# Modular Software Development 2023

① a) i) In the context of object oriented programming,

- Class - The blueprint for creating objects, defined by attributes and behaviours (methods). Classes allow reusability and modularity in code
- Object Instance - An occurance of a class in memory. Object is created when we instantiate it. It inherits attributes and methods defined within the class

ii)
- constructor is a special type of method invoked automatically when an object is instantiated from a class. It sets the state of the object by assigning initial attributes.
- Method is the behaviour of a class that object can execute. Defined in a block of code It manipulates object's state and provides functionality.

iii) since this is already defined in the code, the error will be identified at compile time. This is because type checking is done at compile stage.

The reason for the error is attempting to add an integer with the string concatenation. This results in a string, however Java cannot assign it to the Integer variable "re".

These types of mismatch errors are reported preventing assignment of values between incompatible types. This catches errors early in the running process to enhance code reliability.

b) i) Inheritance is a fundamental OOP concept that allows a class (sub class) to inherit attributes and methods from another class (super class) There is a hierachy enabling code reusabiliy and plays a major role in the mailroom scenario.

Message is the base class representing all forms of communication. It has common getters as defined. It extends into sub classes Letter, Email, Fax, etc. These sub classes inherit methods such as above methods as well as overide methods like "getDestination()" from Letter sub class.

The inheritance alls to filter specific types of messages as instances of Message class while preserving their specific behaviour. Thus inheritance allows to handle various types of messages in the mailroom. Allowing code reusability and extensible design.

ii)
```
class Message {
    private String content;
    private String source;
    private String destination;

    //constructor
    public Message (String content) { this.content = content;}

    //setter
    public void setDestination (String destination) {this.destination = destination;}

    //Getters
    public String getSource () { return source;}

    public string getContent () {return content;}

    public String getDestination () {return destination}   //Overidable

    public String getFullDestination () {return destination}
}
```

iii)

```
class Letter extends Message{
    public Letter (String content) {      //constructor
        super (content);
    }

    @Override
    public String getDestination () {
        //Assuming destination for a Leter is the postal Address
        return (super. get Destination()) .split (",") [0];
    }
}
```

② a) i)

```
interface Deliverable {
    String getDestinationAddress ();
    String getOriginAddress();
}
```

ii)

```
class DeliverableLetter extends Letter implements Deliverable {
    public DelivarbleLetter (String content){      //Constructor
        super (content);
    }

    //Implementing methods from deliverable interface
    @Override
    public string getDestination Address () {
        return (super.getDestinatien()). split (",") [0];
    }

    @Override
    public String getOriginAddress () {
        return (super.getSource ()). split (",") [0];
    }
```

b) i) Theoretical Concept

The proposal to extend both 'sender' and 'Receiver' classes to create the 'MailOperator' class involves multiple inheritance. This allows a class to inherit behaviours and attributes from multiple parent classes. How ever the case is different when practically implementing it in Java.

Practical complexity

Java doesn't support multiple inheritance. As it leads to complexity and ambiguity where child could call a method implemented in both parent classes similarly. This is known as the diamond problem in Java and it faces difficulty in prioratizing. If 'sender' and 'Receiver' share somewhat common functionality, duplicating code in both to a sub class can cause code Redundancy. Furthermore it could cause design complexity.

ii) A more effective method is to use interfaces. we can define common behaviours in interfaces ('sendabe', 'Receivable') and provide functionality in the 'MailOperator' class. This solves the multiple inheritance problem in Java.

The advantage of this approach is solving the 'Diamond Problem' to avoid limitations and complexity. Furthermore, this provides flexibility as new types of message handling ('Fax Sender', 'EmailReceiver') can be added easily by implementing respective interfaces.

c) Polymorphism allows methods to be invoked on objects of different classes in a consistent manner. Method overloading enables creation of multiple methods with same name but different parameters within a class. Method overriding involves redefining a method in a subclass with a specific implementation Polymorphism helps minimize redundancy by promoting code reuse and flexibility in handling different object types through a unified interface. Common functionalities in one place reduces replication of code and inconsistency.

