Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

COMP122 - Assessment 4

Information

Name:	Oliver Legg
Email:	O.Legg@student.liverpool.ac.uk
Student ID:	201244658

Contents:

Requirements

Part 1

For part 1 requirements, I must extend java's JFrame swing class to create window like shown below.

I must sell 9 products on this window and implement stock so that when the user purchases an item, the stock decreases (obviously not below 0). If I try to buy anything below 0 a window should pop up telling me that I can't purchase the item.

Afterwards I must implement a vendor information button that opens a vendor information frame. In this frame it will display the total sales I have made. It will have another button which resets the stock too and sets the total sales back to 0.

Closing the main window should exit the program. However, closing the "Vendor Information" window should not close the application, nor should closing the other "no stock left" message windows.

Part 2

In this assignment I am required to call the application program "Doors.java" and use an abstract class for subclasses to be extended from. The way my program will start is to take 1 parameter N (integer). Which will

represent the number of doors. This must be in a range from 1 to 1000000. I should also error handle input here. This will be done by giving the user a helpful tip if they enter something incorrectly - like a string. After a subsequently invalid input, the program will exit. In the program I will output the doors they open and close. Obviously, before they perform they will have to have all their doors closed (set to 0).

Analysis and Design

Part 1

In part 1 my goal is to develop a GUI. Specifically, I will be building the basic GUI of a Vending Machine. Obviously, there's no actual vending machine that will be used here, but I will be developing a frame that shows nine products, each having a stock level and a price.

The idea is that when I press a button, the corresponding item will have its stock decrease and the total sales that would increase by the amount of the item that has been sold.

The GUI will also include another button that will open a "Vendor's Window". This window will show the total amount of sales (since the last time this button was pressed). This window will also have a button that will reset the stock levels to their start values, and will reset the total sales to 0

From this information I have gathered, I know I am going to use java swing to create my GUI. Knowing that I will be creating 2 frames:

- Main panel
- Vendor Information

I will create 2 classes that extend `JFrame` and implements `ActionListener`. This is because it will make it more organised separating the two frames. The action listener is there to create action when you press a button.

As there is a lot of information to be held about the candy, I intend to create a 'Candy' class. On instantiation, I will pass the name of the candy the price of the candy and the initial stock of it.

When I am programming my frames, I must make the labels public to other files so that the other files can change the text. This is because I will interact with something on frame 1 which will consequent frame 2 to change its data too. Therefore, both classes will have to 'use' each other.

When someone is buying a product, my vending machine shouldn't allow the user to buy an item if it out of stock. If they try to do this, they will get an information message that they can't buy the item because it is out of stock.

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Part 2

Now for this section, I will be considering a situation where there are people who like to open and shut doors in possibly strange fashions. There will be 3 types of people who open and doors in different a consistent fashion. The 3 people are:

- Ginny
- Petra
- Sven

Since these people will have similar properties, I will create an abstract super class which shares the similar traits. For example, Ginny, Petra and Sven would have the same attributes (with different data obviously) such as name, doors, number of doors, doors list etc. Not only this but they would share methods too – especially if attributes are private or even protected. These could be accessors like getName() or getList(). The method that runs the fashion that they open/close doors would be an abstract method that they could all use. I intend the super class to be called players which would be an abstract class. I intend to structure my program so that you instantiate in the subclasses and to check input error in a file called Doors.java. The input I will validate should be strict because my program will create an error if the user inputs a string when they meant to input an integer. I must also create a range of what numbers the user can enter. For example, their input should be 1-1000000. Any higher than that, it might create a memory leak.

When each person begins, I should have them start with a configuration of all closed doors. Then after performing the full procedure for that person, report on the number of doors that are open.

Ginny

Ginny toggles the door based on the number in the greatest common divisor (`gcd(N,k)`). The numbers that are inserted are the number of doors – 1 (n). The second parameter is the index of the doors. So, if there were 10 doors, the first door you would change would be `gcd(11,0)` which is 11. If the `gcd(11,0) == 1`, (which it does), then the doors would flip on the second parameters doors index e.g. my previous example wouldn't change door 0 because it doesn't equal to 1. On `gcd(11,1)` this would equal to 1. So the doors so far would be '01'.

Petra

Petra toggles doors based on it's index too – specifically – if it's a prime number. Petra doesn't consider 1 to be a prime number. p that is less than or equal to N, Petra toggles all doors that are prime. E.g

```
N = 9
```

0 and 1 are not prime therefore the first door Petra would flip would be 1.

```
0000000000
0010000000
```

Afterwards, I would flip 3.

