

**AGROKART – A FARM TO HOME**  
**VIRTUAL MARKET**

**A MINI PROJECT**  
**REPORT**

*Submitted by*

**RAHUL MUSALIYATH DINESH**

*In partial fulfillment for the award of  
the degree of*

**Bachelor of Engineering**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

## ***Certificate***

*This is to certify that the mini project work titled*

**AGROKART – A FARM TO HOME**

**VIRTUAL MARKET**

*Submitted in partial fulfillment of the degree of*

*Bachelor of Engineering in*

*Computer Science and Engineering by*

**RAHUL MUSALIYATH DINESH**

**USN: 1NH18CS738**

*DURING*

*EVEN SEMESTER 2019-2020*

*for*

*COURSE CODE: 19CSE48*

Signature of Reviewer

Signature of HOD

### **SEMESTER END EXAMINATION**

*Name of the Examiner*

*Signature with date*

1. \_\_\_\_\_

\_\_\_\_\_

2. \_\_\_\_\_

\_\_\_\_\_

# **ABSTRACT**

Agriculture forms the backbone of the Indian economy by contributing almost 15% of the national gross domestic product i.e. GDP. A large number of people also depend on agriculture for their daily income and livelihood thus making the contribution of agriculture in India even more important.

In recent years due to factors varying from policies to natural calamities, farmers' have been seeing a decline in their income due to which they are not able to pay back the loans and dues they have thus leading them to take the extreme step. To top this, the produce they make is often brought by middle men at low prices which is then sold in the markets at much higher prices.

This mini project aims to help our farmers by eliminating the need of middle men who cheat them by enabling the farmers to sell their produce at a much more profitable rate and at the same time at an economical rate to the consumer.

There are two different interfaces for both the farmer and the consumer. New users can register themselves via separate portals for both farmers and consumers.

Farmers with existing usernames and passwords can login and add products they wish to sell on the platform. Consumers similarly can login and buy the products enlisted by the farmers via the virtual market.

A bill is provided which shows the total amount as well as the products brought by the consumer. Thus, this platform enables the farmers to directly sell their produce to the consumers without being cheated by middle men.

The entire program has been developed in Java and uses the Eclipse IDE for running the java application. The program has been sub divided into various classes for the sake of understanding and uses the various object-oriented programming concepts available in Java.

# ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman of New Horizon Educational Institutions for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha**, Principal, New Horizon College of Engineering, for his constant support and encouragement.

I am grateful to **Dr. Prashanth C.S.R.**, Dean Academics, for his unfailing encouragement and suggestions, given to me in the course of my project work.

I would also like to thank **Dr. B. Rajalakshmi**, Professor and Head, Department of Computer Science and Engineering, for her constant support.

I also express my gratitude to **Ms. Rajitha Nair**, Senior Assistant Professor, Department of Computer Science and Engineering, my project guide, for constantly monitoring the development of the project and setting up precise deadlines. Her valuable suggestions were the motivating factors in completing the work.

**RAHUL MUSALIYATH DINESH**

**USN: 1NH18CS738**

# CONTENTS

<b>ABSTRACT</b>	<b>I</b>
<b>ACKNOWLEDGEMENT</b>	<b>II</b>
<b>LIST OF FIGURES</b>	<b>V</b>
<b>1. INTRODUCTION</b>	
1.1. PROBLEM DEFINITION	<b>1</b>
1.2. OBJECTIVES	<b>1</b>
1.3. METHODOLOGY TO BE FOLLOWED	<b>2</b>
1.4. EXPECTED OUTCOMES	<b>2</b>
1.5. HARDWARE AND SOFTWARE REQUIREMENTS	<b>2</b>
<b>2. OBJECT ORIENTED CONCEPTS</b>	
2.1. CLASSES	<b>3</b>
2.2. OBJECT	<b>4</b>
2.3. INHERITANCE	<b>6</b>
2.4. POLYMORPHISM	<b>12</b>
2.5. ABSTRACT CLASS	<b>15</b>
2.6. MULTI THREADING	<b>16</b>
2.7. I/O FUNCTIONS	<b>19</b>
2.8. JAVA PACKAGES	<b>22</b>
2.9. EXCEPTION HANDLING	<b>23</b>
<b>3. DESIGN</b>	
3.1. DESIGN GOALS	<b>26</b>
3.2. ALGORITHM – PSEUDOCODE	<b>26</b>
<b>4. IMPLEMENTATION</b>	
4.1. USER MANAGERS	<b>36</b>
4.2. USER PROFILES	<b>47</b>

4.3. OTHER CLASSES	58
4.4. EXCEPTION HANDLING	65
<b>5. RESULTS</b>	
5.1. REGISTERING A NEW USER	68
5.2. ADDING VEGETABLES IN FARMER PROFILE	70
5.3. ADDING VEGETABLES INTO CONSUMER'S CART	73
<b>6. CONCLUSION</b>	79
<b>REFERENCES</b>	80

# LIST OF FIGURES

<b><u>Figure No</u></b>	<b><u>Figure Description</u></b>	<b><u>Page No</u></b>
1	Single Inheritance	8
2	Multilevel Inheritance	10
3	Hierarchical Inheritance	12
4	Lifecycle of a Thread	19
5	Standard I/O Streams	20
6	Various classes of InputStream	21
7	Various classes of OutputStream	21
8	Package hierarchy	22
9	Exception hierarchy	24

## **CHAPTER 1**

# **INTRODUCTION**

### **1.1 PROBLEM DEFINITION**

India is one of the largest countries in the world in terms of size and population. As of today, we are the 2<sup>nd</sup> largest nation in terms of population, only second to China. Due to our growing population, there is an increase in the demand for food supplies like rice, wheat etc. which are an essential part of almost every Indian meal.

This growing demand for food supplies increases the pressure on the agriculture sector and our farmers who struggle day and night to feed our country and at the same time earn a livelihood out of it.

The produce made by farmers are often sold at very low prices to the middlemen who then sell this at a higher rate in the markets and make profits out of the farmers' misery. The farmers, many of whom have dues and loans mostly end up in the vicious cycle of debt which lead many to take the extreme step of suicide.

Thus, farmers are cheated of their rightful share of the profits due to the involvement of middlemen. This also affects the consumers who buy the produce as they often have to buy it at much higher prices, again due to the involvement of middlemen.

### **1.2 OBJECTIVES**

This mini-project aims to provide a platform to farmers to sell their produce at a profitable rate by completely eliminating the need of middle men by directly selling to the consumers. It also provides a platform for consumers to buy the produce and generate a bill for the items they have brought from the farmers.

It provides two separate portals for both the farmer and the consumer to register themselves and to login in case they have already registered.



## 1.3 METHODOLOGY TO BE FOLLOWED

The mini-project is completely based on the high-level language, Java and uses the various object-oriented concepts like classes, polymorphism, encapsulation, inheritance etc. to provide a simple and easy to understand platform for both type of users, i.e. farmers and consumers.

## 1.4 EXPECTED OUTCOMES

The mini-project aims to provide a simple and secure platform to help farmers sell their produce directly to the consumer without involving middlemen. The mini-project aims to build a platform where both kind of users, farmers and consumers will be able to login or register themselves and perform their desired tasks. It aims to help farmers give a platform to sell their goods and a platform to the consumers to buy the goods from the farmers.

## 1.5 HARDWARE AND SOFTWARE REQUIREMENTS

- Processor : Intel 386 or higher
- RAM : 4 MB or higher
- Hard Disk : 25 MB or higher
- Input device : Standard Keyboard and Mouse
- Output device : VGA and High-Resolution Monitor or higher
- Operating system : Microsoft DOS, Microsoft Windows 3.1 or later
- JDK version : 1.8
- IDE : Eclipse Neon

## CHAPTER 2

# OBJECT ORIENTED CONCEPTS

## 2.1 CLASSES

Classes are the most important component of Java. We know that Java is an object oriented programming language and hence in Java, everything is associated with both classes and objects. Classes contain members like methods and variables. Thus we can define a class as a blueprint from which objects are created.

For example, let us suppose there is a class called Dog. Now, we know that a dog has attributes such as breed, age and colour and methods such as barking and sleeping. Thus, the class Dog defines a common blueprint for all kinds of dogs.

This can be represented by the following code snippet which illustrates how the above example would be implemented in Java.

```
public class Dog
{
    String breed;
    int age;
    String colour;

    void barking()
    {

    }

    void sleeping()
    {

    }
}
```

A class can contain several members like

- Static variables and methods
- Local variables
- Instance variables and methods

Local variables are those variables of a class which are defined inside blocks, methods or constructors. These are declared and initialized when the control enters the block, method or constructor and destroyed when the control exits the block.

Instance variables are those variables which are defined outside blocks, methods and constructors. They are created in the heap memory when objects (i.e. instances of a class) are created and are destroyed when the object is destroyed.

Static variables or class variables are those variables which are declared using the keyword 'static' but outside a block, method or a constructor. When static variables are created, only one copy of it is created in the static memory for the entire block irrespective of how many objects or instances are created from the class and are destroyed when the execution of the program is over.

## 2.2 OBJECT

As mentioned before, a class basically provides the blueprint for instances created from it which are known as objects. Hence, we can say that objects are created from classes. In Java, the keyword 'new' is used for creating new objects. There are three steps involved in creating an object from a class:

- Declaration – A variable declaration with a variable name with an object type.
- Instantiation – The 'new' keyword is used to create the object.
- Initialization – The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Let us see how the example of the Dog class can be used to understand how objects are created in Java.

```
public class Dog
{
    public Dog (String name)
    {
        // This constructor has one parameter, name.
        System.out.println("Dog's name is :" + name );
    }

    public static void main (String args[])
    {
        // Following statement would create an object myDog
        Dog myDog = new Dog ("Spike");
    }
}
```

The above example shows an object of the class Dog being created by the keyword 'new' and being referenced by the reference variable myDog. At the time of object creation, we can also see that the parameterized constructor of the class Dog being invoked by passing a string as parameter.

We can use the reference variables to invoke or use the instance methods and variables defined in the class. Let us see another example:

```
public class Dog
{
    String colour = "black";
    public void Barking ()
    {
        System.out.println("Dog is barking!");
    }

    public static void main (String args[])
    {
```

```
        Dog myDog = new Dog ();
        System.out.println("Dog's colour is" + myDog.colour);
        myDog.Barking();
    }
}
```

Here, the reference variable myDog is used to point to the new object being created by the 'new' keyword and for accessing the instance variable 'colour' and instance method 'Barking'.

## 2.3 INHERITANCE

Inheritance is one of the key features of an object-oriented language like Java. It is an OOPS concept that allows one class to inherit the features (i.e. the members like instance variables and methods) from another class. There are few important terms used when we deal with inheritance like:

- *Parent Class*: It is the class from whom the features are being inherited. It is also known as the base class or super class.
- *Child Class*: It is the class that inherits the features of the parent class. It is also known as derived class or sub class.
- *Reusability*: Inheritance allows reusability which basically helps recycle code and reduces the typing load. For example, when we want to create a new class but there is already another class which contains some of the methods and variables, we want to use then we can derive the new class from the already existing class.

The basic syntax for inheritance in Java is:

```
class ChildClass extends ParentClass
{
    //methods and fields
}
```

The keyword 'extends' is used for implementing inheritance in Java. Let us understand this better using an example. Let there be a parent class named Animal and a child class named Dog which inherits from Animal.

```
class Animal
{
    public void eat ()
    {
        System.out.println ("Animal is eating.");
    }
    public void sleep ()
    {
        System.out.println("Animal is sleeping.");
    }
}
Class Dog extends Animal
{
    public void bark ()
    {
        System.out.println("Dog is barking.");
    }
    Public static void main (String args[])
    {
        Dog myDog = new Dog ();
        myDog.eat();
        myDog.sleep();
        myDog.bark();
    }
}
```

In the above code snippet, the class Dog is inheriting from the parent class Animal. Therefore, the child class Dog inherits all the members (only public) of the parent class Animal. Thus, when we create an object of Dog, the reference variable pointing to the object can be used to invoke all the methods of the parent class as well as those already in child class Dog.