(3) a) Software engineers today are crucial in meeting industry needs by developing customized solutions, driving innovation, and ensuring system scalability. Simultaneously, they address people's needs through user-centric design, enhanced communication and software tailored applications.

Ethical considerations guide their decisions, focusing privacy, security and user well-being. By balancing industry demands with user recquirements, software engineers play major role in present context.

b) A software process is a structured set of activities in software development to ensure systematic and efficient production of systems. In the software lifecycle, they play a vital role as seen in the stages below.

During software specification, requirements are gathered and defined to establish scope. In the development phase, software is designed and programmed as per requirements. Next validation involves checking the software to meet customer needs. Finally in the evolution stage, software is modified and updated to customer needs.

An example is the agile methodology. It focuses on iterative development, collaboration and adaptability. Teams can deliver incremental value to users responding to changing recquirements.

c) i) Software Prototyping

This is a development approach where simplified version of the system is created to demonstrate key features and gather feedback early in the process. It helps stakeholders visualize final product and refine requirements.

ii) Software Testing

This is the process of evaluating software application to identify defects and ensure it meets quality standards. It involves executing the software, ver fying its behaviour and validating functions as expected

iii) Batch Processing Systems

These are systems that process large amounts of data and at scheduled intervals as batches. suitable to efficiently handle large volumes of data where real time processing is not recquired.

eg Payroll Processing systems

iv) Embedded Control Systems.

These are specialized computing systems integrated into hardware to manage and control specific function.

eg Traffic light controller

(ii) a) customer - End users interacting directly with system

Drivers - stakeholder that uses the system

Management team - Operators of the service in the company.

b) Functional Requirement - Booking management

- The system must allow customers to book taxis through a user friendly interface and customized detail entry.
- This recquirement is essential to support the core functionality of the system - to enable customers to make bookings efficiently and accurately.
- Validate by testing system's ability to accept bookings, validate input data and confirm bookings through automation.

No: ______ Date: __/__/____

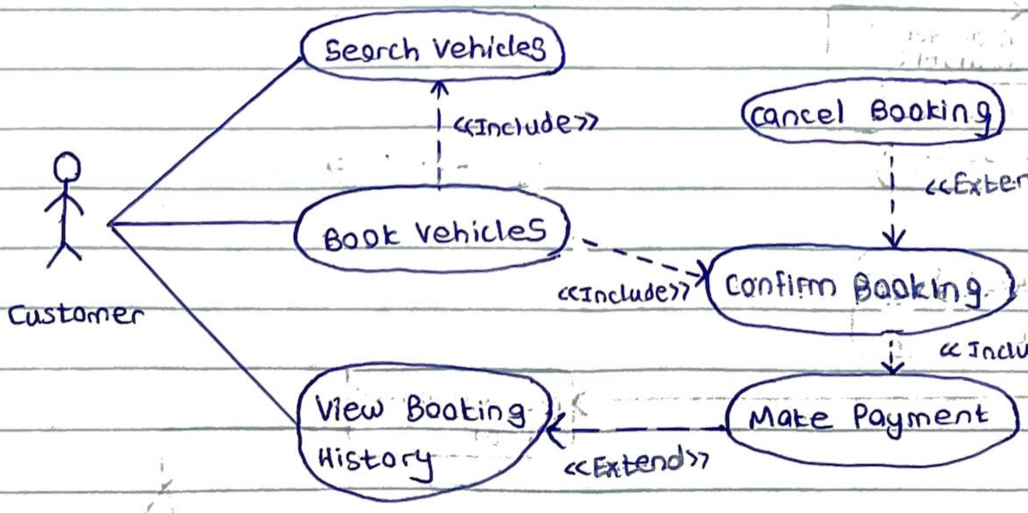
Non-Functional Recquirement - Performance

- The system must respond to booking requests quickly on average even during peak hours.
- This is recquired as it directly impacts user satifaction and operational efficiency. Deloys can lead to frustration and loss of business.
- Validate by performance testing with load testing tools to simulate peak loads and measure system response times. compare with the requirements.