O.Legg@student.liverpool.ac.uk

```
0011000000
```

Etc

Sven

Sven flips pairs of perfect square numbers. A perfect square number is something like (1,4,9). Because 1x1 = 1, 2x2 = 4, 3x3 = 9. However, Sven would get a pair of perfect square numbers like (1,4). The first number of the parameters is the distance Sven would walk in. After that the number would be 1+4. Therefore, the index of 1 and 5 would change. For example,

```
N = 1
Pair = (1,4)
0 1 2 3 4 5 6 //indexes
0 0 0 0 0 0 0 //before
0 1 0 0 0 1 0 //after
```

This would then repeat for all the pairs it can make. Obviously, there can't be a pair of (3, 5) because this would go to the index of 8 and that's out of range.

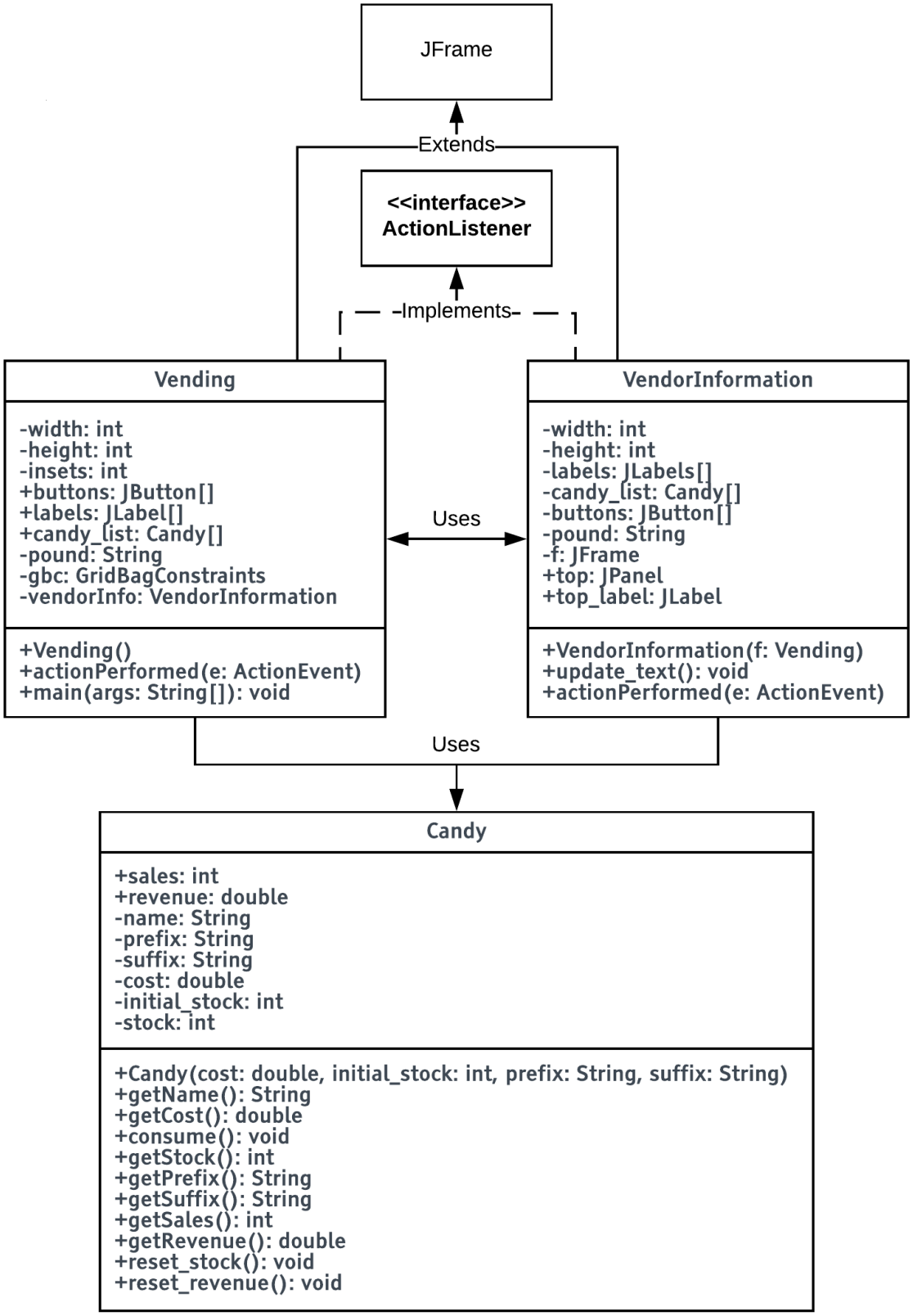
O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Class diagram