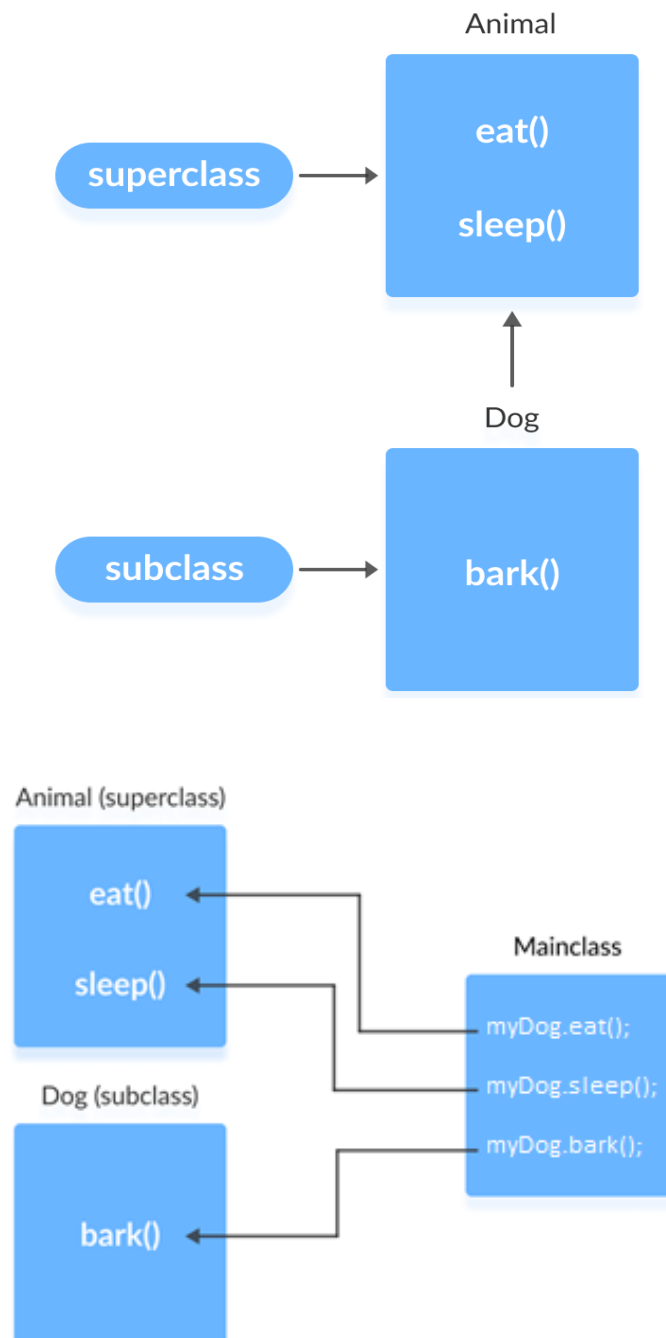


Figure 1: Single Inheritance

There are 3 types of inheritance in Java:

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

In Single inheritance, the features and methods of the parent class are inherited by a single child class. The example of the Animal and Dog classes showed single inheritance where the Dog class was deriving or inheriting features from the Animal class.

In multilevel inheritance, the sub class inherits the features of the superclass and simultaneously it acts as a superclass for another sub class. Hence there is a chain of inheritance. For example,

```
class Animal
{
    void eat()
    {
        System.out.println("Currently Eating");
    }
}
class Dog extends Animal
{
    void bark()
    {
        System.out.println("Currently Barking");
    }
}
class BabyDog extends Dog
{
    void weep()
    {
        System.out.println("Currently Weeping");
    }
}
class TestInheritance
{
    public static void main(String args[])
    {
        BabyDog d =new BabyDog();
    }
}
```



```
        d.weep();  
        d.bark();  
        d.eat();  
    }  
}
```

In the above example, the class BabyDog is inheriting from the class Dog which itself is inheriting from the class Animal. Thus, there are various levels of inheritance and hence this type of inheritance is called multilevel inheritance.

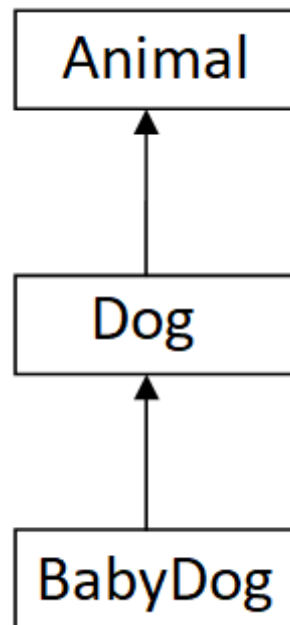


Figure 2: Multilevel Inheritance

In hierarchical inheritance, two or more classes inherit from the same parent class in a hierarchical manner. Let us look at an example,

```
class Animal  
{  
    void eat()  
    {  
        System.out.println("Currently Eating");  
    }  
}
```

```
class Dog extends Animal
{
    void bark()
    {
        System.out.println("Currently Barking");
    }
}
class Cat extends Animal
{
    void meow()
    {
        System.out.println("Currently Meowing");
    }
}
class TestInheritance
{
    public static void main(String args[])
    {
        Dog d = new Dog();
        Cat c = new Cat();
        d.bark();
        d.eat();
        c.meow();
        c.eat();
    }
}
```

In the above example, the class Dog as well as the class Cat both are inheriting from the class Animal. Thus both classes Cat and Dog will be able access the eat method and their respective sound functions. Thus, the reference variable c of type Cat will not be able to call bark as in c.bark() because the method bark is not defined for the class Cat. Similarly, the reference variable d of type Dog will not be able to call the meow as in d.meow() because the method meow is not defined for the class Dog.

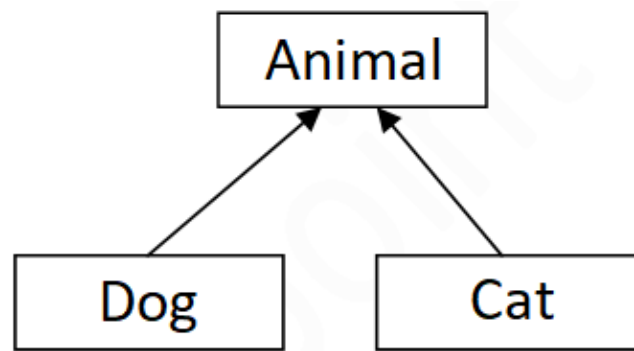


Figure 3: Hierarchical Inheritance

## 2.4 POLYMORPHISM

Polymorphism is another key feature of object oriented programming languages like Java. The name comes from two Greek words, 'poly' which means 'many' and 'morphs' which means 'forms'. Hence polymorphism can be simply be put as 'many forms'.

Polymorphism in Java are of two types:

- Compile-time Polymorphism
- Run-time Polymorphism

Polymorphism in Java is implemented using two concepts:

- Method overloading
- Method overriding

A real-life example of polymorphism: A person at a time can have different roles. That is a man at a time can be a father, a son, an uncle, a husband or even an employee. Hence a person possesses different roles or forms in different situations. This is known as polymorphism.

A Java implementation of polymorphism can be given like this: Let there be a parent class called 'Animal' which has a method called 'SoundMadebyAnimal'. Classes inheriting from the parent class, i.e. the child classes of 'Animal' like Bird, Dog, Cat and

Horse may have their own implementation of this method, i.e. a 'Bird' chirps, a 'Dog' barks, a 'Cat' meows and a 'Horse' neighs.

```
class Animal {
    public void SoundMadebyAnimal() {
        System.out.println("Generic sound");
    }
}

class Cat extends Animal {
    public void SoundMadebyAnimal() {
        System.out.println("The Cat says: meow meow");
    }
}

class Horse extends Animal {
    public void SoundMadebyAnimal() {
        System.out.println("The Horse says: neigh neigh");
    }
}
```

As discussed earlier, there are two types of polymorphism. The first type is compile time polymorphism also known as static polymorphism. Polymorphism of this type is achieved through method overloading and operator overloading.

Method overloading happens when there exists multiple methods with the same name but having different parameters. Example of method overloading:

```
class MultiplyDemo {
    // Method with 2 int type parameter
    static int Multiply(int a, int b)
    {
        return a * b;
    }
}
```

```
// Method with the same name but 2 double type parameters
static double Multiply(double a, double b)
{
    return a * b;
}
}
class Main {
    public static void main(String[] args)
    {
        System.out.println(MultiplyDemo.Multiply(2, 4));
        System.out.println(MultiplyDemo.Multiply(5.5, 6.3));
    }
}
```

Operators can also be overloaded in Java. For example. The addition symbol '+' can be used to concatenate two string though we know that this operator is used to add two operands. Hence an operator like '+' when used between two integer operands, adds them and concatenated them when used between string operands.

The second type of polymorphism is called runtime polymorphism. This is mainly achieved through method overriding. Method overriding is basically a derived class or a child class defining a method which belongs to the parent class. When this happens, the parent class method is said to be overwritten. Example:

```
class Parent {
    void Print()
    {
        System.out.println("This is Parent class.");
    }
}
class ChildClass1 extends Parent {
    void Print()
    {
        System.out.println("This is ChildClass1.");
    }
}
```

```
    }  
}  
class ChildClass2 extends Parent {  
    void Print()  
    {  
        System.out.println("This is ChildClass2.");  
    }  
}  
class PolyDemo {  
    public static void main(String[] args)  
    {  
        Parent a;  
        a = new ChildClass1();  
        a.Print();  
        a = new ChildClass2();  
        a.Print();  
    }  
}
```

Here in the above example we can see that the overridden method 'Print' is called through a reference variable of the 'Parent' class. The JVM decides which method to call based on the object being referred by the reference variable.

## 2.5 ABSTRACT CLASS

Before understanding abstract classes, one needs to know what abstraction is. Abstraction is an object-oriented programming concept which deals with hiding the implementation details from the user and only showing the functionality part.

In Java, abstraction is implemented using two concepts; abstract classes and interfaces. A class which is declared using the 'abstract' keyword is known as an abstract class. It may have abstract (non-concrete) methods or non-abstract (concrete) methods. While abstract classes achieve abstraction between 0 – 100%, interfaces achieve 100% abstraction.

Abstract classes need to be extended and its abstract methods should be implemented in the extending class. Abstract classes can never be instantiated as such objects would be useless, because an abstract class is not fully defined. Example:

```
abstract class A {
    abstract void abstractM(); //abstract method
    // concrete method
    void concreteM() {
        System.out.println("This is a concrete method.");
    }
}
class B extends A {
    void abstractM() {
        System.out.println("B's implementation of abstractM.");
    }
}
class AbstractDemo {
    public static void main(String args[]) {
        B b = new B();
        b.abstractM();
        b.concreteM();
    }
}
```

## 2.6 MULTI THREADING

Multithreading is a feature in Java that allows simultaneous execution of two or more parts of a program. This is done for maximum utilization of the CPU. Here each part of the program is called a thread. Hence threads can also be seen as light weight processes within a process, i.e. the program in execution.

Multi-threading is implanted in two ways:

- Extending Thread class
- Implementing runnable interface

The first way a thread is created is by creating a class that extends the 'Thread' class. This class should override the 'run' method present in the Thread class because when we create an object of this class and call the 'start' method, the 'start' method implicitly calls the 'run' method. Thus a thread begins its life inside the 'run' method. Example:

```
class MTDemo extends Thread
{
    public void run()
    {
        try
        {
            // Displaying the thread that is running
            System.out.println ("Thread " +
                                Thread.currentThread().getId() +
                                " is running");
        }
        catch (Exception e)
        {
            System.out.println ("Exception caught!");
        }
    }
}

public class MTMain
{
    public static void main(String[] args)
    {
        int n = 8;
        for (int i = 0; i < n; i++)
        {
            MTDemo ob = new MTDemo();
            ob.start();
        }
    }
}
```



```
    }  
  }  
}
```

The second method for creating a thread is by implementing the 'Runnable' interface and again overriding the 'run' method. Example:

```
class MTDemo implements Runnable  
{  
    public void run()  
    {  
        try  
        {  
            // Displaying the thread that is running  
            System.out.println ("Thread " +  
                                Thread.currentThread().getId() +  
                                " is running");  
        }  
        catch (Exception e)  
        {  
            System.out.println ("Exception caught!");  
        }  
    }  
}  
  
public class MTMain  
{  
    public static void main(String[] args)  
    {  
        int n = 8;  
        for (int i = 0; i < n; i++)  
        {  
            MTDemo m = new MTDemo();  
            Thread t = new Thread(m);  
            t.start();  
        }  
    }  
}
```

```
        Thread t = new Thread (m);  
        t.start();  
    }  
}
```

A thread goes through various stages in its life cycle. Initially a thread when is created, it is in new born state. It then runs and later dies.

