SC2002
GROUP PROJECT

FAST FOOD MANAGEMENT SYSTEM

Members

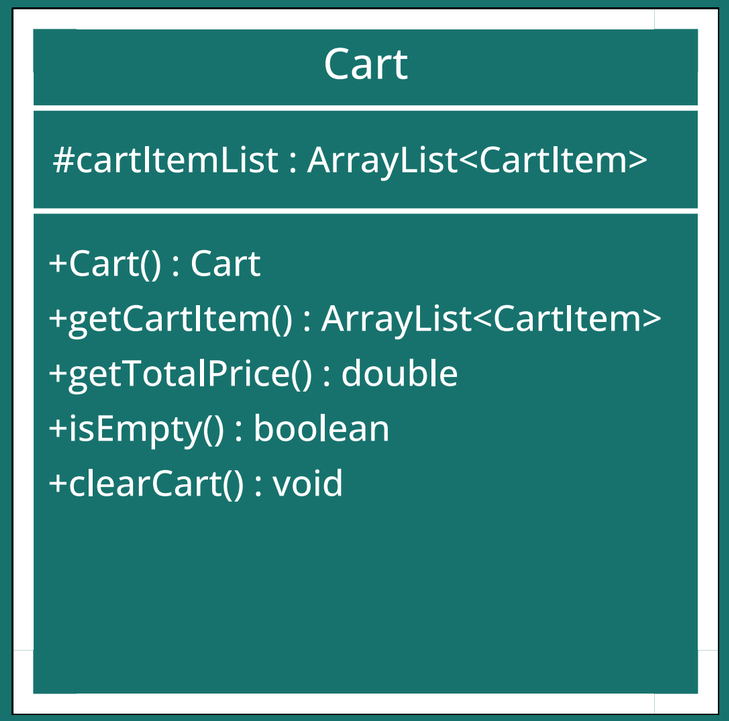
Choo Wei Boon
Duong Hoang Vu Lam
Goh Jie Rong, Sean
Krystal Pek Ke Yun
Wong Kang Yi, Rhyan

TABLE OF CONTENT

APPROACH

Design Rationale

1. Actors and related interfaces, simplicity
2. UML Diagram
3. SOLID principles - Low Coupling, High Cohesion
4. Code Compatibility and Resolving Conflicts



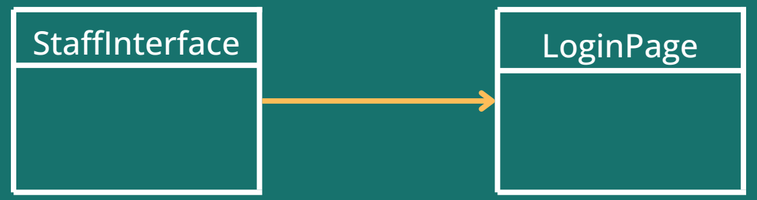
UML DIAGRAM

Process to construct UML Diagram

1. (Develop the code)
2. Create the boxes of class
   - Access modifier
   - Attributes
   - Methods
   - Datatype (for attributes & methods)
   - Enumeration (if there is)
3. Define the connection
   - Relation between classes

Dependency:

Association:

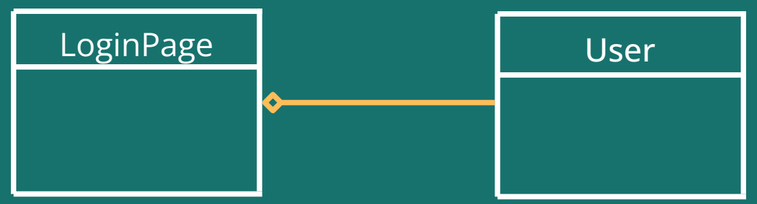


UML DIAGRAM

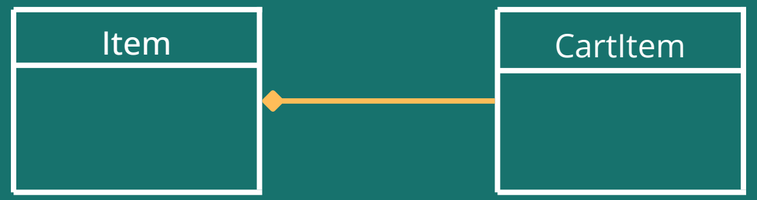
Dependency vs Association

- Dependency relation
  - A references B (as a method parameter or return type)
- Association relation
  - A has-a C object (as a member variable)
  - Stronger relationship than Dependency