Domain Requirement → Data Privacy and Security.

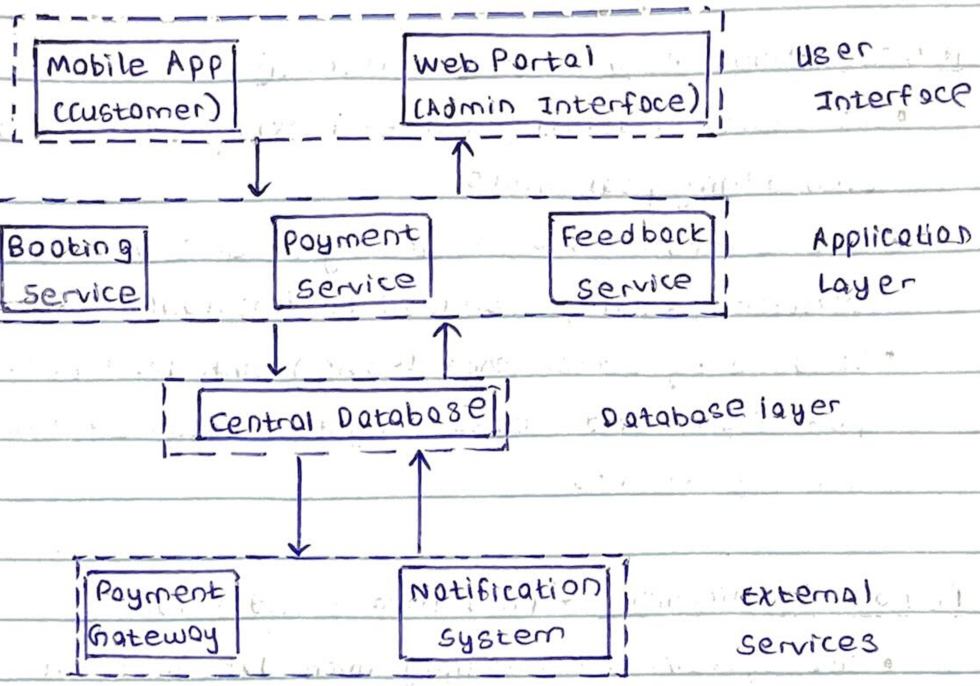
- The system must comply with data protection regulations to ensure privacy and data security.
- This is required to maintain customer trust and complionce with legal authority; to not damage company reputation.
- Validate by conducting regular security audits and vulnerability assesments. Implement recquired protocols and ensure adherence.

c)