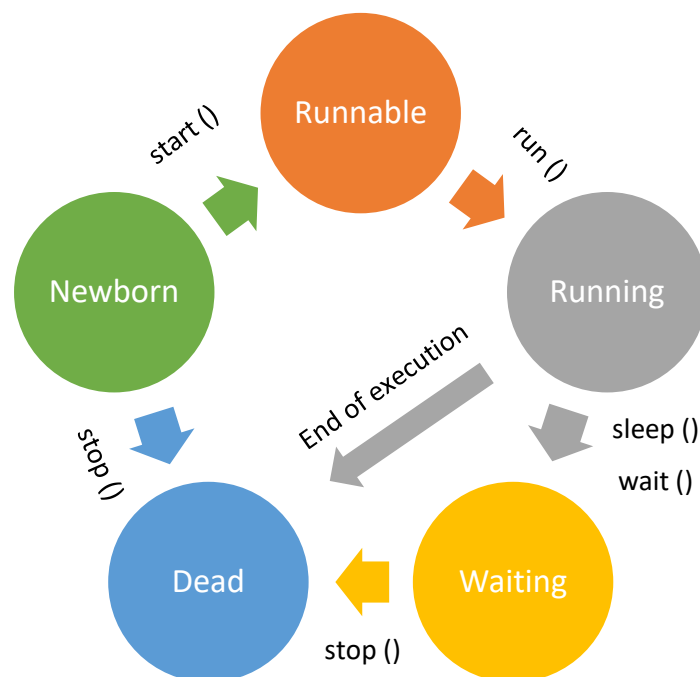


Figure 4: Lifecycle of a Thread

## 2.7 I/O FUNCTIONS

Input and output operations in Java are performed using various streams provided in the I/O package. All kinds of objects, datatypes, characters and files are supported by these streams to execute the I/O operations. There are 3 types of standard I/O streams that Java provides.

- System.in
- System.out
- System.err

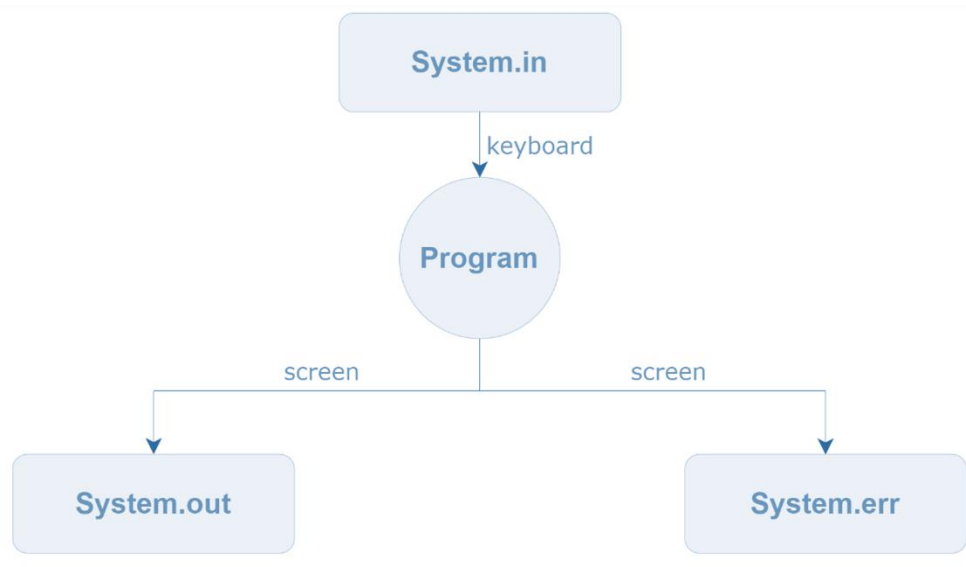


Figure 5: Standard I/O Streams

System.in is the standard stream for inputs. It is mainly used for reading characters from the keyboard or any other input device.

System.out is the standard stream for outputs. It is mainly used to provide the result on an output screen like the monitor or computer screen. It has various print functions that are used to provide various print methods like:

- print ()
- println ()
- printf ()

System.err is the standard stream for all errors. It is used to display all the error data that the program may throw during execution, on any output device like a computer screen or monitor. The stream also uses the above three methods to display the errors.

Streams can also be divided into 2 primary classes based on the type of operation.

- Input Stream
- Output Stream

Data from any source arrays or any peripheral devices can be read using classes of `InputStream`. Example, `DataInputStream`, `FileInputStream` etc.

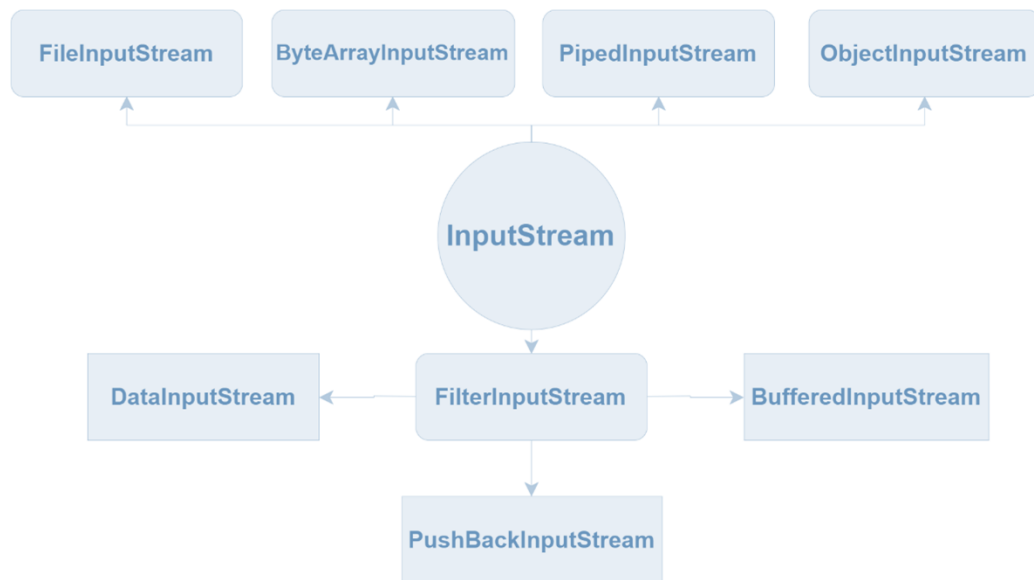


Figure 6: Various classes of `InputStream`

If data is to be written onto arrays, files or any peripheral devices then we use classes of `OutputStream`. Example, `BufferedOutputStream`, `DataOutputStream` etc.

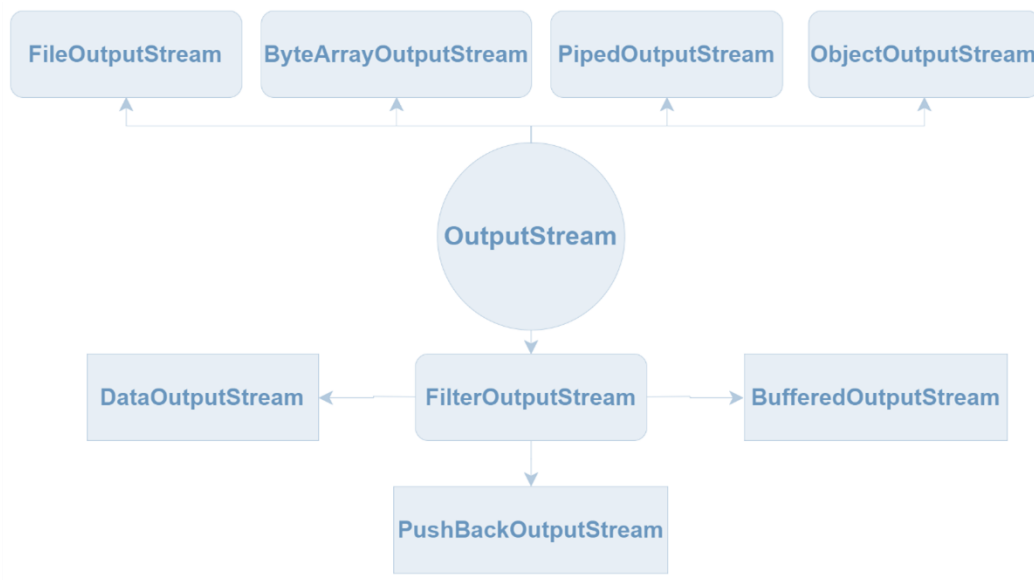


Figure 7: Various classes of `OutputStream`

## 2.8 PACKAGES

Encapsulation is achieved in Java using classes and packages. Classes encapsulate methods and data members while packages encapsulate classes, interfaces and sub packages. Packages have a number of uses:

- Packages help prevent naming conflicts. For example there can exist two separate classes with the same name 'Employee' in two separate packages nhce.cse.employee and nhce.isc.employee.
- It helps in organizing classes in an efficient manner and makes searching, locating and using classes easier.
- Packages also provide an other layer of access protection. The access layers 'protected' and 'default' are used for package level access control. Methods, constructors and data members which are declared protected in a parent class can be accessed only by the child classes in other package or any class within the same package of the protected members' class. A default member (i.e. without any access specifier) is accessible by classes within the same package.

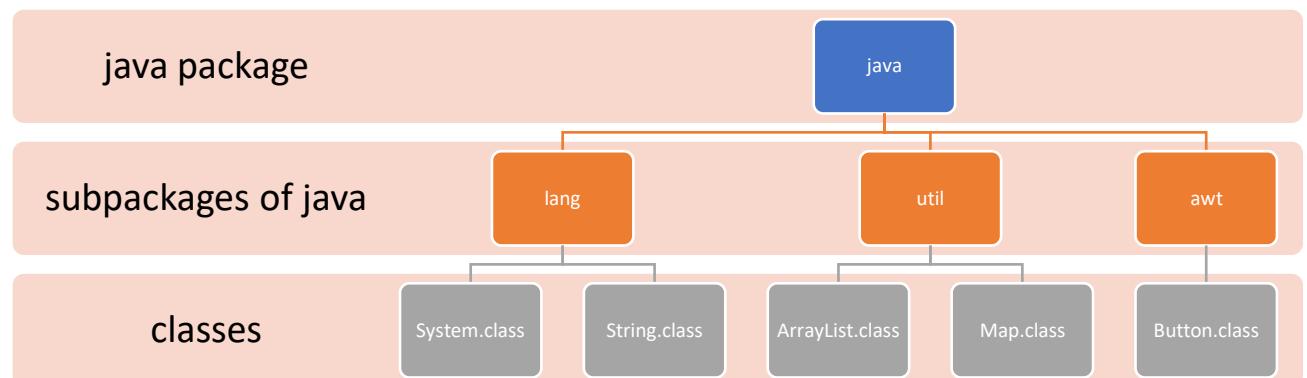


Figure 8: Package hierarchy

Packages can be both built in and user defined. There are many built-in packages such as java, lang, javax, awt, net, swing, util, io, sql etc. The keyword 'import' is used to import an entire package or a class of a particular package. User defined packages can be created using the keyword, 'package'.

## 2.9 EXCEPTION HANDLING

Before understanding Java's exception handling feature, one should know what are exceptions. An exception can be described as an unexpected or unwanted event which mainly occurs during the program execution. An exception disturbs the regular flow of the program instructions and terminates execution. Simply put, exceptions are abnormal conditions.

An exception may arise due to various different reasons. Some scenarios where exceptions in Java occur are:

- When invalid data is entered by the user.
- When a file that needs to be opened is not found.
- When network connection is lost in the middle of communications
- When JVM has run out of memory.

Exceptions in Java are of three types:

- Checked exceptions
- Unchecked exceptions
- Errors

Checked exceptions are also known as compile-time exceptions because these exceptions are checked or notified at compilation time. These exceptions are to be handled by the programmer and can not be ignored. Example, suppose we need to read data from a file using the `FileReader` class. If the file specified in its constructor does not exist then an exception called '`FileNotFoundException`' is thrown by the JVM and the compiler thus urges the programmer to handle the exception.

Unchecked exceptions are thrown when the program is in execution and hence are also called runtime exceptions. '`ArithmeticException`' caused by division by zero is one of the most common types of unchecked exception. Another example is the

'`ArrayIndexOutOfBoundsException`' which occurs when we try to use an array index which is not declared for a particular array.

Exceptions in Java are subclasses of the `java.lang.Exception` class. The exception class is in turn a derived or child class of the `Throwable` class.

