted = 0;
				items.remove(placement);
				aMachine = new VendingMachine(items);
				return "Product purchased.";
			}
			else
				return "Not enough money inserted.";
		}
		else
			return "Product not available";
	}
}

import java.util.Scanner;
/**
	Saket Bakshi. 3/28/19. Period 6. This is used for project 1 of Chapter 12.
	Tests the vending machine project.
*/
public class VendingMachineTester
{
	public static void main(String[] args)
	{
		Scanner keyboard = new Scanner(System.in);
		VendingMachine theVendingMachine = new VendingMachine();

		theVendingMachine.add(new Product("Potato chips", 0.99));
		theVendingMachine.add(new Product("Potato chips", 0.99));
		theVendingMachine.add(new Product("Potato chips", 0.99));
		theVendingMachine.add(new Product("Potato chips", 0.99));

		theVendingMachine.add(new Product("Soda", 1.99));
		theVendingMachine.add(new Product("Soda", 1.99));

		theVendingMachine.add(new Product("Candy", 2.99));
		theVendingMachine.add(new Product("Candy", 2.99));

		Cashier vendor = new Cashier(theVendingMachine);

		String productWanted = "";
		do
		{
		System.out.println("This is inside the machine: ");
		theVendingMachine.whatsInside();
		System.out.println();

		double value = 0;
		while(value != -1)
		{
			System.out.println("Insert a coin (quarter, dime, nickel, penny)\n(Type -1
to stop)");
			value = keyboard.nextDouble();
			if(value != -1)
				vendor.addCoin(value);
		}
		keyboard.nextLine();
		System.out.println();

		System.out.println("What would you like to buy?\n(Type nothing to stop)");
		productWanted = keyboard.nextLine();
		System.out.println();

		if(!productWanted.equals("nothing"))
			System.out.println(vendor.attemptTransaction(productWanted));

		System.out.println();
		} while(!productWanted.equals("nothing"));
		System.out.println("Thanks for shopping!");

		keyboard.close();
	}
}
```



```
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket\PracticeExercisesCh12P1> java VendingMachineTester
This is inside the machine:
Potato chips
Potato chips
Potato chips
Potato chips
Soda
Soda
Candy
Candy

Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
0.50
Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
0.50
Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
0.25
Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
0.25
Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
-1

What would you like to buy?
(Type nothing to stop)
potato chips

Product purchased.

This is inside the machine:
Potato chips
Potato chips
Potato chips
Soda
Soda
Candy
Candy

Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
0.25
Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
-1

What would you like to buy?
(Type nothing to stop)
candy

Not enough money inserted.

This is inside the machine:
Potato chips
Potato chips
Potato chips
Soda
Soda
Candy
Candy

Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
0.25
Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
-1

What would you like to buy?
```



```
(Type nothing to stop)
juice

Product not available

This is inside the machine:
Potato chips
Potato chips
Potato chips
Soda
Soda
Candy
Candy

Insert a coin (quarter, dime, nickel, penny)
(Type -1 to stop)
-1

What would you like to buy?
(Type nothing to stop)
nothing


Thanks for shopping!
PS C:\Users\saket\Git\CSWork\JAVA\ChapterAssignments\C12EXBakshiSaket\PracticeExercisesCh12P1>
```