Refrigerator Simulator

Object-Oriented Programming- Final Project

Bobby Atwal
CS 151, Mr. Yeung

Use Cases

Use Case 1

- Change Temperature

1. Input new temperature
2. Update temperature

Use Cases

Use Case 2

- Open Refrigerator

1. Inside view of Refrigerator
2. Total number of items in the refrigerator
3. Add and remove items

Use Cases

Use Case 3

- Close Refrigerator

1. Hide refrigerator internal view
2. Current state of refrigerator insert into file.

User Cases

Use Case 4

- Fridge App

1. Display another view
2. Show all details of refrigerator.

CRC1

Items	
Items Names Add items Remove items Show items	GUI

CRC2

Timer	
Start Time Stop Time Total Time Print Time	GUI Refrigerator

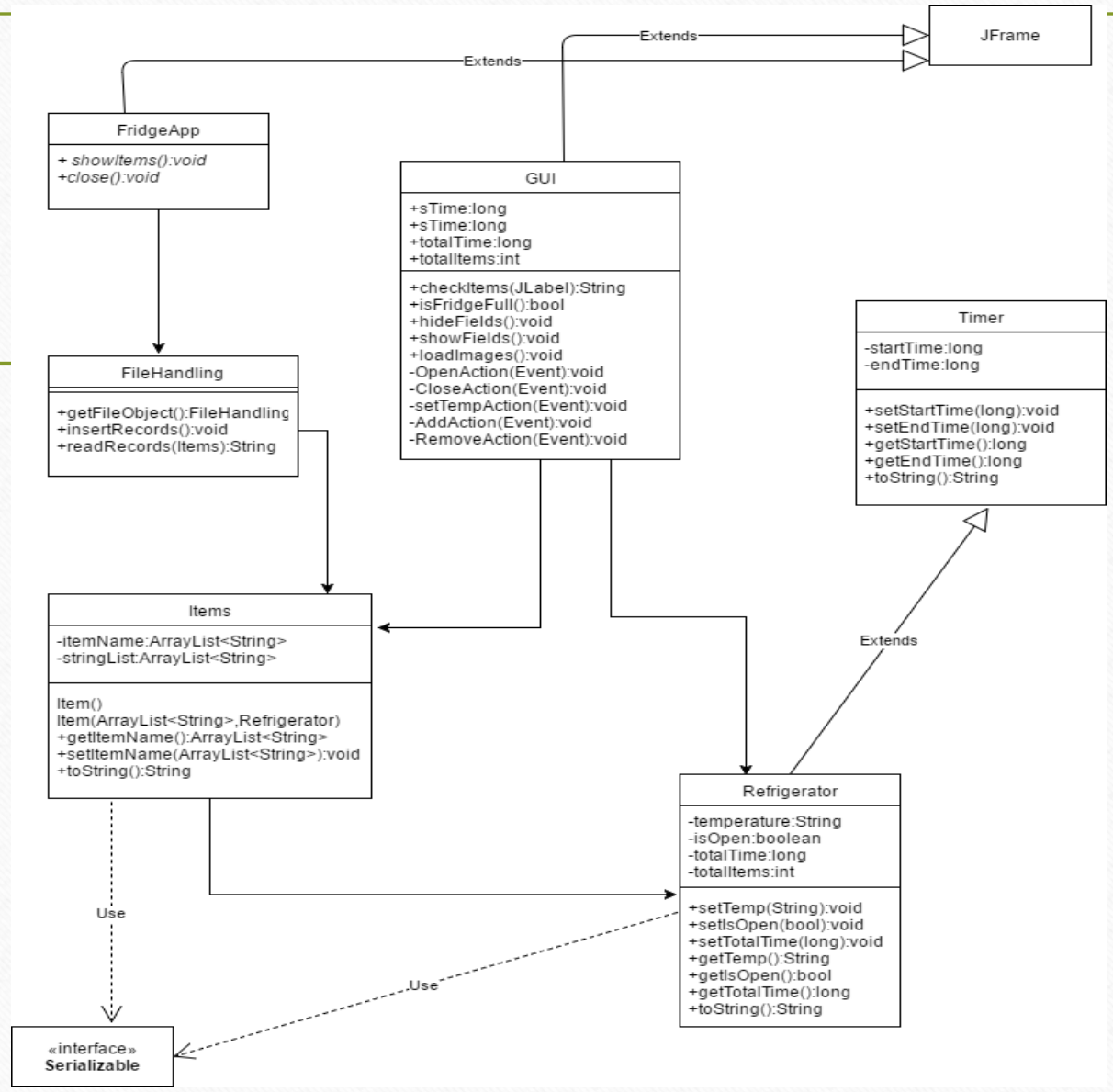
CRC3

Refrigerator	
Temperature Open Close Total Items Total Time	GUI Items

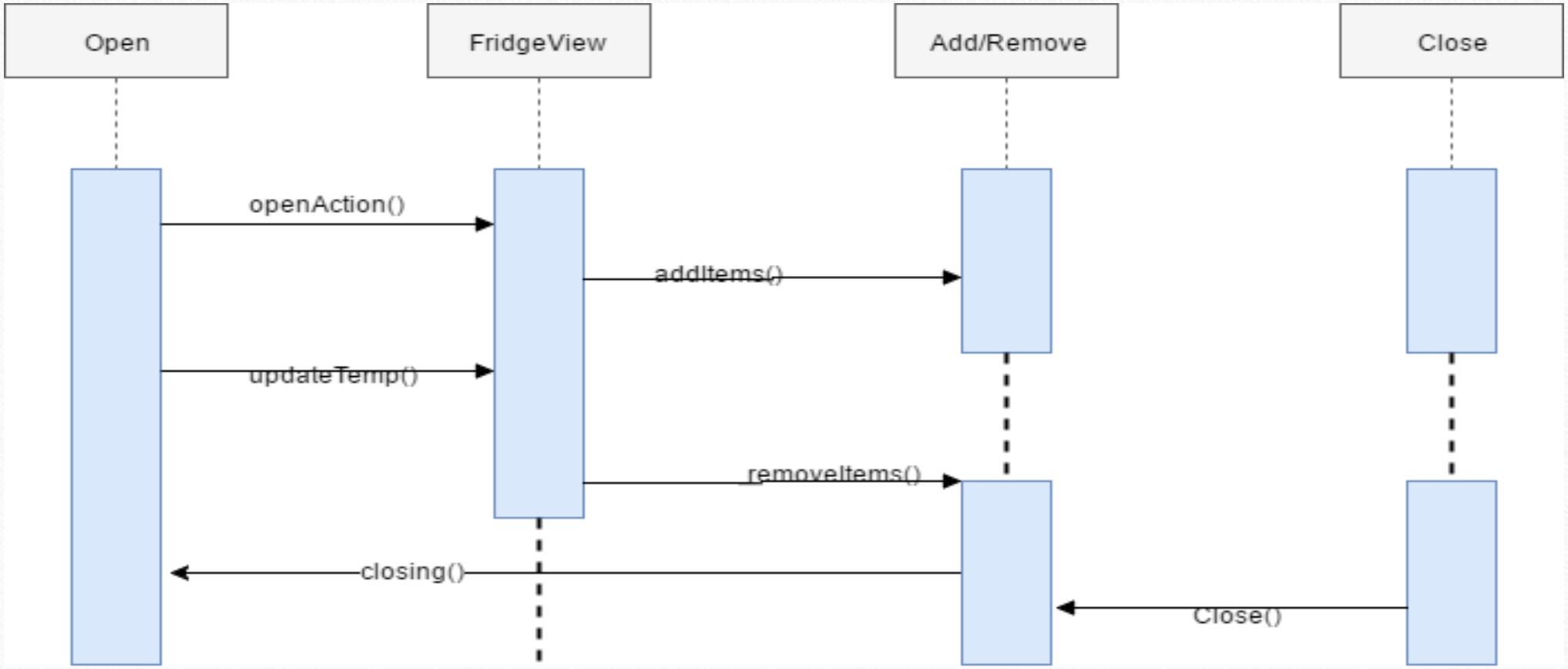
CRC4

FileHandling	
Insert Record Read Record	GUI FridgeApp

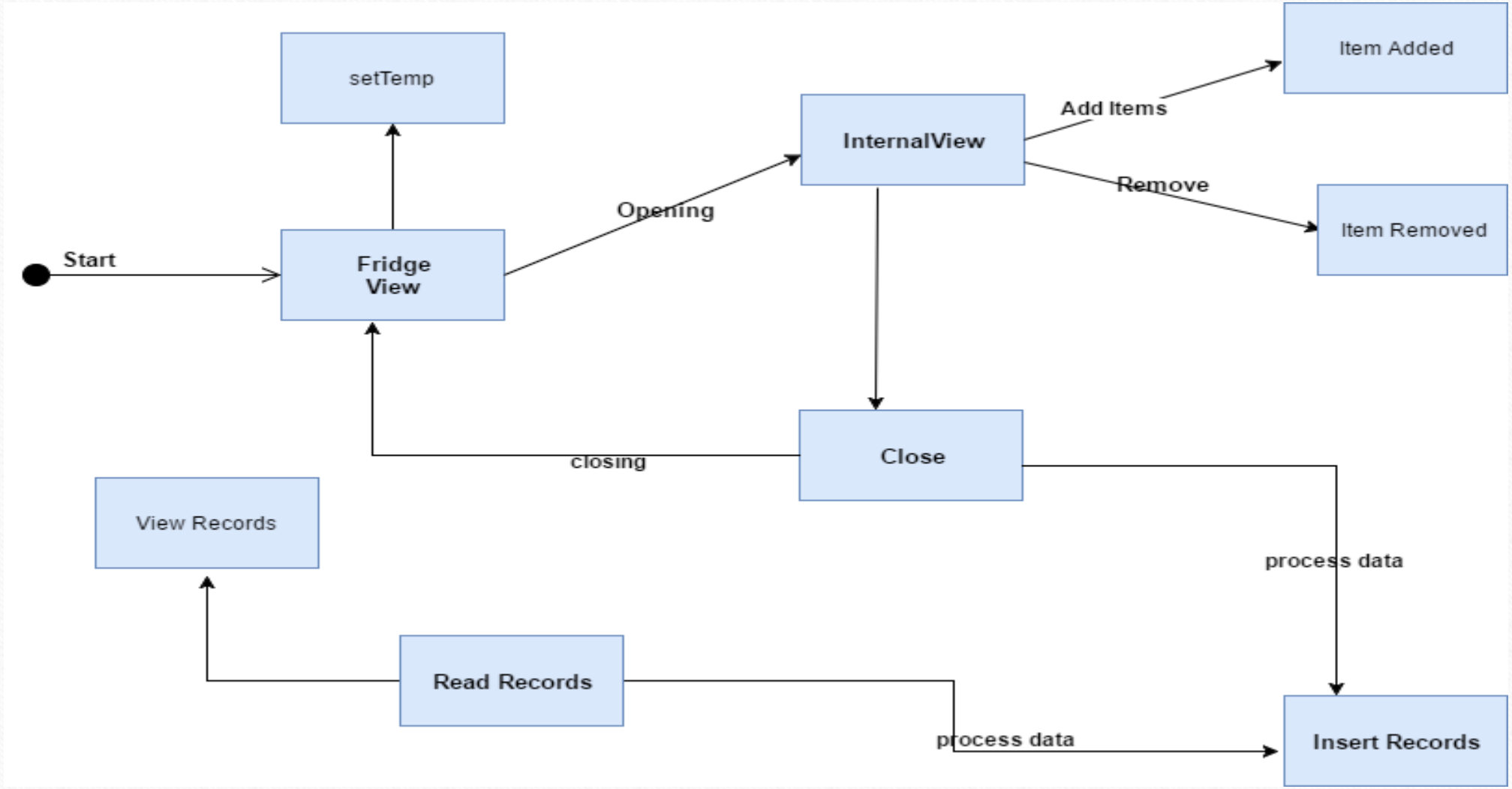
UML Diagram


JFrame
Extends
Extends
FridgeApp
+ showItems():void