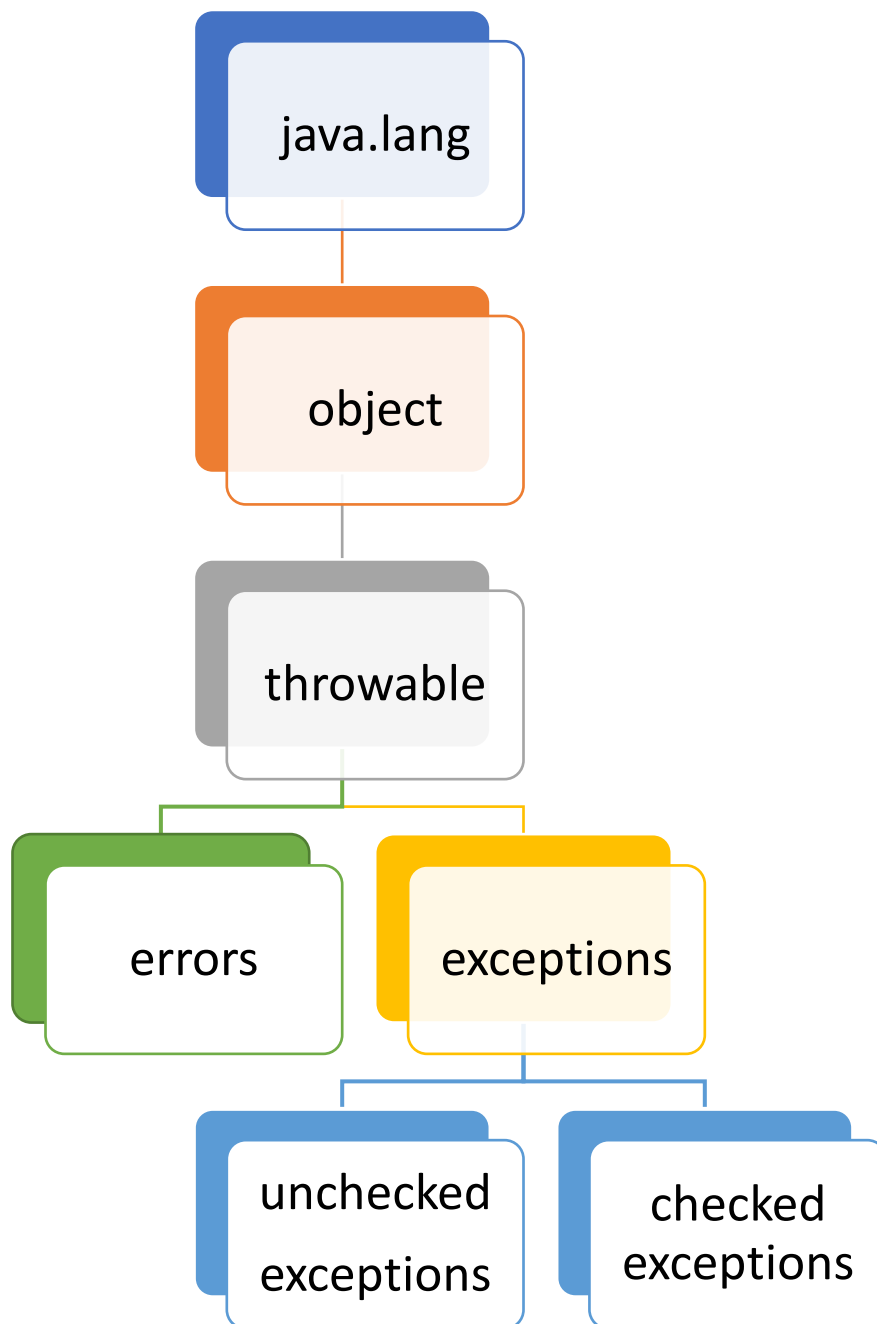


Figure 9: Exception Hierarchy

Exceptions in Java are handled primarily using ‘try – catch’ clauses. A block of code which may generate an exception is placed inside the try block. When an exception occurs, it is handled by the associated catch block. It is mandatory that every try block should be followed by a catch block or a finally block.

```
class EHDemo {  
    public static void main(String args[]) {  
        int num1, num2;  
        try {  
            num1 = 0;  
            num2 = 62 / num1;  
            System.out.println(num2);  
            System.out.println("End of try block.");  
        }  
        catch (ArithmeticException e) {  
            System.out.println("You should not divide a number by zero");  
        }  
        catch (Exception e) {  
            System.out.println("Exception occurred");  
        }  
        System.out.println("Outside try-catch block in Java.");  
    }  
}
```

Control is passed to the corresponding catch block when an exception arises inside a try block. Multiple catch blocks can be associated with a try block. One should ensure that the generic exception handler, i.e. the parent class exception handler should be placed last.

Though the parent class exception handler or generic exception handler can handle all the exceptions, but if you place it before other catch blocks then the generic handler message will be displayed instead of a specific exception handler’s message. One should always try to ensure that the user is provided a specific message rather than a general message.



## CHAPTER 3

### DESIGN

#### 3.1 DESIGN GOALS

This mini project has ensured that the user has an interactive and explorable environment. The interface is user friendly, simple to understand and has tried to ensure that there are no bugs.

I have tried to ensure that the code used is simple yet logical without compromising on the final output.

Various object-oriented concepts have been used across the mini project so that the number of lines in the program is reduced and also for sake of simplicity and reading.

#### 3.2 ALGORITHM – PSEUDOCODE

Several classes, interfaces, abstract classes have been used in the application along with Java features like constructors, constructor overloading, exception handling among others.

The Java application I developed can be broadly classified into two sections:

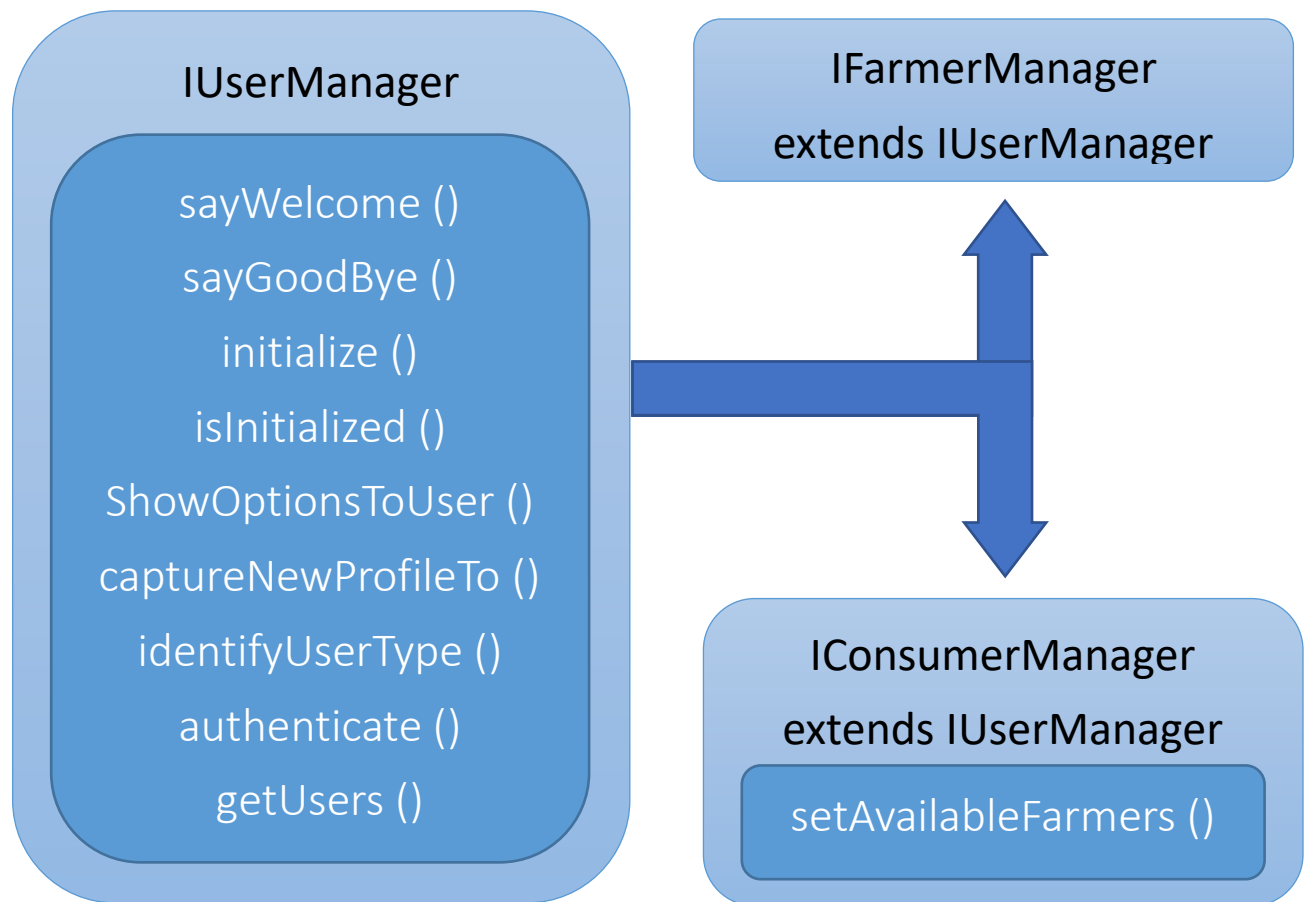
- User Managers
- User Profiles

The manager classes and interfaces help organizing a group of User Profiles and perform appropriate functions. The User Profiles are used to perform operations within a user profile.

Interfaces have been used extensively throughout the application so as to achieve the object-oriented concept of abstraction. Inheritance is used between various abstract and concrete classes to reduce the typing load and for easier understanding. Various exceptions that could have occurred have been handled appropriately using Java's exception handling mechanism.

## User Managers

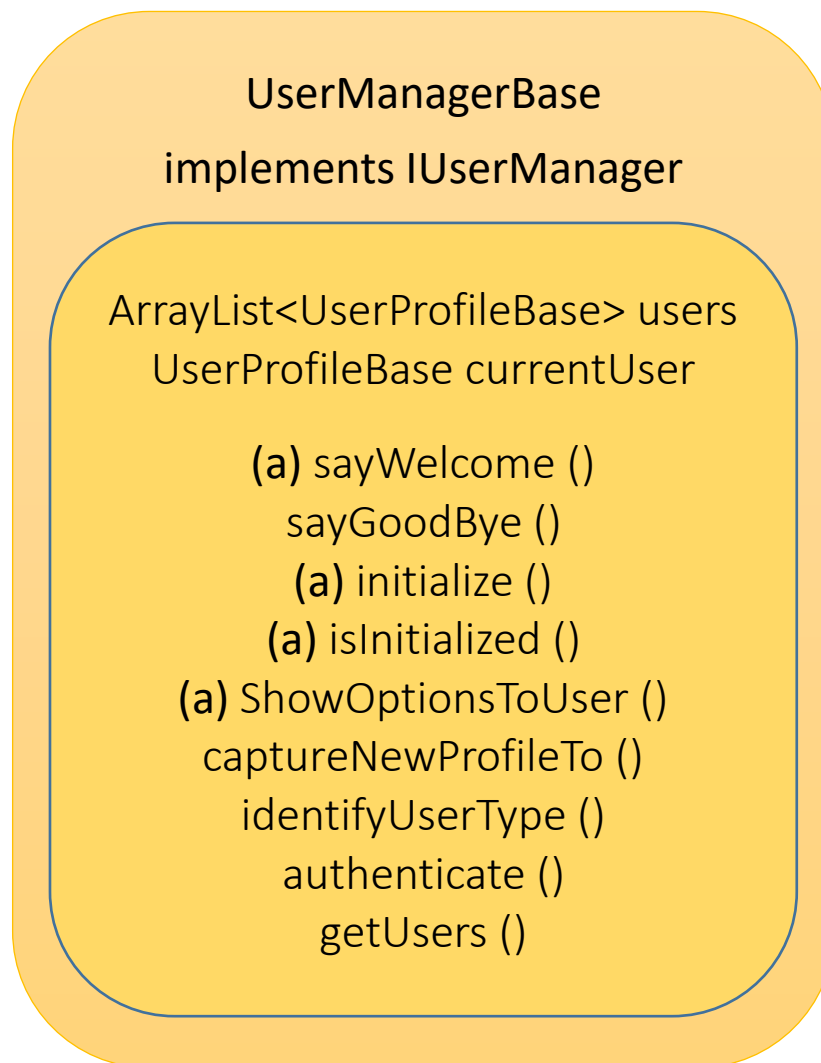
- **Interfaces**



The IUserManager is an interface that has several non-concrete (undefined) methods which are used for non-profile specific operations like

- Displaying greeting messages like 'Welcome' and 'Goodbye'
- Instantiate a new user profile
- Showing various options available to user profiles
- Identifying if a user is a registered user or a new user
- Capturing new profiles for non-registered users
- Authenticating credentials of registered users
- Returning list of users to appropriate methods

- **Abstract classes**



The **UserManagerBase** is an abstract base class that implements few methods of the **IUserManager** interface. The methods prefixed with **(a)** are abstract methods which still have not been implemented in this abstract class. An array list of type **UserProfileBase** is declared as a data member along with a reference variable of type **UserProfileBase**.

The methods responsible for capturing a new profile for non-registered users, identifying whether a user is a registered user or a non-registered user, authenticating whether the entered credentials are that of a registered user and returning the list of users are implemented or defined in this abstract class.