Part 1

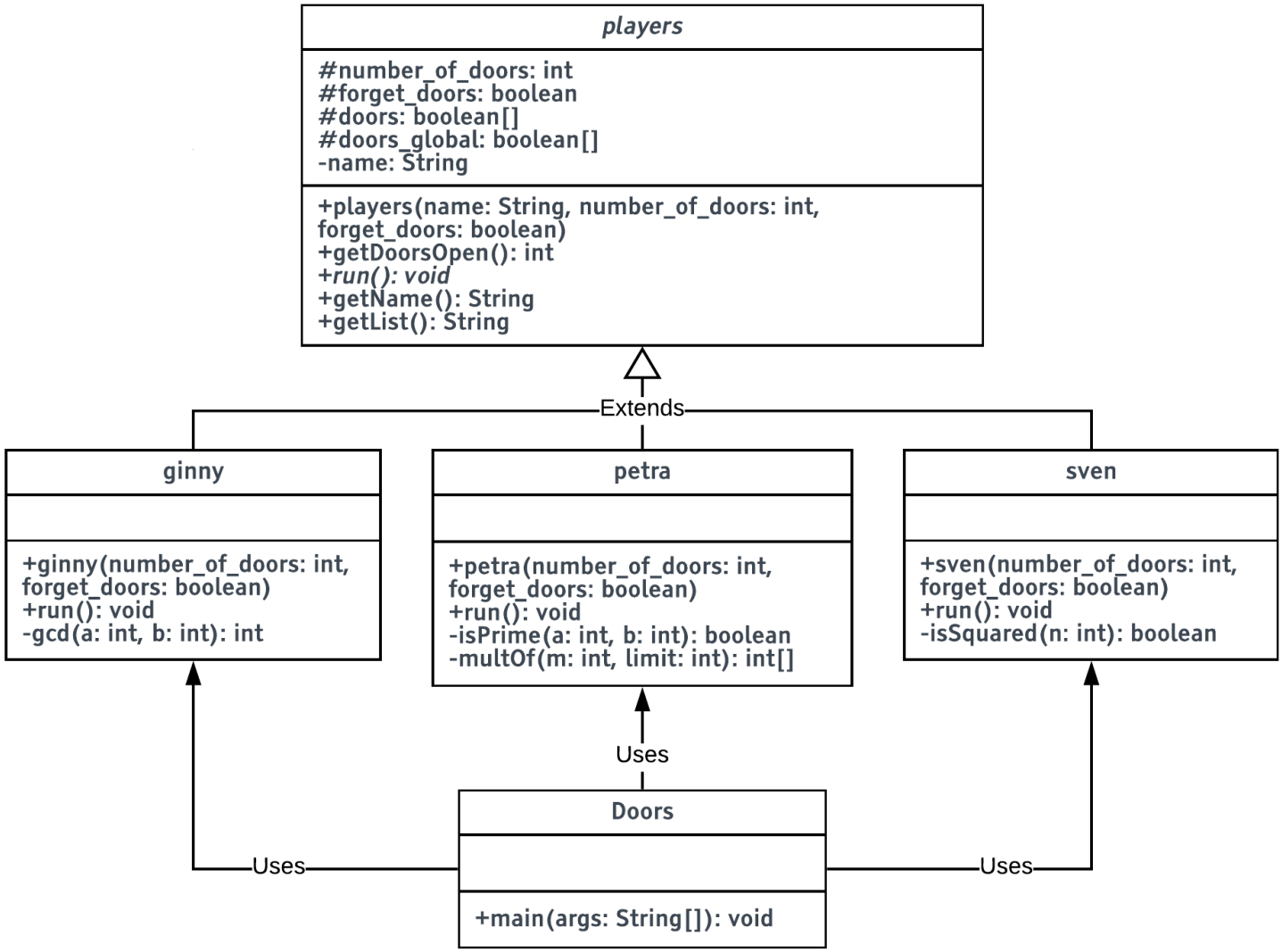


O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Part 2



O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Pseudocode

Part 1

```
Vending()
    setFrameName("Twisty JigglyBomb surprises")
    vendorInfo.SETVISIBLE(false)
    candy_list[0] := Candy("Chocolate", "Jigglypuffs", 1.30, 4)
    candy_list[1] := Candy("Caramel", "Jigglypuffs", 1.30, 4)
    candy_list[2] := Candy("French Vanilla", "Jigglypuffs", 1.30, 4)
    candy_list[3] := Candy("Chocolate", "Bombs", 1, 4)
    candy_list[4] := Candy("Caramel", "Bombs", 1, 4)
    candy_list[5] := Candy("French Vanilla", "Bombs", 1, 4)
    candy_list[6] := Candy("Chocolate", "Twists", 0.80, 4)
    candy_list[7] := Candy("Caramel", "Twists", 0.80, 4)
    candy_list[8] := Candy("French Vanilla", "Twists", 0.80, 4)
    x := 0
   y := 0
    -- THIS LOOP PLACES THE BUTTONS IN A 3x3 LAYOUT
    FOR i := 0 loop till i < buttons.length by i++ each step
        text  := candy_list[i].getName()
        stock := candy_list[i].getStock() + " left"
        cost  := "£"+candy_list[i].getCost()
        buttons[i] := JButton(text + cost)
        buttons[i].setPreferredSize(Dimension(257, 30)) -- width, height
        labels[i] := JLabel(stock)
        IF (i % 3 == 0)
            x++
            y := 0
        gridx := x
        y++
        gridy := y
        buttons[i].addActionListener(this)
        y++
        gridy := y
    b := JButton("Vendor Information")
    b.setPreferredSize(Dimension(257, 30))
    b.addActionListener(this)
    setBounds(0, 0, width, height)
    setVisible(true)
   setLocation(0, 70)

function actionPerformed(ActionEvent e)
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

```
    FOR i := 0 loop till i < candy_list.length by i++ each step
        IF (e.getActionCommand().equals(button[i].name))
            IF (candy_list[i].getStock() > 0)
                candy_list[i].consume()
                labels[i].setText(candy_list[i].getStock() + " left")
                vendorInfo.update_text()
            ELSE
                JOptionPane.showMessageDialog(NIL, "Oops. there are none left!")
    IF (e.getActionCommand().equals("Vendor Information"))
        vendorInfo.update_text()
        vendorInfo.repaint()
        vendorInfo.pack()
        vendorInfo.setVisible(true)

function main(String[] args)
    Vending()
```

```
VendorInformation(Vending f)
    setFrameName("Vendor Information")