+close():void
GUI
+sTime:long
+sTime:long
+totalTime:long
+totalItems:int
+checkItems(JLabel):String
+isFridgeFull():bool
+hideFields():void
+showFields():void
+loadImages():void
-OpenAction(Event):void
-CloseAction(Event):void
-setTempAction(Event):void
-AddAction(Event):void
-RemoveAction(Event):void
Timer
-startTime:long
-endTime:long
+setStartTime(long):void
+setEndTime(long):void
+getStartTime():long
+getEndTime():long
+toString():String
FileHandling
+getFileObject():FileHandling
+insertRecords():void
+readRecords(Items):String
Items
-itemName:ArrayList<String>
-stringList:ArrayList<String>
Item()
Item(ArrayList<String>,Refrigerator)
+getItemName():ArrayList<String>
+setItemName(ArrayList<String>):void
+toString():String
Extends
Refrigerator
-temperature:String
-isOpen:boolean
-totalTime:long
-totalItems:int
+setTemp(String):void
+setIsOpen(bool):void
+setTotalTime(long):void
+getTemp():String
+getIsOpen():bool
+getTotalTime():long
+toString():String
Use
Use
«interface»
Serializable

Sequence Diagram



State Diagram



JavaDocs 1

```java
/**
 *
 * @Name : Refrigerator.java
 * @Description : entends parent class Timer also Implements serializable
 *
 *
 */

public class Refrigerator extends Timer implements Serializable{

    //Instance variables
    private String temperature;
    private boolean isOpen;
    private long totalTime;
    private int totalItems;


    //Setters and Getters
    public String getTemperature() {
        return temperature;
    }
```

JavaDocs 2

```java
/**
 *
 * @Name : Timer.java
 * @Description : Parent class define the Timer business logic
 *
 *
 */
public class Timer {

    //Instance variables
    private long startTime;
    private long stopTime;

    //get start time
    public long getStartTime() {
        return startTime;
    }
    //set start time
    public void setStartTime(long startTime) {
        this.startTime = startTime;
    }
```

JavaDocs 3

```java
/**
 *
 * @Name : Items.java
 * @Description : Class define items in the Refrigerator also implements Serializable
 *
 *
 */

public class Items implements Serializable{

    //Instance variables
    private ArrayList<String> itemName;
    private final ArrayList<String> stringList = new ArrayList<>();
    private Refrigerator rf;

    //Default constructor
    Items(){

    }
```

5C Criteria

```
public class Refrigerator extends Timer implements Serializable{

    //Instance variables
    private String temperature;
    private boolean isOpen;
    private long totalTime;
    private int totalItems;
```

5C Criteria

1. **Cohesion** class is responsible for doing one thing only.
2. **Completeness** class is completely describe the purpose.
3. **Convenience** class nothing do itself but access to other classes.
4. **Clarity** class is clearly define the logic of all methods and it's behaviour
5. **Consistency** class provide consistency in logic and all operations.

Unit Test

```
@Test
public void testFridgeItems() {
    Refrigerator rf = new Refrigerator();
    rf.setTotalItems(3);

    assertEquals(3,rf.getTotalItems());

}
```