- **Concrete classes**

**ConsumerManager**  
extends UserManagerBase  
implements IConsumerManager

boolean initialized

sayWelcome ()  
initialize ()  
isInitialized ()  
showOptionsToUser ()  
setAvailableFarmers ()

**FarmerManager**  
extends UserManagerBase  
implements IFarmerManager

boolean initialized

sayWelcome ()  
initialize ()  
isInitialized ()  
showOptionsToUser ()

The concrete classes ConsumerManager and FarmerManager defines or implements the abstract methods not defined in the super / parent abstract class UserManagerBase.

Here, the object-oriented programming concept of polymorphism is seen when two child classes, ConsumerManager and FarmerManager are derived from the abstract parent base class. Thus, those abstract methods in the parent class are now having different implementations for both type of users, farmers and consumers.

## User Profiles

- **Interfaces**

### IConsumerProfile

```
addItemToCart ()  
buyItemsFrom ()  
viewOrders ()  
setAvailableFarmers ()  
getAvailableFarmers ()  
displayAvailableItems ()  
payMoney ()
```

### IFarmerProfile

```
addNewVegetableDetails ()  
displayVegetables ()  
sellVegetableTo ()  
viewSales ()  
receiveMoney ()
```

The interface IConsumerProfile has abstract methods required for Consumer Profile specific operations like adding items to cart, buying items from the farmer, viewing order history, fetching the list of available farmers and their products, adding funds to account and paying the farmer the bill amount. The IFarmerProfile interface provides abstract methods for Farmer Specific operations like adding vegetables he wants to sell, viewing all his listed products, selling his vegetables, viewing his sales and receiving money from the consumer.

- **Abstract classes**

### UserProfileBase

```
static long currentId
String name
String username
String password
long id
double moneyBalance

(c) UserProfileBase ()
(a) prepareUser ()
    getName ()
    getUserName ()
    getName ()
    getPassword ()
    getNextId ()
    setName ()
    setUserName ()
    setPassword ()
    getNextId ()
```

The abstract base class `UserProfileBase` is used for operations pertaining to all kinds of profiles. It stores and returns a number of user profile credentials like user id, name, username and password.

- **Concrete classes**

**ConsumerProfile**  
extends `UserProfileBase`  
implements `IConsumerProfile`

`ArrayList<IFarmerProfile> farmers`  
`ArrayList<Order> orders`  
`boolean isFundInitialized`

`prepareUser ()`  
`setAvailableFarmers ()`  
`getAvailableFarmers ()`  
`displayAvailableItems ()`  
`payMoney ()`  
`buyItemsFrom ()`  
`addItemsToCart ()`  
`viewOrders ()`

The `ConsumerProfile` concrete class extends the `UserProfileBase` abstract class and implements the `IConsumerProfile` interface. It has an array list of type `IFarmerProfile` to fetch the list of available farmers. It also has an array list of type `Order` to view all orders the consumer has made. It also implements the methods of the `IConsumerProfile` interface by defining methods for paying the farmer the bill amount, adding funds to the consumer's account among others.

**FarmerProfile**  
extends UserProfileBase  
implements IFarmerProfile

```
ArrayList<Vegetable> vegetables  
ArrayList<VegetableSaleInfo> salesInfo  
prepareUser ()  
addNewVegetableDetails ()  
displayVegetables ()  
sellVegetableTo ()  
receiveMoney ()  
view Sales ()
```

The FarmerProfile concrete class extends the UserProfileBase abstract class and implements the IFarmerProfile interface. It has an array list of type Vegetable to add the farmer's goods. It also has an array list of type VegetableSaleInfo to receive the details about the sales the farmer has made. It also has methods to display the farmer's product list, sell his vegetables to the consumer, receive money and view his sales.

## Other Classes

Various other classes have been used that provide the implementation for aspects common to both consumer and farmer like:

- Holding information related to a vegetable
- Holding information related to an order
- Holding information related to an item to be added to the cart
- Holding information about the sold vegetables
- Handling exceptions that arise when the user enters incorrect values for input using Java's exception handling mechanism



### Item

Vegetable vegetable ()  
double quantity ()

(c) Item ()  
getVegetable ()  
getQuantity ()  
Print ()

### Vegetable

static long currentId  
String name  
long id  
double pricePerKilo  
double stockInKilos

(c) Vegetable ()  
(c) Vegetable (long id)  
getName ()  
getId ()  
getPricePerKilo ()  
getStockInKilos ()  
setName ()  
setPricePerKilos ()  
setStockInKilos ()  
getNextId ()

(c) -> constructor

### Order

static long currentId  
ArrayList<Item> items  
long id

(c) Order ()  
getTotal ()  
Print ()  
AddItem ()  
getItems ()  
getNextOrderId ()

### VegetableSaleInfo

Item soldItem  
IConsumerProfile whoBoughtFromMe

(c) VegetableSaleInfo ()  
getSoldItem ()  
getWhoBoughtFromMe ()  
Print ()

### ScanUtils

getInt ()  
getLong ()  
getDouble ()

## CHAPTER 4

# IMPLEMENTATION

### 4.1 USER MANAGER

- **IUserManager**

```
package Abstractions.Managers;
import java.util.ArrayList;
import Abstractions.Profiles.UserProfileBase;
public interface IUserManager {

    void sayWelcome ();
    void sayGoodBye ();
    void initialize ();
    boolean isInitialized ();
    void showOptionsToUser ();

    void captureNewProfileTo (UserProfileBase newUser);

    UserProfileBase identifyUserType (UserProfileBase newUser);

    UserProfileBase authenticate (String userName, String password);

    ArrayList<UserProfileBase> getUsers ();

}
```

- **IFarmerManager**

```
package Abstractions.Managers;

public interface IFarmerManager extends IUserManager {

}
```

- **IConsumerManager**

```
package Abstractions.Managers;
import java.util.ArrayList;
import Abstractions.Profiles.IFarmerProfile;

public interface IConsumerManager extends IUserManager {

    void setAvailableFarmers (ArrayList<IFarmerProfile> farmers);
}
```

- **UserManagerBase**

```
package Abstractions.Managers;

import java.util.ArrayList;
import java.util.Scanner;

import Abstractions.Profiles.UserProfileBase;

public abstract class UserManagerBase implements IUserManager {

    protected Scanner in = new Scanner (System.in);

    protected ArrayList<UserProfileBase> users
    = new ArrayList<UserProfileBase> ();
    protected UserProfileBase currentUser;

    public abstract void sayWelcome ();
    public abstract void initialize ();
    public abstract boolean isInitialized ();
    public abstract void showOptionsToUser ();
}
```

```
public void captureNewProfileTo (UserProfileBase newUserProfile)
{
    String password, confirmPassword;

    System.out.print ("Enter your name: ");
    newUserProfile.setName (this.in.nextLine());

    System.out.print("Enter your preferred user name: ");
    newUserProfile.setUserName(this.in.nextLine());

    while (true) {
        System.out.print ("Enter a password: ");
        password = this.in.nextLine();

        System.out.print ("Re-enter your password: ");
        confirmPassword = this.in.nextLine();

        if (password.equals(confirmPassword)) {
            newUserProfile.setPassword(password);
            break;
        }
        else {
            System.out.println ("The passwords you
            entered are not same. Please try again.");
        }
    }
}

public void sayGoodBye () {
    System.out.println ("Thank you for using the Agro
    Kart. Good bye!");
}
```

```
public UserProfileBase identifyUserType(UserProfileBase newUser)
{
    int userChoice;
    String userName, password;

    System.out.print ("What do you want to do?\n\n\t
(1) Sign in as existing user \n\t
(2) Register as new user.\n\n
Enter your choice here: ");
    userChoice = this.in.nextInt();

    // Clear the keyboard buffer.
    String buffer = this.in.nextLine();

    switch (userChoice) {
    case 1:
        System.out.print ("Enter your user name:");
        userName = this.in.nextLine();

        System.out.print ("Enter your password:");
        password = this.in.nextLine();

        this.currentUser = this.authenticate(userName,
        password);

        if (this.currentUser == null) {
            System.out.println ("The specified user
            name or password do not match.");
        }

        return (this.currentUser);
    }
```

```
        case 2:
            this.captureNewProfileTo(newUser);
            this.users.add(newUser);
            this.currentUser = newUser;
            return (this.currentUser);

        default:
            this.sayGoodBye();
            return (null);
    }
}

public UserProfileBase authenticate(String userName, String
password) {

    for (int i=0; i<this.users.size();i++){

        UserProfileBase thisProfile = this.users.get(i);

        if (thisProfile.getUserName().equals(userName) &&
thisProfile.getPassword().equals(password)) {
            return thisProfile;
        }
    }

    return null;
}

public ArrayList<UserProfileBase> getUsers () {
    return this.users;
}
}
```

- **FarmerManager**

```
package Aggregates;

import Abstractions.Managers.IFarmerManager;
import Abstractions.Managers.UserManagerBase;
import Abstractions.Profiles.IFarmerProfile;
import Entities.*;
import Utilities.ScanUtils;

public class FarmerManager extends UserManagerBase implements
IFarmerManager {

    private boolean initialized = false;

    @Override
    public void sayWelcome() {
        System.out.println ();
        System.out.println ("Hello, farmer!");
    }

    @Override
    public void initialize () {
        FarmerProfile newUser = new FarmerProfile ();
        super.identifyUserType(newUser);
        this.initialized = true;
    }

    @Override
    public boolean isInitialized() {
        return this.initialized;
    }
}
```



```
@Override
public void showOptionsToUser() {

    boolean userWantsToSignOut = false;

    if (this.initialized == false) {
        System.out.println ("The farmer manager instance
        is not yet initialized!");
        return;
    }

    if (this.currentUser != null) {
        System.out.println ();
        System.out.println
        ("=====");
        System.out.println ("Welcome, farmer " +
        this.currentUser.getName() + "!");
        System.out.println
        ("=====");

        while (true) {
            int userChoice;

            System.out.print ("What do you want to
            do?\n"+ "\n\t(1) Add vegetables that you
            sell"+ "\n\t(2) List your vegetables" +
            "\n\t(3) View Sales" + "\n\t(4) Sign
            out.\n\nEnter your choice here: ");

            userChoice = ScanUtils.getInt();

            IFarmerProfile farmer = (FarmerProfile)
            super.currentUser;
```

```
        switch (userChoice) {
        case 1:
            farmer.addNewVegetableDetails();
            break;

        case 2:
            farmer.displayVegetables();
            break;

        case 3:
            farmer.viewSales ();
            break;

        case 4:
            userWantsToSignOut = true;
            break;

        default:
            System.out.println ("Returning to the
            previous menu ...");
            break;
        }

        if (userWantsToSignOut) {
            break;
        }
    }
}
}
```

- **ConsumerManager**

```
package Aggregates;

import java.util.ArrayList;
import java.util.Scanner;

import Abstractions.Managers.IConsumerManager;
import Abstractions.Managers.UserManagerBase;
import Abstractions.Profiles.IConsumerProfile;
import Abstractions.Profiles.IFarmerProfile;
import Entities.*;
import Utilities.ScanUtils;

public class ConsumerManager extends UserManagerBase implements
IConsumerManager {
    private boolean initialized = false;
    private Scanner in = new Scanner (System.in);

    public void sayWelcome() {
        System.out.println ();
        System.out.println ("Hello, consumer!");
    }
    public void initialize () {
        IConsumerProfile consumer = new ConsumerProfile ();
        UserProfileBase normalUser = (UserProfileBase)
        consumer;
        consumer = (IConsumerProfile)
        super.identifyUserType(normalUser);

        if (consumer != null) {
            ((UserProfileBase) consumer).prepareUser();
        }

        this.initialized = true;
    }
}
```

```
public boolean isInitialized() {
    return this.initialized;
}

public void setAvailableFarmers(ArrayList<IFarmerProfile>
farmers) {
    IConsumerProfile thisConsumer
    = (IConsumerProfile) super.currentUser;
    thisConsumer.setAvailableFarmers(farmers);
}

@Override
public void showOptionsToUser() {
    boolean userWantsToSignOut = false;

    if (this.initialized == false) {
        System.out.println ("The consumer manager
instance is not yet initialized!");
        return;
    }

    if (this.currentUser != null) {
        System.out.println ();
        System.out.println
        ("=====");
        System.out.println ("Welcome, consumer " +
this.currentUser.getName() + "!");
        System.out.println
        ("=====");
        while (true) {
            int userChoice;
```

```
System.out.print ("What do you want to do?\n"
+ "\n\t(1) Display all farmers and their products" +
"\n\t(2) Add items to cart" + "\n\t(3) View orders"
+ "\n\t(4) Sign out.\n\nEnter your choice here: ");
        userChoice = this.in.nextInt();

        IConsumerProfile thisConsumer =
        (ConsumerProfile) super.currentUser;

        switch (userChoice) {
        case 1:
        thisConsumer.displayAvailableItems();
            break;

        case 2:
            thisConsumer.addItemToCart ();
            break;

        case 3:
            thisConsumer.viewOrders ();
            break;

        case 4:
            userWantsToSignOut = true;
            break;

        default:
            System.out.println ("Returning
            to the previous menu ...");
            break;
        }
    }
```

```
        if (userWantsToSignOut) {  
            break;  
        }  
    }  
}
```