Aggregation:



Composition:



UML DIAGRAM

Association: Aggregation vs Composition

- Aggregation
  - A utilizes B (B exist independently from A)
- Composition
  - A owns C (C has no meaning without A)
  - Stronger relationship than Aggregation

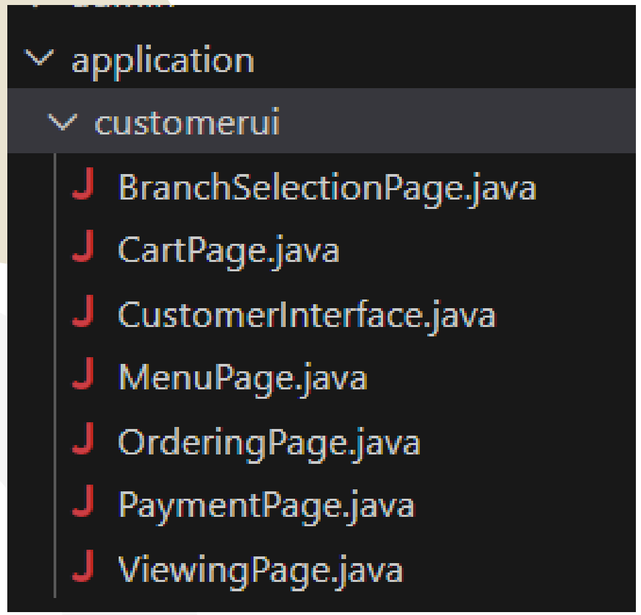
CODE DEMO

```java
import java.io.FileInputStream;

public class FileRead {
    private static final String directory = System.getProperty("user.dir") + "/resources/";
    private static final String SEPARATOR = ",";

    public static void loadBranches(String filename, ArrayList<Branch> branchList) throws IOException {
        // read String from text file
        ArrayList<String> stringArray = (ArrayList<String>)read(filename);

        for (int i = 1; i < stringArray.size(); i++) {
            String st = (String)stringArray.get(i);
            // get individual 'fields' of the string separated by SEPARATOR
            StringTokenizer star = new StringTokenizer(st, SEPARATOR);

            String name = star.nextToken().trim();
            String location = star.nextToken().trim();
            int staffQuota = Integer.parseInt(star.nextToken().trim());
            String operationStatus = star.nextToken().trim().toUpperCase();

            try {
                OperationStatus os = OperationStatus.valueOf(operationStatus);
                Branch branch = new Branch(name, location, staffQuota, os);
                branchList.add(branch);
            } catch (IllegalArgumentException e) {
                System.err.println("Error creating Branch object: " + e.getMessage());
            }
        }
    }
    private static List<String> read(String fileName) throws IOException {
        List<String> data = new ArrayList<String>();
        Scanner scanner = new Scanner(new FileInputStream(directory + fileName));
//      if (scanner.hasNextLine()) scanner.nextLine();  // to remove csv header
        try {
            while (scanner.hasNextLine()) data.add(scanner.nextLine());
        } finally {
            scanner.close();
        }
        return data;
    }
}
```

```java
package system;

import java.io.FileWriter;

public class FileWrite {
    private static final String directory = System.getProperty("user.dir") + "/resources/";
    private static final String SEPARATOR = ",";

    public static void saveBranches(String filename) throws IOException {
        List<String> alw = new ArrayList<String>();

        alw.add("Name,Location,Staff Quota,OperationStatus");
        for (Branch branch : Database.branchList) {
            StringBuilder st = new StringBuilder();

            st.append(branch.getBranchName().trim());
            st.append(SEPARATOR);
            st.append(branch.getBranchLocation().trim());
            st.append(SEPARATOR);
            st.append(branch.getStaffQuota());
            st.append(SEPARATOR);
            st.append(branch.getOperationStatus().name());
            alw.add(st.toString());
        }
        write(filename, alw);
    }
    private static void write(String fileName, List<String> data) throws IOException {
        PrintWriter out = new PrintWriter(new FileWriter(directory + fileName));

        try {
            for (int i = 0; i < data.size(); i++) {
                out.println((String)data.get(i));
            }
        } finally {
            out.close();
        }
    }
}
```