    button.addActionListener(SELF)
    top_label := JLabel("Total Sales: " + "£" + Candy.getRevenue())
    setBounds(0, 0, width, height)

function update_text()
    top_label.setText("Total Sales: " + pound + "£" + Candy.getRevenue())

function actionPerformed(ActionEvent e)
    IF (e.getActionCommand().equals("Reset stock"))
        Candy.reset_revenue()
        FOR i := 0 loop till i < candy_list.length by i++ each step
            candy_list[i].reset_stock()
            labels[i].setText(candy_list[i].getStock() + " left")
            labels[i].setForeground(Color.BLACK)
        String temp := "£" + Candy.getRevenue()
        top_label.setText("Total Sales: " + pound + temp)
```

```
Candy(prefix, suffix, cost, initial_stock)
    prefix := prefix
    suffix := suffix
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

```
    name := prefix + " " + suffix
    cost := cost
    stock := initial_stock
    initial_stock := initial_stock

function getName()
    RETURN name

function getCost()
    RETURN cost

function getStock()
    RETURN stock

function getPrefix()
   RETURN prefix

function consume()
    IF (stock > 0)
        stock--
        sales++
        revenue := revenue + cost

function getSales()
    RETURN sales

function getRevenue()
    RETURN revenue

function reset_stock()
    stock := initial_stock

function reset_revenue()
    revenue := 0
```

Part 2

```
function main(String[] args)
    num_of_doors := 0
    testing := true
    IF (args.length <= 0)
        OUTPUT "\nOops, not enough arguments!"
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658


```
        OUTPUT "Usage: java Doors N (String)"
        EXIT()
    TRY
        num_of_doors := args[0]
    EXCEPT (VARTYPE)
        OUTPUT "\nOops, enter an integer!"
        OUTPUT "Usage: java Doors N (String)"
        EXIT()
    IF (num_of_doors < 1 OR num_of_doors > 1000000)
        OUTPUT "\nN must be between 1 and 1000000 !"
        OUTPUT "Usage: java Doors N (String)"
        EXIT()
    IF (args.length > 2)
        OUTPUT "\nOops, too many arguments!"
        OUTPUT "Usage: java Doors N (String)"
        EXIT()
    players[] p := players[0..2]
    p[0] := ginny(num_of_doors)
    p[1] := petra(num_of_doors)
    p[2] := sven(num_of_doors)
    FOR i := 0 loop till i < p.length by i++ each step
        OUTPUT p[i].getName()
        p[i].run()
        IF (testing)
            OUTPUT p[i].getList()
        OUTPUT p[i].getDoorsOpen() + " doors open"
```