## 4.2 USER PROFILES

- **IConsumerProfile**

```
package Abstractions.Profiles;  
  
import java.util.ArrayList;  
  
import Entities.Item;  
  
public interface IConsumerProfile {  
  
    void addItemToCart ();  
    void buyItemsFrom(ArrayList<Item> items);  
    void viewOrders ();  
  
    void setAvailableFarmers (ArrayList<IFarmerProfile> farmers);  
    ArrayList<IFarmerProfile> getAvailableFarmers ();  
    void displayAvailableItems ();  
  
    double payMoney (double amount, String remarks);  
}
```

- **IFarmerProfile**

```
package Abstractions.Profiles;

import Entities.Item;

public interface IFarmerProfile {

    void addNewVegetableDetails ();
    void displayVegetables ();
    boolean sellVegetableTo (IConsumerProfile consumer, Item item);

    void viewSales ();
    void receiveMoney (double money);
}
```

- **UserProfileBase**

```
package Abstractions.Profiles;

public abstract class UserProfileBase {
    private static long currentId = 0;

    protected String name;
    protected String userName;
    protected String password;
    protected long id;
    protected double moneyBalance = 0;

    protected UserProfileBase () {
        this.id = UserProfileBase.getNextId();
    }
}
```

```
    public String getUserName () {
        return this.userName;
    }

    public String getPassword () {
        return this.password;
    }

    public void setName (String name) {
        this.name = name;
    }

    public void setUserName (String userName) {
        this.userName = userName;
    }

    public void setPassword (String password) {
        this.password = password;
    }

    private static long getNextId () {
        return UserProfileBase.currentId++;
    }
}
```

- **ConsumerProfile**

```
package Entities;

import java.util.Scanner;
import java.util.ArrayList;
```



```
import Abstractions.Profiles.IConsumerProfile;
import Abstractions.Profiles.IFarmerProfile;
import Abstractions.Profiles.UserProfileBase;
import Utilities.ScanUtils;

public class ConsumerProfile extends UserProfileBase implements
IConsumerProfile {

    private Scanner in = new Scanner (System.in);
    private ArrayList<IFarmerProfile> farmers;
    private ArrayList<Order> orders = new ArrayList<Order> ();
    private boolean isFundInitialized = false;

    public void prepareUser () {
        if (this.isFundInitialized == false) {
            System.out.print ("How much fund do you want to
add? Rs. ");
            double amount = ScanUtils.getDouble();

            if (amount > 0) {
                super.moneyBalance += amount;
                this.isFundInitialized = true;
            }
            else {
                System.out.println("Money should be more
than zero to add to funds.");
            }
        }
    }
    public void setAvailableFarmers(ArrayList<IFarmerProfile>
farmers) {
        this.farmers = farmers;
    }
    public ArrayList<IFarmerProfile> getAvailableFarmers() {
        return this.farmers;
    }
}
```

```
@Override
public void displayAvailableItems() {
    for (int i = 0; i < this.farmers.size (); i++) {
        this.farmers.get(i).displayVegetables();
    }
}

public double payMoney(double amount, String remarks) {
    if (super.moneyBalance >= amount) {
        super.moneyBalance -= amount;
        return amount;
    }
    else {
        System.out.println("\t==> Payment amount for "
+ remarks + " is Rs. " + amount + ", however,
you have only left Rs. " + super.moneyBalance +
" in your account. Unable to continue with the
purchase.");
        return 0;
    }
}

public void buyItemsFrom(ArrayList<Item> items) {
    Order order = new Order ();

    for (int i = 0; i < this.farmers.size(); i++) {
        IFarmerProfile thisFarmer = this.farmers.get(i);

        for (int j = 0; j < items.size(); j++) {
            Item oneItem = items.get(j);

            if (thisFarmer.sellVegetableTo(this,
oneItem)) {
                order.AddItem(oneItem);
            }
        }
    }
}
```

```
        }  
    }  
  
    if (order.getItems().size() > 0) {  
        this.orders.add(order);  
  
        order.Print();  
        System.out.println("You have Rs. " +  
            this.moneyBalance + " left in your account.");  
    }  
}  
  
@Override  
public void addItemToCart() {  
    ArrayList<Item> itemsToBuy = new ArrayList<Item> ();  
  
    while (true) {  
        System.out.println();  
        System.out.print ("Specify vegetable code: ");  
        long vegCode = ScanUtils.getLong();  
  
        System.out.print ("Specify the quantity: ");  
        double quantity = ScanUtils.getDouble();  
  
        Vegetable vegToBuy = new Vegetable (vegCode);  
  
        Item itemToBuy = new Item (vegToBuy, quantity);  
        itemsToBuy.add(itemToBuy);  
        System.out.println();  
        System.out.print ("Do you want to add any more  
        vegetables? (Y/N): ");  
        String userResponse = this.in.nextLine();
```

```
        if (! (userResponse.equals("yes") ||
            userResponse.equals("Yes") ||
            userResponse.equals("y") ||
            userResponse.equals("Y"))) {
            break;
        }
    }

    System.out.println("Checking out the items in cart
    ...");

    this.buyItemsFrom(itemsToBuy);
}

@Override
public void viewOrders() {
    for (int i = 0; i < this.orders.size(); i++) {
        Order thisOrder = this.orders.get(i);
        thisOrder.Print();
    }
}
}
```

- **FarmerProfile**

```
package Entities;

import java.util.Scanner;
import java.util.ArrayList;

import Abstractions.Profiles.IConsumerProfile;
import Abstractions.Profiles.IFarmerProfile;
import Abstractions.Profiles.UserProfileBase;
```

```
import Utilities.ScanUtils;

public class FarmerProfile extends UserProfileBase implements
IFarmerProfile {

    // Instance fields.
    private Scanner in = new Scanner (System.in);

    private ArrayList<Vegetable> vegetables
    = new ArrayList<Vegetable> ();

    private ArrayList<VegetableSaleInfo> salesInfo
    = new ArrayList<VegetableSaleInfo> ();

    // Add new vegetables to the inventory.
    public void addNewVegetableDetails () {
        int vegetablesCount;
        String vegetableName;
        double pricePerKilo;
        int stockInKilos;

        System.out.print ("How many types of vegetables are
        you selling? ");
        vegetablesCount = ScanUtils.getInt();
        System.out.println();

        // Get each vegetable's details.
        for (int i = 0; i < vegetablesCount; i++) {
            System.out.print("Enter "+(i == 0 ? "first":(
            i == 1 ? "second":(i == 2 ? "third":((i + 1) %
            10 == 3 && i != 12? String.valueOf(i + 1) + "rd"
            : String.valueOf(i + 1) + "th")))) +
            " vegetable's name: ");
```

```
        vegetableName = this.in.nextLine ();

        System.out.print ("Enter vegetable's price per
        Kilo: Rs. ");
        pricePerKilo = ScanUtils.getDouble();

        System.out.print ("Enter vegetable's stock in kg
        that you have to sell: ");
        stockInKilos = ScanUtils.getInt();

        System.out.println("-----
        -----");

        Vegetable veg = new Vegetable ();
        veg.setName (vegetableName);
        veg.setPricePerKilo (pricePerKilo);
        veg.setStockInKilos(stockInKilos);
        this.vegetables.add (veg);
    }
}

public void displayVegetables () {
    System.out.println ("Farmer's name: " +
    super.getName());
    System.out.println();

    for (int i = 0; i < this.vegetables.size (); i++) {
        Vegetable thisVeg = this.vegetables.get (i);
        System.out.println ("Vegetable Code: " +
        thisVeg.getId() + "\t|\t Vegetable: " +
        thisVeg.getName () + "\t|\t Price per kilo: Rs.
        " + thisVeg.getPricePerKilo () + "\t|\t Stock
        available: " + thisVeg.getStockInKilos() + "
        kg.");
    }
}
```

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

    public boolean sellVegetableTo (IConsumerProfile consumer,
    Item item) {
        for (int i = 0; i < this.vegetables.size(); i++) {
            Vegetable thisVeg = this.vegetables.get(i);
            double stockInKilos = thisVeg.getStockInKilos();
            double quantity = item.getQuantity();

            if (thisVeg.getId() ==
            item.getVegetable().getId()) {
                if (stockInKilos >= item.getQuantity()) {
                    Vegetable itemVeg =
                    item.getVegetable();
                    itemVeg.setName(thisVeg.getName());

                    itemVeg.setPricePerKilo(thisVeg.getPricePerKilo());

                    thisVeg.setStockInKilos(stockInKilos - quantity);
                    String remarks = quantity + " kg. of " +
                    thisVeg.getName();
                    if(this.receiveMoney(consumer.payMoney
                    (thisVeg.getPricePerKilo() * quantity, remarks))) {
                        VegetableSaleInfo saleInfo = new VegetableSaleInfo
                        (item, consumer);
                        this.salesInfo.add(saleInfo);
                        return true; }}
                else {
                    System.out.println ("\t ==> From farmer '" +
                    this.getName() + "': The requested quantity " +
                    quantity + " kg. of '" + thisVeg.getName() + "' is
                    not available.");
                }
            }
        }
    }
}
```

```
                return false;
            }
        }
    }

    return false;
}

@Override
public void receiveMoney(double money) {
    if (money > 0) {
        super.moneyBalance += money;
    }
    else {
        System.out.println("Money should be greater than
        zero to receive.");
    }
}

@Override
public void viewSales() {
    System.out.println("=====");
    System.out.println("Sales Info");
    System.out.println("=====");
    System.out.println("Total money in account: Rs. " +
    this.moneyBalance);
    System.out.println();

    for (int i = 0; i < this.salesInfo.size(); i++) {
        VegetableSaleInfo salesInfo
        = this.salesInfo.get(i);
        saleInfo.Print ();
    }
    System.out.println ();
    System.out.println("=====");
    System.out.println ();
}
}
```



## 4.3 OTHER CLASSES

- **Vegetable**

```
package Entities;

public class Vegetable {
    private static long currentId = 1;
    private String name;
    private long id;
    private double pricePerKilo;
    private double stockInKilos;

    public Vegetable () {
        this.id = Vegetable.getNextId();
    }

    public Vegetable (long id) {
        this.id = id;
    }

    // Getters.
    public String getName () {
        return this.name;
    }

    public long getId () {
        return this.id;
    }

    public double getPricePerKilo () {
        return this.pricePerKilo;
    }
}
```

```
public void setName (String name) {
    this.name = name;
}

public void setPricePerKilo (double pricePerKilo) {
    this.pricePerKilo = pricePerKilo;
}

public void setStockInKilos (double stockInKilos) {
    this.stockInKilos = stockInKilos;
}

private static long getNextId () {
    return Vegetable.currentId++;
}
}
```

- **VegetableSaleInfo**

```
package Entities;

import Abstractions.Profiles.IConsumerProfile;
import Abstractions.Profiles.UserProfileBase;

public class VegetableSaleInfo {
    private Item soldItem;
    private IConsumerProfile whoBoughtFromMe;
    public VegetableSaleInfo (Item item, IConsumerProfile
    whoBoughtFromMe) {
        this.soldItem = item;
        this.whoBoughtFromMe = whoBoughtFromMe;
    }
}
```

```
    public Item getSoldItem () {
        return this.soldItem;
    }

    public IConsumerProfile getWhoBoughtFromMe () {
        return this.whoBoughtFromMe;
    }

    public void Print () {
        UserProfileBase consumer
        = (UserProfileBase) this.whoBoughtFromMe;

        System.out.print("Consumer: " + consumer.getName() +
        "\t");
        this.soldItem.Print();
        System.out.println();
    }
}
```

- **Item**

```
package Entities;

public class Item {
    private Vegetable vegetable;
    private double quantity;

    public Item (Vegetable vegetable, double quantity) {
        this.vegetable = vegetable;
        this.quantity = quantity;
    }
}
```

```
public Vegetable getVegetable () {
    return this.vegetable;
}

public double getQuantity () {
    return this.quantity;
}

public void Print () {
    System.out.print ("Vegetable: " +
        this.vegetable.getName() + "\tQuantity: " +
        this.quantity + "\tRate: " +
        this.vegetable.getPricePerKilo());
}
}
```

- **Order**