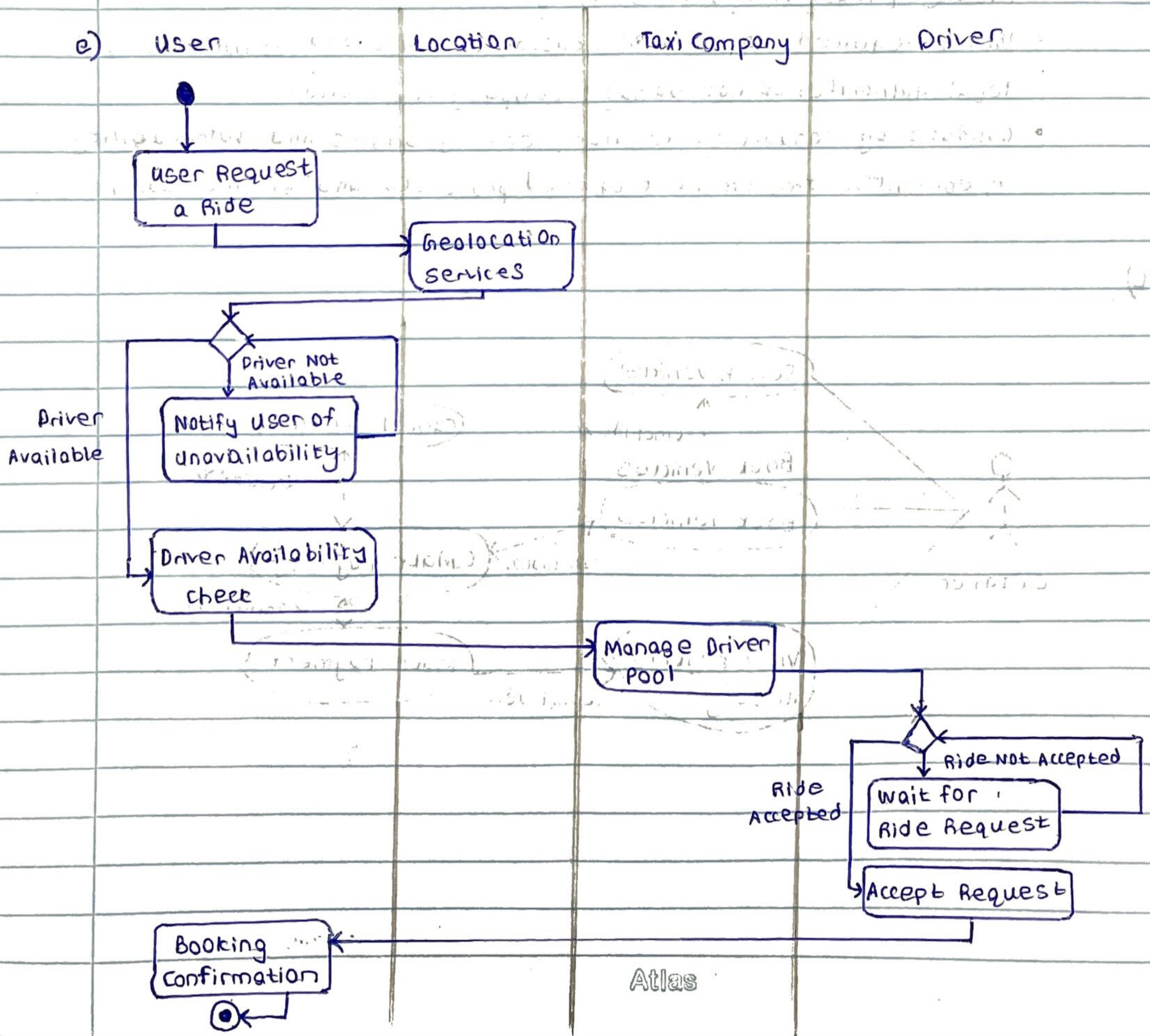


No: ________ Date: __/__/____

d)


Mobile App (Customer)
Web Portal (Admin Interface)
User Interface
Booking Service
Payment Service
Feedback Service
Application Layer
Central Database
Database layer
Payment Gateway
Notification System
External Services

e)


User
Location
Taxi Company
Driver
User Request a Ride
Geolocation Services
Driver Not Available
Notify User of unavailability
Driver Available
Driver Availability check
Manage Driver Pool
Ride Not Accepted
Wait for Ride Request
Ride Accepted
Accept Request
Booking Confirmation

Assumptions

- user has already logged in the system by default.
- Flow passes through following mentioned threads in the swimlane vertical activity diagram
- System process ends only when taxis is reserved. Meaning Booking is confirmed.
- System attempts to connect drivers to users and repeatedly checks driver availability.

f) using Agile software Development process

- Agile allows iterative development and assesment of requirements
- Promotes close collaboration with stakeholders
- Emphasizes regulor testing and quality assuarance
- Deliver functional components incrementally.