```
players(String name, number_of_doors)
    name := name
    number_of_doors := number_of_doors
    doors := [number_of_doors-1]
    FOR i := 0 loop till i < doors.length by i++ each step
        doors[i] := false
function getDoorsOpen()
    count := 0
    FOR i := 0 loop till i < number_of_doors by i++ each step
        IF (doors[i])
            count++
    RETURN count

function ABSTRACT run()
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

```
function getName()
    RETURN name

function getList()
    String output := ""
    FOR i := 0 loop till i < number_of_doors by i++ each step
        IF (doors[i])
            output := output + "1"
        ELSE
            output := output + "0"
    RETURN output
```

```
ginny(number_of_doors):
    SUPER("Ginny", number_of_doors + 1)

function run()
    gcdlist := []
    FOR i := 0 loop till i < number_of_doors by i++ each step
        gcdlist[i] := gcd(number_of_doors - 1, i)
    FOR i := 0 loop till i < number_of_doors by i++ each step
        IF (number_of_doors == 2)
                    IF (doors[i])
                        doors[i] := false
                    ELSE
                        doors[i] := true
        IF (gcdlist[i] == 1)
                    IF (doors[i])
                        doors[i] := false
                    ELSE
                        doors[i] := true


function gcd(a, b)
    IF (a == 0 OR b == 0)
        RETURN 0
    ans := 1
    FOR i := 1 loop till (i <= a AND i <= b) by ++i each step
        IF (a % i == 0 AND b % i == 0)
            ans := i
    RETURN ans
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

```
petra(number_of_doors):
    SUPER("Petra", number_of_doors + 1)

function run()
    [] flippers
    FOR i := 0 loop till i < number_of_doors by i++ each step
        IF (i < 2)
            doors[i] := false
        ELSE
            IF (isPrime(i))
                flippers := multOf(i, (number_of_doors - 1))
                FOR v := 0 loop till v < flippers.LENGTH by v++ each step
                    IF (doors[flippers[v]])
                        doors[flippers[v]] := false
                    ELSE
                        doors[flippers[v]] := true

function isPrime(n)
    IF (n == 2 OR n == 3)
        RETURN true
    ELSE
        FOR i := 2 loop till i <= ROUNDUP(sqrt(n)) by i++ each step
            IF ((n % i) == 0)
                RETURN false
        RETURN true

function multOf(m, limit)
    newLimit := Math.ceil(limit / m)
    [] list := [0..newLimit + 1-1]
    FOR i := 0 loop till i <= (newLimit) by i++ each step
        list[i] := i * m
    RETURN list
```

```
    sven(number_of_doors):
        SUPER("Sven", number_of_doors + 1)

    function run()
        FOR i := 1 loop till i < number_of_doors by i++ each step
            FOR v := i + 1 loop till v < number_of_doors by v++ each step
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

```
            IF (isSquared(i) AND isSquared(v))
               IF ((i + v) < number_of_doors)
                  IF (doors[i])
                     doors[i] := false
                  ELSE
                     doors[i] := true
                  IF (doors[(i + v)])
                     doors[(i + v)] := false
                  ELSE
                     doors[(i + v)] := true

   function isSquared(n)
      n2 := Math.sqrt(n)
      IF (n2 == n2)
         RETURN true
      ELSE
         RETURN false
```

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Testing

Part 1 – To evidence

Test number	Description	Expected Results	Actual results	Remedial Action
1	Frame	The frame it's self appears and works in a readable layout.	Yes	
2	Buttons press	They're interactable	Yes	
3	Button press 2	Stock corresponding to product decreases when pressing buttons	Yes	
4	Stock validation	Stock should remain at 0 when trying to purchase another item	Yes	
5	InfoBox	When trying to purchase below 0, you should get an info box telling you that you can't purchase below 0	Yes	
6	Vendor Information Button	The vendor information frame must appear when you press the button	Yes	
7	Total sales	Correct number of total sales shows and updates correctly. Purchasing 1 Chocolate bomb. Should (£1.00)	Yes	
8	Reset stock button	Reset stock button should stock all of the products back up to it's original value and set the total sales back to £0.00	Yes	