```
package Entities;

import java.util.ArrayList;

public class Order {
    private static long currentId = 1;

    private ArrayList<Item> items = new ArrayList<Item> ();
    private long id;

    public Order () {
        this.id = Order.getNextOrderId();
    }
}
```

```
public double getTotal () {
    double total = 0;

    for (int i = 0; i < this.items.size(); i++) {
        Item itemInCart = this.items.get(i);
        total +=
        itemInCart.getVegetable().getPricePerKilo() *
        itemInCart.getQuantity();
    }

    return total;
}

public void Print () {
    System.out.println("=====");
    System.out.println("Order # "+this.id +" details: ");
    System.out.println("-----");
    System.out.println();

    for (int i = 0; i < this.items.size(); i++) {
        System.out.print "[" + (i + 1) + " ] ";
        this.items.get(i).Print();
        System.out.println();
    }

    System.out.println("-----");
    System.out.println("Order total: Rs. " +
    this.getTotal());
    System.out.println("=====");
    System.out.println();
    System.out.println();
}
```

```
    public void AddItem (Item itemToAdd) {
        this.items.add(itemToAdd);
    }

    public ArrayList<Item> getItems () {
        return this.items;
    }

    private static long getNextOrderId () {
        return Order.currentId++;
    }
}
```

- **Main**

```
import java.util.ArrayList;
import java.util.Scanner;

import Abstractions.Managers.IConsumerManager;
import Abstractions.Managers.IFarmerManager;
import Abstractions.Profiles.IFarmerProfile;
import Abstractions.Profiles.UserProfileBase;
import Aggregates.ConsumerManager;
import Aggregates.FarmerManager;
import Utilities.ScanUtils;
public class Main {
    private static IFarmerManager farmerManager
    = new FarmerManager ();
    private static IConsumerManager consumerManager
    = new ConsumerManager ();
    private static Scanner in = new Scanner (System.in);
```

```
public static void main (String args []) {
    int userChoice;

    while (true) {
        System.out.print ("What role do you want to
        login as?\n\n\t(1) Farmer \n\t(2)
        Consumer.\n\t(3) Exit application.\n\n
        Enter your choice here: ");
        userChoice = ScanUtils.getInt();

        switch (userChoice) {
            case 1:
                farmerManager.sayWelcome();
                farmerManager.initialize();
                farmerManager.showOptionsToUser();
                break;

            case 2:
                consumerManager.sayWelcome();
                consumerManager.initialize();
                // Get a list of farmers from farmer manager.
                ArrayList<UserProfileBase> farmers =
                farmerManager.getUsers();
                ArrayList<IFarmerProfile> availableFarmers = new
                ArrayList<IFarmerProfile> ();

                for (int i = 0; i < farmers.size(); i++) {
                    availableFarmers.add(((IFarmerProfile)
                    farmers.get(i)));
                }
            }
        }
    }
```

```
        // Transfer that list to the consumer manager.
        consumerManager.setAvailableFarmers
        (availableFarmers);

        // Show options
        consumerManager.showOptionsToUser();
        break;

    case 3:
        System.out.println ("Thank you for
        using Agro Kart! Have a nice day! ");
        System.exit (0);
        break;
    default:
        break;
    }
}
}
```

## 4.4 EXCEPTION HANDLING

- **ScanUtils**

```
package Utilities;

import java.util.Scanner;

public class ScanUtils {
    private static Scanner in = new Scanner (System.in);

    public static int getInt () {
```



```
        int valueFromConsole;

        while (true) {
            try {
                valueFromConsole = in.nextInt();
                return valueFromConsole;
            }
            catch (Exception ex) {
                System.out.println("Exception! You entered
an unacceptable value for an integer.
Please try again!");
                String buffer = in.nextLine();
            }
        }
    }

    public static long getLong () {
        long valueFromConsole;

        while (true) {
            try {
                valueFromConsole = in.nextLong();
                return valueFromConsole;
            }
            catch (Exception ex) {
                System.out.println("Exception! You entered
an unacceptable value for a long. Please
try again!");
                String buffer = in.nextLine();
            }
        }
    }
}
```

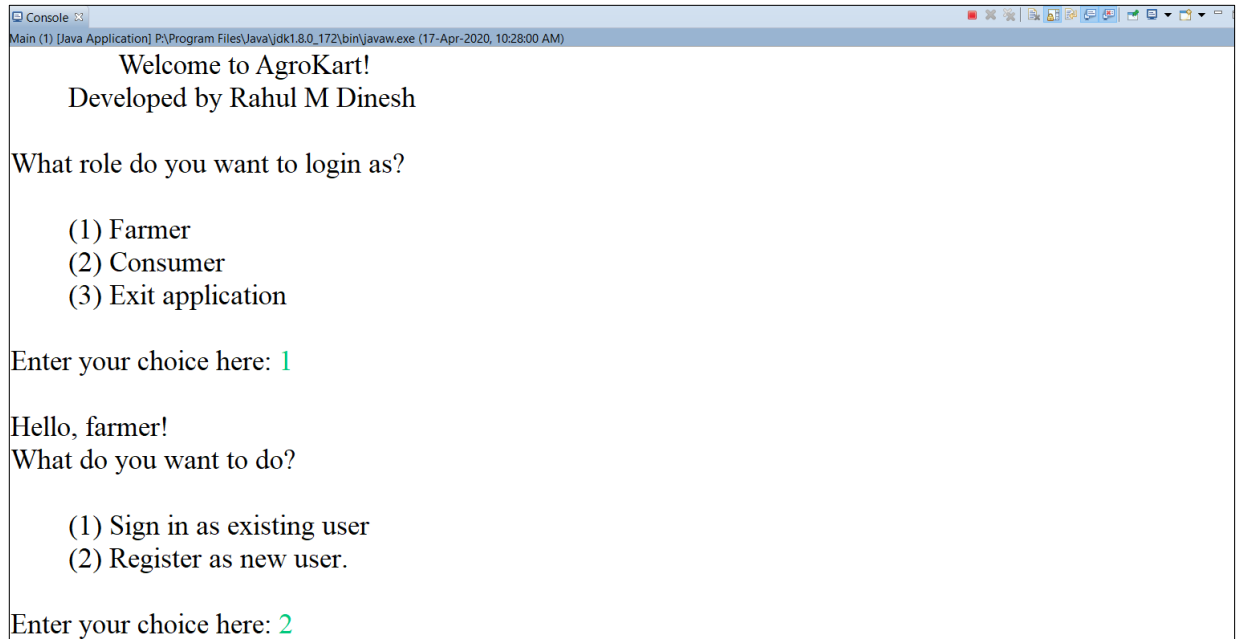
```
public static double getDouble () {  
    double valueFromConsole;  
  
    while (true) {  
        try {  
            valueFromConsole = in.nextDouble();  
            return valueFromConsole;  
        }  
  
        catch (Exception ex) {  
            System.out.println("Exception! You entered  
an unacceptable value for a double. Please  
try again!");  
            String buffer = in.nextLine();  
        }  
    }  
}
```

## CHAPTER 5

## RESULTS

### 5.1 REGISTERING A NEW USER

- Farmer



```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)

Welcome to AgroKart!
Developed by Rahul M Dinesh

What role do you want to login as?

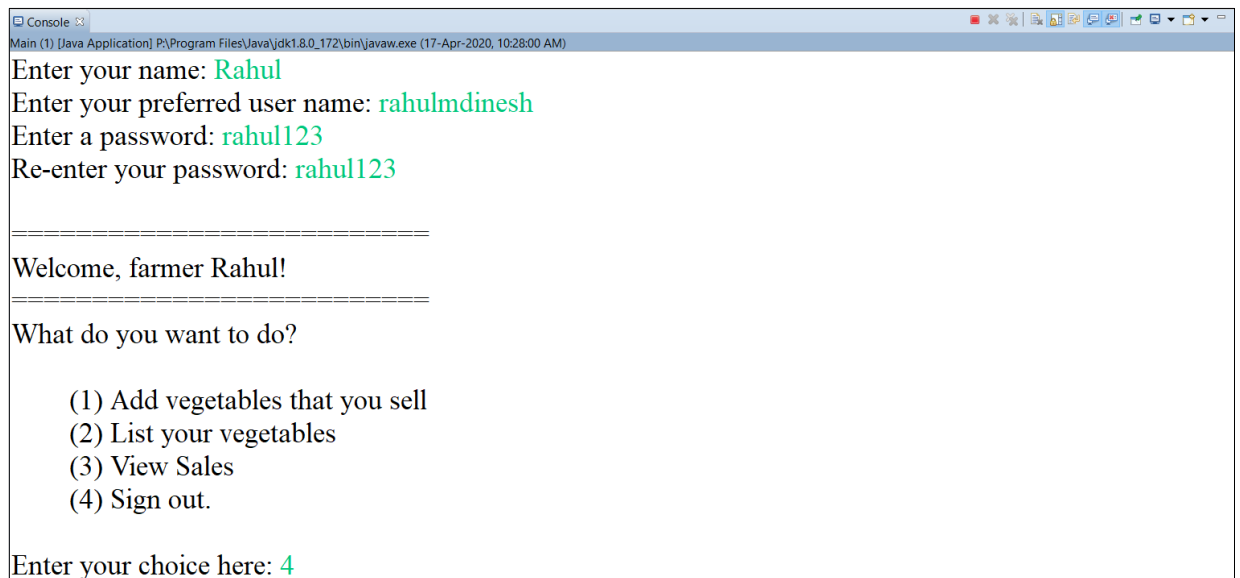
(1) Farmer
(2) Consumer
(3) Exit application

Enter your choice here: 1

Hello, farmer!
What do you want to do?

(1) Sign in as existing user
(2) Register as new user.

Enter your choice here: 2
```



```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)

Enter your name: Rahul
Enter your preferred user name: rahulmdinesh
Enter a password: rahul123
Re-enter your password: rahul123

=====

Welcome, farmer Rahul!

=====

What do you want to do?

(1) Add vegetables that you sell
(2) List your vegetables
(3) View Sales
(4) Sign out.

Enter your choice here: 4
```

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)
What role do you want to login as?

(1) Farmer
(2) Consumer
(3) Exit application

Enter your choice here: 1

Hello, farmer!
What do you want to do?

(1) Sign in as existing user
(2) Register as new user.

Enter your choice here: 2
```

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)
Enter your name: Suja
Enter your preferred user name: sujathak
Enter a password: suja123
Re-enter your password: suja123

=====
Welcome, farmer Suja!
=====

What do you want to do?

(1) Add vegetables that you sell
(2) List your vegetables
(3) View Sales
(4) Sign out.

Enter your choice here: 4
```

### ● Consumer

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)
What role do you want to login as?

(1) Farmer
(2) Consumer
(3) Exit application

Enter your choice here: 2

Hello, consumer!
What do you want to do?

(1) Sign in as existing user
(2) Register as new user.

Enter your choice here: 2
```

Enter your name: **Dinesh**  
Enter your preferred user name: **dineshmj**  
Enter a password: **dinesh123**  
Re-enter your password: **dinesh123**  
How much fund do you want to add? Rs. **500**

=====

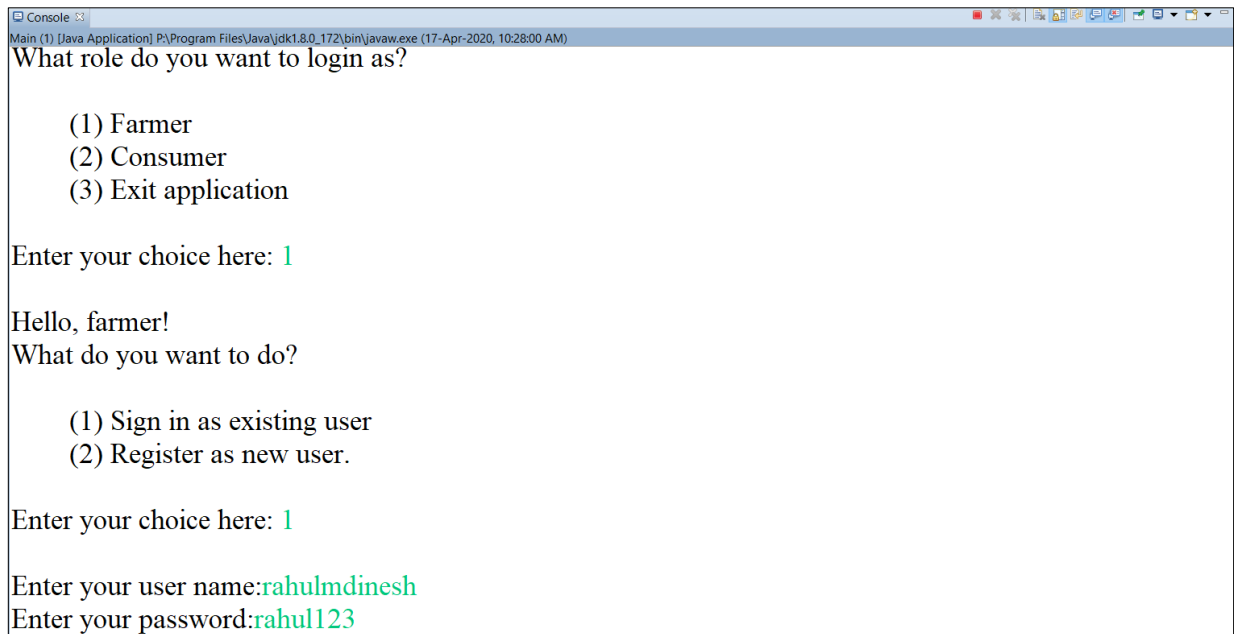
Welcome, consumer Dinesh!