SRP

Single Responsibility Principle

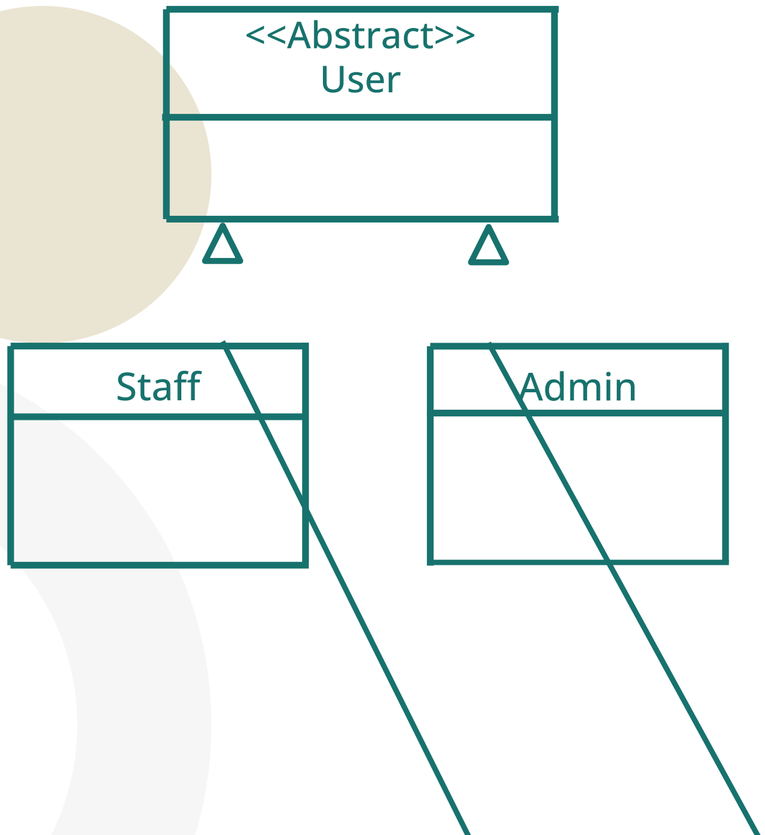


Each class within the ‘customerui’ sub-package is focused on a particular system page or user interaction, adhering to SRP by encapsulating specific user interface functionalities.

This makes it easier to modify individual components without impacting other parts of the system

OCP

Open-Closed Principle



‘Admin’ and ‘Staff’ class extending from the abstract ‘User’ class that has basic set and get functions for common attributes like userId and password

LSP

Liskov-Substitution Principle

ArrayList<User> accountList

```
Admin admin = new Admin(loginId, password, name, Gender.valueOf(gender), age, branch);
accountList.add(admin);
```

```
case 'S':
    Staff staff = new Staff(b, loginId, password, name, g, age);
    accountList.add(staff);
    b.getStaffList().add(staff);
    break;
case 'M':
    Manager manager = new Manager(b, loginId, password, name, g, age);
    accountList.add(manager);
    b.getManagerList().add(manager);
    break;
```

Upcasting to ‘User’ Class to be stored in accountList

```
switch (loginPage.user.getUserType()) {
    case ADMINISTRATOR:
        Admin admin = (Admin) loginPage.user;
        new AdminPage(sc, admin, database);
        break;
    case MANAGER:
        Manager manager = (Manager) loginPage.user;
        new ManagerPage(sc, manager, branchManagement.getBranch(manager.getBranch().getBranchName()));
        break;
    case STAFF:
        Staff staff = (Staff) loginPage.user;
        new StaffPage(sc, staff, branchManagement.getBranch(staff.getBranch().getBranchName()));
        break;
    default:
        System.out.println(x:"Invalid staff type.");
        break;
}
```

Downcasting to specific instance to access specific subclass functionalities

THANK YOU

Q&A