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Part 2 – To Evidence

Test number	Description	Expected Results	Actual results	Remedial Action
1	Input validation 1	Entering no parameters when starting the program should produce a helpful message for the user and exit the program	Yes	
2	Input validation 2	Entering a number below 1 or above 1000000 should produce a helpful message for the user and exit the program	Yes	
3	Input validation 3	Entering a string should return a helpful message and exit the program	Yes	
4	Input Validation 4	Entering an extra parameter should create a helpful message and exit the program	Yes	
5	Sample output 1	Input 9 – see evidence for expected result (click the link on the subheading)	Yes	
6	Sample output 2	Input 30	Yes	
7	Sample output 3	Input 23	Yes	
8	Sample output 4	Input 1	Yes	
9	Sample output 5	Input 25000	Yes	
10	Sample output 6	Input 67325	Yes	

O.Legg@student.liverpool.ac.uk

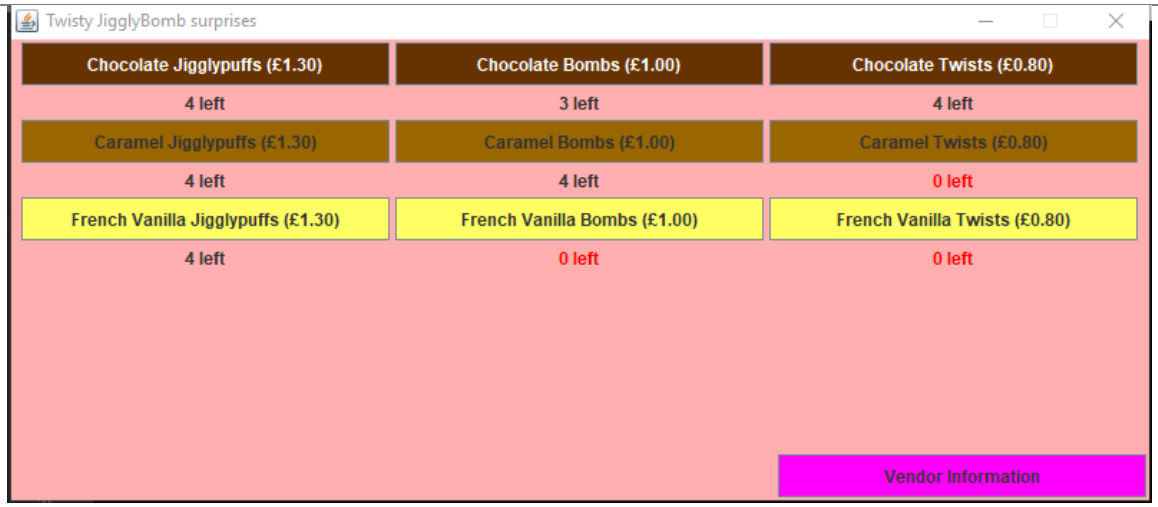
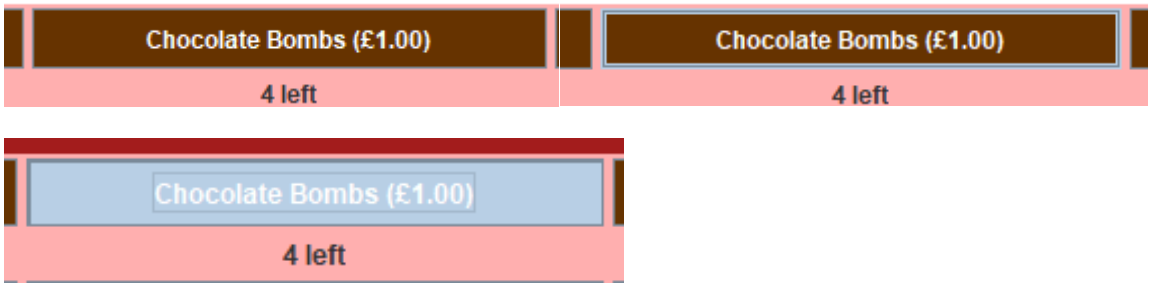
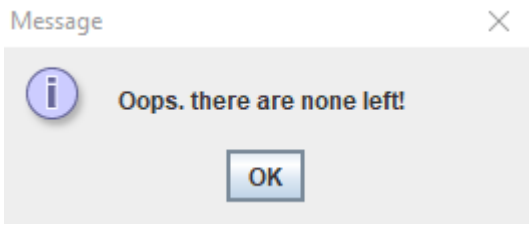
Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Testing Evidence

(Test number correspond with tests on previous section)