=====

What do you want to do?

- (1) Display all farmers and their products
- (2) Add items to cart
- (3) View orders
- (4) Top up balance
- (5) Sign out.

## 5.2 ADDING VEGETABLES IN FARMER PROFILE



Console

Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0\_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)

What role do you want to login as?

- (1) Farmer
- (2) Consumer
- (3) Exit application

Enter your choice here: **1**

Hello, farmer!

What do you want to do?

- (1) Sign in as existing user
- (2) Register as new user.

Enter your choice here: **1**

Enter your user name:**rahulmdinesh**

Enter your password:**rahul123**

=====

Welcome, farmer Rahul!

=====

What do you want to do?

- (1) Add vegetables that you sell
- (2) List your vegetables
- (3) View Sales
- (4) Sign out.

Enter your choice here: **1**

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)
How many types of vegetables are you selling? 3

Enter first vegetable's name: Beans
Enter vegetable's price per Kilo: Rs. 120
Enter vegetable's stock in kg that you have to sell: 10
-----
Enter second vegetable's name: Potato
Enter vegetable's price per Kilo: Rs. 40
Enter vegetable's stock in kg that you have to sell: 6
-----
Enter third vegetable's name: Carrot
Enter vegetable's price per Kilo: Rs. 35
Enter vegetable's stock in kg that you have to sell: 9
-----
What do you want to do?

(1) Add vegetables that you sell
(2) List your vegetables
(3) View Sales
(4) Sign out.
```

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)
Enter your choice here: 2
Farmer's name: Rahul

Vegetable Code: 1 | Vegetable: Beans | Price per kilo: Rs. 120.0 | Stock available: 10.0 kg.
Vegetable Code: 2 | Vegetable: Potato | Price per kilo: Rs. 40.0 | Stock available: 6.0 kg.
Vegetable Code: 3 | Vegetable: Carrot | Price per kilo: Rs. 35.0 | Stock available: 9.0 kg.

What do you want to do?

(1) Add vegetables that you sell
(2) List your vegetables
(3) View Sales
(4) Sign out.

Enter your choice here: 3
=====
Sales Info
=====
Total money in account: Rs. 0.0
```

```
What do you want to do?

(1) Add vegetables that you sell
(2) List your vegetables
(3) View Sales
(4) Sign out.

Enter your choice here: 4
```

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)

What role do you want to login as?

(1) Farmer
(2) Consumer
(3) Exit application

Enter your choice here: 1

Hello, farmer!
What do you want to do?

(1) Sign in as existing user
(2) Register as new user.

Enter your choice here: 1

Enter your user name:sujathak
Enter your password:suja123
```

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)

=====
Welcome, farmer Suja!
=====

What do you want to do?

(1) Add vegetables that you sell
(2) List your vegetables
(3) View Sales
(4) Sign out.

Enter your choice here: 1
How many types of vegetables are you selling? 3

Enter first vegetable's name: Onion
Enter vegetable's price per Kilo: Rs. 40
Enter vegetable's stock in kg that you have to sell: 7
=====
Enter second vegetable's name: Beetroot
Enter vegetable's price per Kilo: Rs. 35
Enter vegetable's stock in kg that you have to sell: 5
```

```
=====
Enter third vegetable's name: Cabbage
Enter vegetable's price per Kilo: Rs. 30
Enter vegetable's stock in kg that you have to sell: 4
=====

What do you want to do?

(1) Add vegetables that you sell
(2) List your vegetables
(3) View Sales
(4) Sign out.
```

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (17-Apr-2020, 10:28:00 AM)
Enter your choice here: 3

=====
Sales Info
=====

Total money in account: Rs. 0.0

=====

What do you want to do?

(1) Add vegetables that you sell
(2) List your vegetables
(3) View Sales
(4) Sign out.

Enter your choice here: 4
```

### 5.3 ADDING VEGETABLES IN TO CONSUMER'S CART

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (18-Apr-2020, 11:45:57 AM)
What role do you want to login as?

(1) Farmer
(2) Consumer
(3) Exit application

Enter your choice here: 2

Hello, consumer!
What do you want to do?

(1) Sign in as existing user.
(2) Register as new user.

Enter your choice here: 1

Enter your user name:dineshmj
Enter your password:dinesh123

=====
Welcome, consumer Dinesh!
=====

What do you want to do?

(1) Display all farmers and their products
(2) Add items to cart
(3) View orders
(4) Top up balance
(5) Sign out.
```



```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (18-Apr-2020, 6:58:25 PM)
Enter your choice here: 1
Farmer's name: Rahul

Vegetable Code: 1 | Vegetable: Beans | Price per kilo: Rs. 120.0 | Stock available: 10.0 kg.
Vegetable Code: 2 | Vegetable: Potato | Price per kilo: Rs. 40.0 | Stock available: 6.0 kg.
Vegetable Code: 3 | Vegetable: Carrot | Price per kilo: Rs. 35.0 | Stock available: 9.0 kg.

Farmer's name: Suja

Vegetable Code: 4 | Vegetable: Onion | Price per kilo: Rs. 40.0 | Stock available: 7.0 kg.
Vegetable Code: 5 | Vegetable: Beetroot | Price per kilo: Rs. 35.0 | Stock available: 5.0 kg.
Vegetable Code: 6 | Vegetable: Cabbage | Price per kilo: Rs. 30.0 | Stock available: 4.0 kg.

What do you want to do?

(1) Display all farmers and their products
(2) Add items to cart
(3) View orders
(4) Top up balance
(5) Sign out.
```

```
Enter your choice here: 2

Specify vegetable code: 1
Specify the quantity: 3

Do you want to add any more vegetables? (Yes/No): Yes

Specify vegetable code: 2
Specify the quantity: 2

Do you want to add any more vegetables? (Yes/No): Yes

Specify vegetable code: 3
Specify the quantity: 10

Do you want to add any more vegetables? (Yes/No): Yes

Specify vegetable code: 4
Specify the quantity: 8
```

```
Do you want to add any more vegetables? (Yes/No): Yes

Specify vegetable code: 5
Specify the quantity: 1

Do you want to add any more vegetables? (Yes/No): Yes

Specify vegetable code: 6
Specify the quantity: 3

Do you want to add any more vegetables? (Yes/No): No
```

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (18-Apr-2020, 11:45:57 AM)

Checking out the items in cart ...
====> From farmer 'Rahul': The requested quantity 10.0 kg. of 'Carrot' is not available.
====> From farmer 'Suja': The requested quantity 8.0 kg. of 'Onion' is not available.
====> Payment amount for 3.0 kg. of Cabbage is Rs. 90.0, however, you have only Rs. 25.0
      left in your account. Unable to continue with the purchase.

=====
Order # 1 details: -
-----

[1] Vegetable: Beans      Quantity: 3.0  Rate: 120.0
[2] Vegetable: Potato     Quantity: 2.0  Rate: 40.0
[3] Vegetable: Beetroot   Quantity: 1.0  Rate: 35.0
-----
Order total: Rs. 475.0

=====

You have Rs. 25.0 left in your account.
What do you want to do?

What do you want to do?

(1) Display all farmers and their products
(2) Add items to cart
(3) View orders
(4) Top up balance
(5) Sign out.

Enter your choice here: 3

=====
Order # 1 details: -
-----

[1] Vegetable: Beans      Quantity: 3.0  Rate: 120.0
[2] Vegetable: Potato     Quantity: 2.0  Rate: 40.0
[3] Vegetable: Beetroot   Quantity: 1.0  Rate: 35.0
-----
Order total: Rs. 475.0

=====

What do you want to do?

(1) Display all farmers and their products
(2) Add items to cart
(3) View orders
(4) Top up balance
(5) Sign out.

Enter your choice here: 4

=====
Your account balance is: Rs 25.0
=====
```

How much fund would you like to add to your existing balance? Rs 300  
What do you want to do?

- (1) Display all farmers and their products
- (2) Add items to cart
- (3) View orders
- (4) Top up balance
- (5) Sign out.

Enter your choice here: 5  
What role do you want to login as?

- (1) Farmer
- (2) Consumer
- (3) Exit application

Console 33  
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0\_172\bin\javaw.exe (18-Apr-2020, 11:45:57 AM)

Enter your choice here: 1

Hello, farmer!

What do you want to do?

- (1) Sign in as existing user.
- (2) Register as new user.

Enter your choice here: 1

Enter your user name:sujathak

Enter your password:suja123

=====

Welcome, farmer Suja!

=====

What do you want to do?

- (1) Add vegetables that you sell
- (2) List your vegetables
- (3) View Sales
- (4) Sign out.

Enter your choice here: 3

=====

Sales Info

=====

Total money in account: Rs. 35.0

Consumer: Dinesh Vegetable: Beetroot Quantity: 1.0 Rate: 35.0

=====

What do you want to do?

- (1) Add vegetables that you sell
- (2) List your vegetables
- (3) View Sales
- (4) Sign out.

Enter your choice here: 4

What role do you want to login as?

- (1) Farmer
- (2) Consumer
- (3) Exit application

Enter your choice here: 1

Hello, farmer!

What do you want to do?

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (18-Apr-2020, 11:45:57 AM)

(1) Sign in as existing user.
(2) Register as new user.

Enter your choice here: 1

Enter your user name:rahulmdinesh
Enter your password:rahul123

=====
Welcome, farmer Rahul!
=====

What do you want to do?

(1) Add vegetables that you sell
(2) List your vegetables
(3) View Sales
(4) Sign out.

Enter your choice here: 3
```

```
Console
Main (1) [Java Application] P:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (18-Apr-2020, 11:45:57 AM)

=====
Sales Info
=====

Total money in account: Rs. 440.0

Consumer: Dinesh Vegetable: Beans Quantity: 3.0 Rate: 120.0
Consumer: Dinesh Vegetable: Potato Quantity: 2.0 Rate: 40.0

=====

What do you want to do?
```

- (1) Add vegetables that you sell
- (2) List your vegetables
- (3) View Sales
- (4) Sign out.

Enter your choice here: 4

What role do you want to login as?

- (1) Farmer
- (2) Consumer
- (3) Exit application

Enter your choice here: 3

Thank you for using Agro Kart! Have a nice day!

## CHAPTER 6

### CONCLUSION

The mini project has successfully accomplished the goals it had set out in the objectives and design sections of this report.

The Java application developed in this mini project has been able to provide a simple and easy to use platform for the farmer and consumer to sell and buy goods respectively.

Separate portals have been made for the farmers and consumers. Similar login interfaces have been used for both types of users which helps the user to either register as a new user or login as an existing user.

The farmer portal helps the farmer to carryout his or her various activities like adding vegetables, listing his products and selling his goods to the consumers along with a provision to see his or her revenue made.

The consumer portal has successfully implemented the various requirements of a consumer like viewing the catalogue of each farmer available, adding items to his or her cart, topping up his account balance, viewing his orders among others.

The application has also appropriately implemented the various object-oriented concepts concepts like objects, encapsulation by the usage of various classes and packages, abstraction through the many abstract classes and interfaces used and polymorphism using inheritance.

Exceptions generated by user inputs have also been handled in a suitable manner using the Java feature of exception handling using try and catch statements.

Through this mini project, I have been able to enhance my knowledge about Java and its various concepts. It has also helped me understand the real time implementation of the above-mentioned concepts.

In the future, I aim to extend this mini project using suitable GUIs or through a mobile application along with additional features.

## REFERENCES

- [1] Herbert Schildt, Java™: The Complete Reference, McGraw-Hill, Tenth Edition, 2018
- [2] Cay S. Horstmann, Core Java® SE 9 for the Impatient, Addison Wesley, Second Edition, 2018
- [3] SAMS teach yourself Java – 2: 3rd Edition by Rogers Cedenhead and Leura Lemay Publication Pearson Education.
- [4] Ken Kousen, Modern Java Recipes, O'Reilly Media, Inc., 2017
- [5] Introduction to Java OOPs concepts:  
<https://stackify.com/oops-concepts-in-java/>
- [6] Encapsulation:  
<https://www.programiz.com/java-programming/encapsulation>
- [7] Polymorphism:  
<https://www.edureka.co/blog/polymorphism-in-java/>
- [8] Abstraction:  
<https://howtodoinjava.com/oops/understanding-abstraction-in-java/>
- [9] Exception handling:  
<https://beginnersbook.com/2013/04/java-exception-handling/>