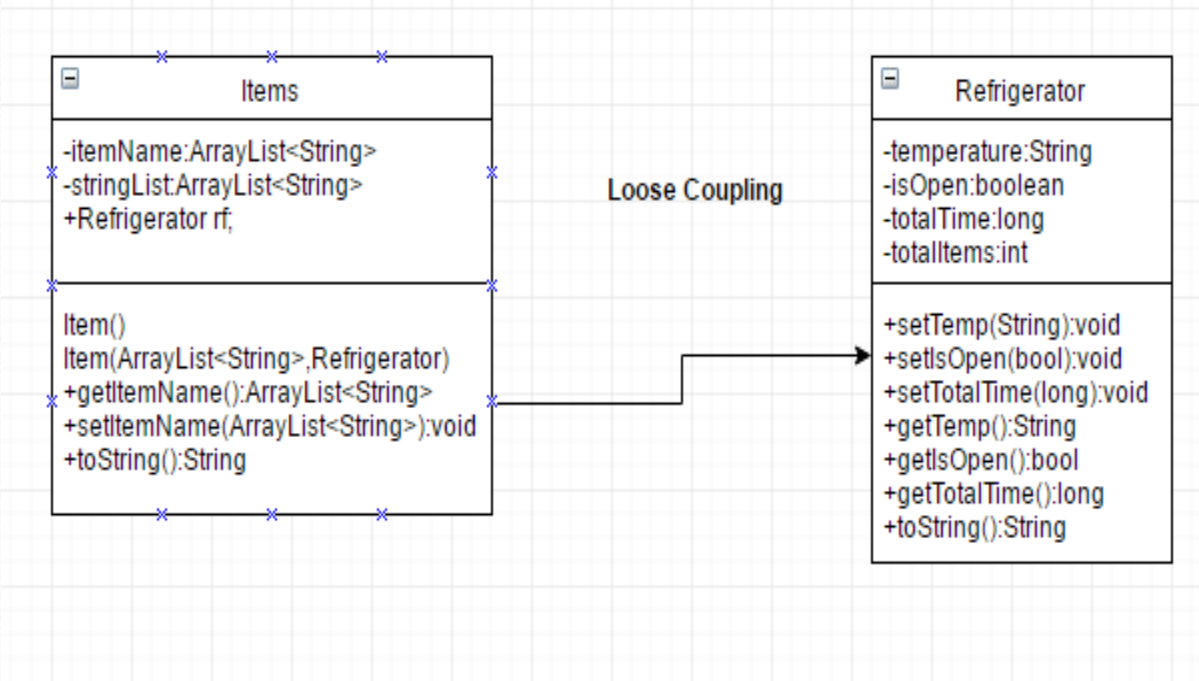
Encapsulated 1

```
public class Refrigerator extends Timer implements Serializable{

    //Instance variables
    private String temperature;
    private boolean isOpen;
    private long totalTime;
    private int totalItems;


    //Setters and Getters
    public String getTemperature() {
        return temperature;
    }

    public void setTemperature(String temperature) {
        this.temperature = temperature;
    }

    public boolean isIsOpen() {
        return isOpen;
    }

    public void setIsOpen(boolean isOpen) {
```