Part 1 – To tests

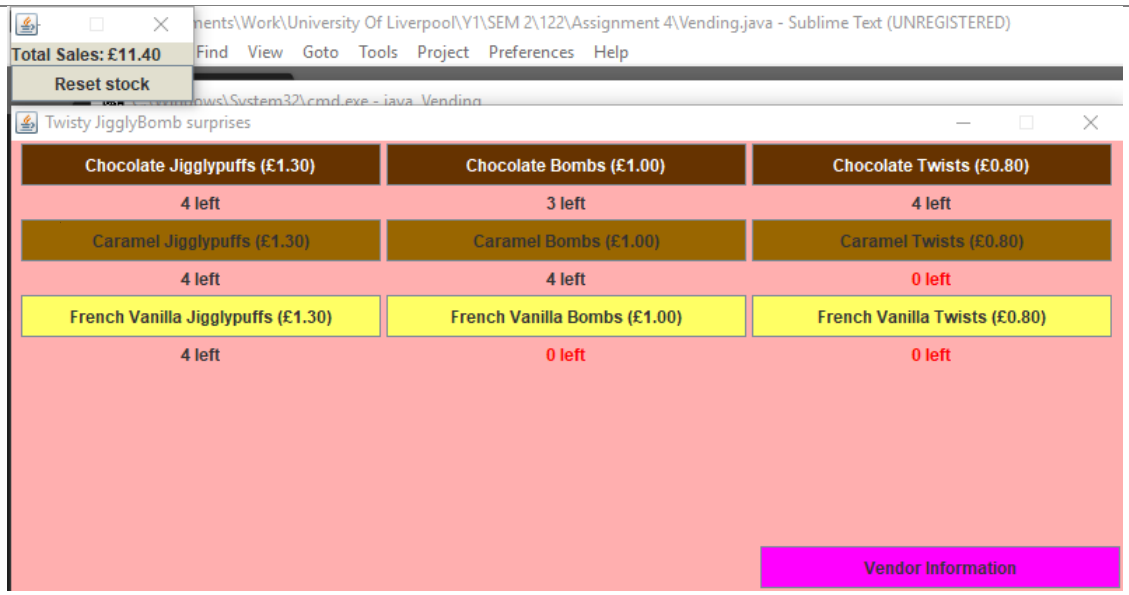
Test number	Screenshot Evidence
1	
2	
3	
4	
5	

O.Legg@student.liverpool.ac.uk

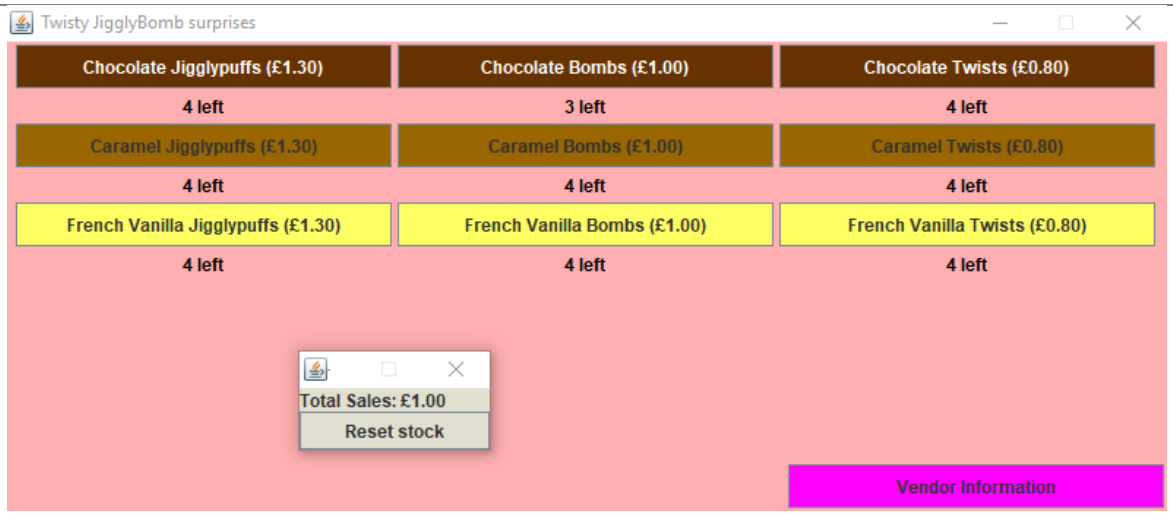
Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

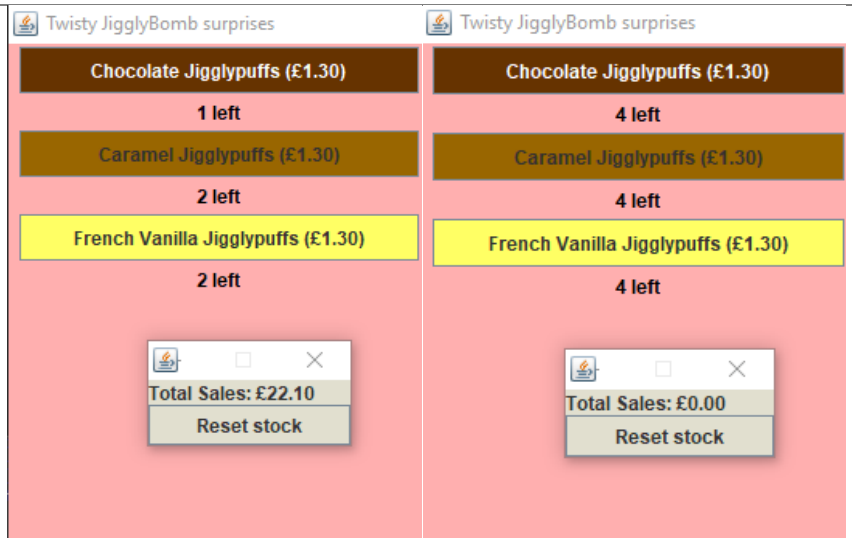
6



7



8



O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

Part 2 – To tests

Test no	Screenshot Evidence

1	`C:\Users\Olee\Documents\Work\University Of Liverpool\Y1\SEM 2\122\Assignment 4>java Doors`
	
	`Oops, not enough arguments!`
	`Usage: java Doors N (String)`

2	`C:\Users\Olee\Documents\Work\University Of Liverpool\Y1\SEM 2\122\Assignment 4>java Doors 0`
	
	`N must be between 1 and 1000000 !`
	`Usage: java Doors N (String)`
	
	`C:\Users\Olee\Documents\Work\University Of Liverpool\Y1\SEM 2\122\Assignment 4>java Doors 1000001`
	
	`N must be between 1 and 1000000 !`
	`Usage: java Doors N (String)`

3	`C:\Users\Olee\Documents\Work\University Of Liverpool\Y1\SEM 2\122\Assignment 4>java Doors er`
	
	`Oops, enter an integer!`
	`Usage: java Doors N (String)`

4	`C:\Users\Olee\Documents\Work\University Of Liverpool\Y1\SEM 2\122\Assignment 4>java Doors 9 9`
	
	`Invalid arg: 9`
	`Usage: java Doors N (String)`
	This appears like this because of the extra question which allows another parameter. However, it doesn't allow

5	`$ java Doors 9`
	`Ginny`
	`0110110110`
	`6 doors open`
	
	`Petra`
	`0011110111`
	`7 doors open`
	
	`Sven`
	`0100010000`
	`2 doors open`
	
	`>java Doors 9`
	`Ginny`
	`0110110110`
	`6 doors open`
	
	`Petra`
	`0011110111`
	`7 doors open`
	
	`Sven`
	`0100010000`
	`2 doors open`

6	`$ java Doors 30`
	`Ginny`
	`010000010001010001010001000001`
	`8 doors open`
	
	`Petra`
	`001111011101010011010001010101`
	`17 doors open`
	
	`Sven`
	`000011000110010001001000011001`
	`10 doors open`
	
	`>java Doors 30`
	`Ginny`
	`010000010001010001010001000001`
	`8 doors open`
	
	`Petra`
	`001111011101010011010001010101`
	`17 doors open`
	
	`Sven`
	`000011000110010001001000011001`
	`10 doors open`

O.Legg@student.liverpool.ac.uk

Name: Oliver Legg
Student ID: 201244658

University of Liverpool
COMP122

7

```
$ java Doors 23
Ginny
01111111111111111111110
22 doors open

Petra
10111101110101001101000 1
14 doors open

Sven
01000100001001000100100 0
6 doors open
```

```
>java Doors 23
Ginny
01111111111111111111110
22 doors open

Petra
10111101110101001101000 1
14 doors open

Sven
01000100001001000100100 0
6 doors open
```

8

```
$ java Doors 1
Ginny
11
2 doors open

Petra
00
0 doors open

Sven
00
0 doors open
```

```
>java Doors 1
Ginny
11
2 doors open

Petra
00
0 doors open

Sven
00
0 doors open
```

9

```
$ java Doors 25000
Ginny
10000 doors open

Petra
12593 doors open

Sven
4084 doors open
```

```
>java Doors 25000
Ginny
10000 doors open

Petra
12593 doors open

Sven
4084 doors open
```

10

```
$ java Doors 67325
Ginny
53840 doors open

Petra
33899 doors open

Sven
9880 doors open
```

```
>java Doors 67325
Ginny
53840 doors open

Petra
33899 doors open

Sven
9880 doors open
```

O.Legg@student.liverpool.ac.uk