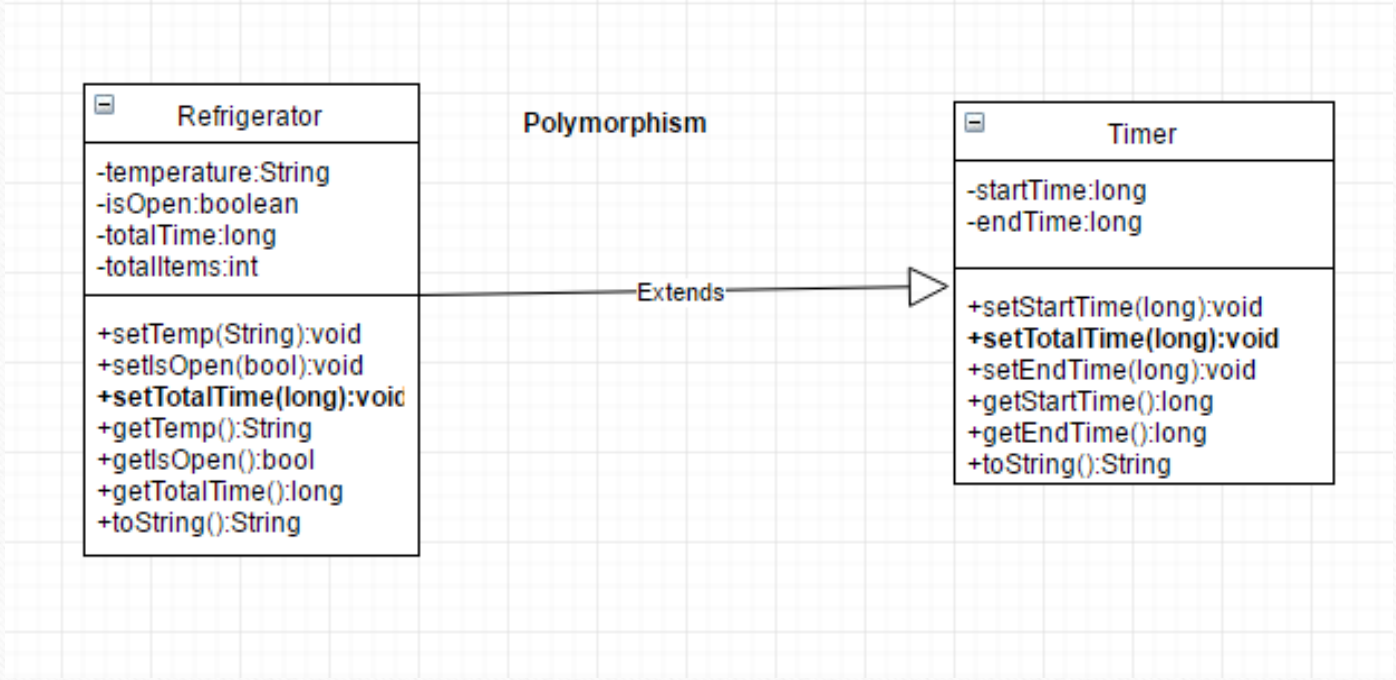
Encapsulated 2

```
public class Items implements Serializable{

    //Instance variables
    private ArrayList<String> itemName;
    private final ArrayList<String> stringList = new ArrayList<>();
    private Refrigerator rf;

    //Default constructor
    Items(){

    }

    //Arguments constructor
     Items(ArrayList<String> in, Refrigerator r){

        this.itemName = in;
        this.rf = r;

    }

     //Setter and Getter
    public ArrayList<String> getItemName() {
        return itemName;
    }
```

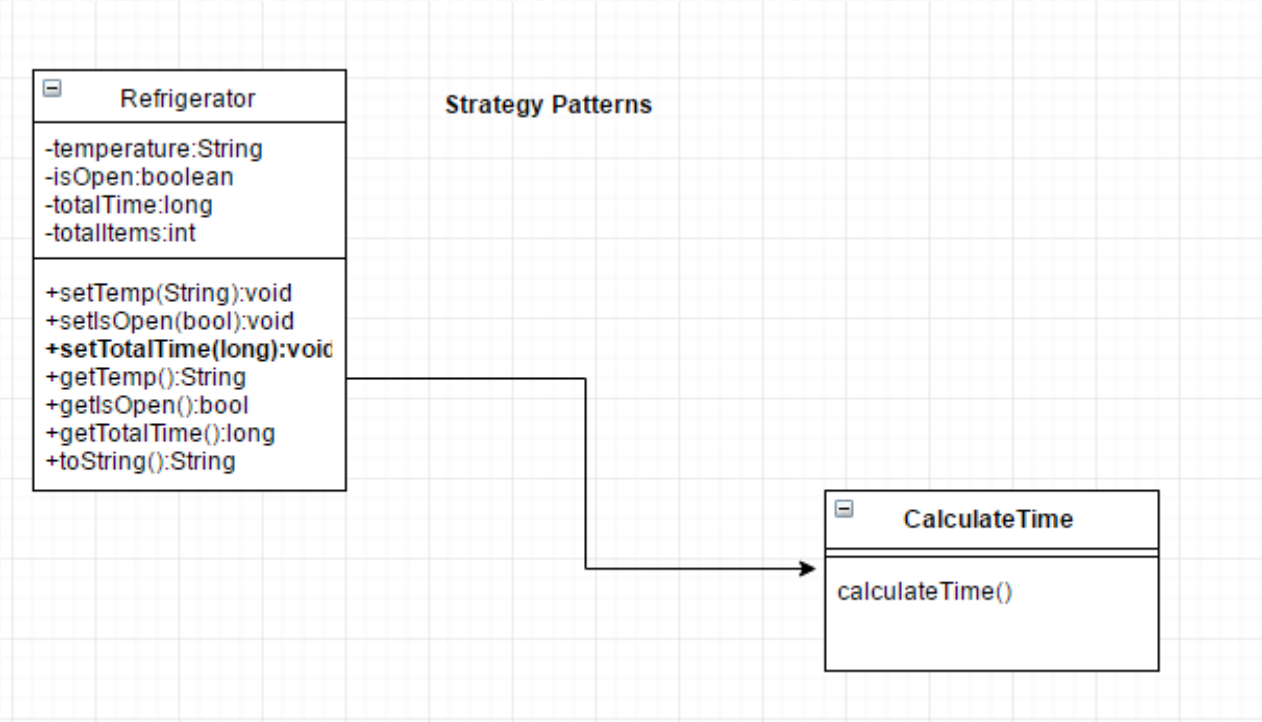
Loose Coupling



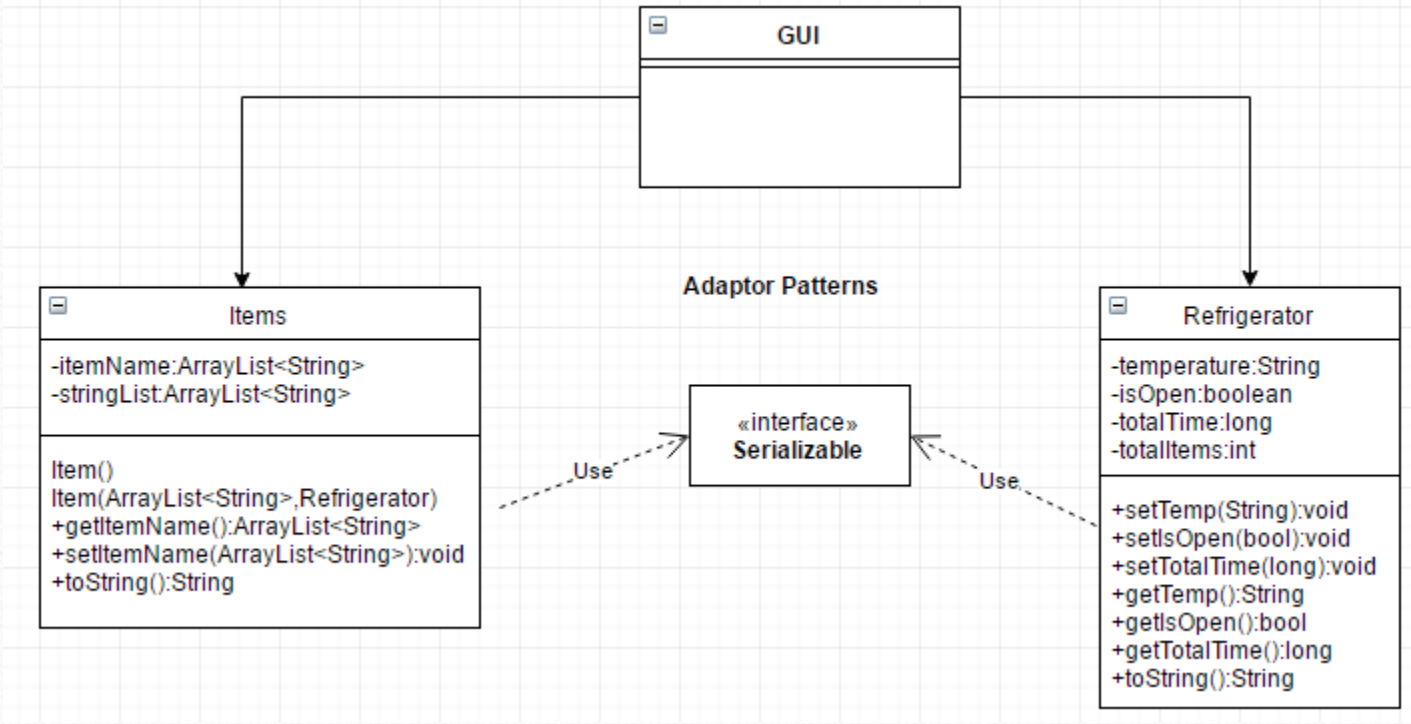
Polymorphism



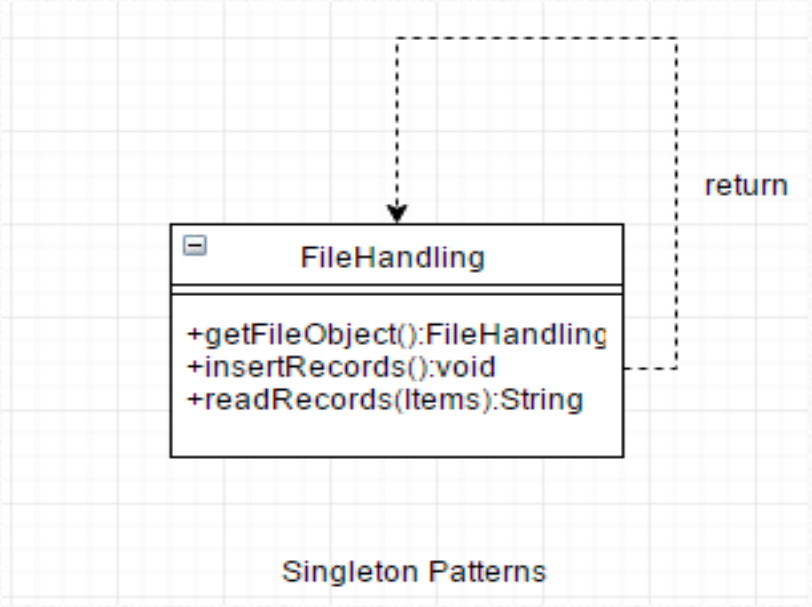
Behaviour Design Patterns



Structural Design Patterns



Creational Design Patterns



Override equals() method

Timer Class

```
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Timer other = (Timer) obj;
    if (startTime != other.startTime)
        return false;
    if (stopTime != other.stopTime)
        return false;
    return true;
}
```

Serialization

```
    //Insert Object
    public void insertRecords(Items item){



          try {
        FileOutputStream fop=new FileOutputStream("items.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fop);

         oos.writeObject(item);
         System.out.println("Data is inserted!!!");
    } catch (Exception e) {
}
     }

    //Read Object
    public String readRecords(Items item) throws FileNotFoundException, IOException, ClassNotFoundE

          FileInputStream fis=new FileInputStream("items.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);

        item=(Items)ois.readObject();

       return item.toString();
    }
```

Reflection

```
//Reflection invoke

Class myClass = fileIO.getClass();
Method fileObject;
try {
    fileObject = myClass.getMethod("getFileObject", new Class[] {});

fileIO = (FileHandling) fileObject.invoke(fileIO, new Object[] {});
}
catch (Exception e) {

    e.printStackTrace();
}
```

Threads a)

```
//Thread use for time calculation
 private static void calculateTime() throws InterruptedException {

    //Sleep 2 seconds
    TimeUnit.SECONDS.sleep(2);

}
```

Threads b)

```
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new GUI().setVisible(true);
            }
        });
    }



    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new FridgeApp().setVisible(true);
        }
    });
}
```