**ATM Simulator**

Final Report

Submitted in Partial Fulfilment of the Requirements for

the Award of Degree of

**Bachelor in Technology**



**Submitted by:**

<your name><your roll number>

CERTIFICATE

This is to certify that the work presented in this Project entitled “ATM Simulator” is a bonafide record of the work done during the period from Jan – June 2024 at “FUTURE FINDERS” by <your name>

The project work is an authentic record of my own work and is carried out under the supervision and guidance of Guide <GUIDE NAME>, <X> Department. The matter presented in the report has not been submitted elsewhere, wholly or in part, for the award of any other degree or diploma.

Your name(19xxxxx)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

<Guide name>

**Guide Name**

Department of <X>

<HOD NAME>

**HOD <X>**

<X> Department <College name>

ACKNOWLEDGEMENT

I take this opportunity to express my sincere gratitude to the Principal, <XYZ>, for providing this opportunity

to carry out the present work.

The constant guidance and encouragement received from Prof. (Dr.) <X>, Professor and Head, Department

of <XYZ> Engineering, has been of great help in carrying our present work and helped us in completing

this project with success.

I would like to express a deep sense of gratitude to “FUTURE FINDERS” team and my Project Guide Prof.

<X>, <XYZ> department for the guidance and support in defining the design problem and towards the

completion of my project work. Without their wise counsel and able guidance, it would have been impossible

to complete the thesis in this manner.

I am also thankful to all the faculty and staff members of FUTURE FINDERS ORGANISATION for their

intellectual support throughout the course of this work.

<Your name>(19xxxxx)

**ABSTRACT**

The ATM Simulator project in Java aimed to design, implement, and evaluate a realistic simulation of an Automated Teller Machine (ATM) system. This report presents a comprehensive overview of the project, detailing its objectives, methodology, implementation process, and outcomes.

The project's primary objectives included developing a user-friendly ATM interface with essential functionalities such as cash withdrawal, deposit, balance inquiry, and account management. Additionally, robust backend systems were implemented to handle transaction processing, account management, and security features. A realistic simulation environment was created to mimic the behavior of a real ATM machine. The project also aimed to evaluate the performance, usability, and security aspects of the developed simulator.

To achieve these objectives, the project followed a structured methodology comprising several phases. Initially, a thorough requirement analysis was conducted to gather and analyze the functional and non-functional requirements of the ATM system. Subsequently, detailed design specifications were created for both the frontend user interface and backend systems. The implementation phase involved writing modular and scalable code in Java programming language to realize the design specifications. Rigorous testing, including unit tests, integration tests, and user acceptance tests, was conducted to ensure the correctness and reliability of the simulator. Finally, the simulator's performance, usability, and security were evaluated through various metrics and user feedback.

During the implementation phase, object-oriented programming principles were utilized, along with the Java Swing framework for the graphical user interface, and data structures for efficient data management. Security measures, such as authentication, encryption, and transaction validation, were integrated to safeguard user transactions.

The evaluation phase included usability testing with potential users to gather feedback on the interface design and overall user experience. Performance testing was conducted to assess the simulator's responsiveness and efficiency under different usage scenarios. Security testing was also performed to identify and mitigate potential vulnerabilities in the system.

**Table of Contents**

Certificate

Acknowledgement

Abstract

Chapter 1 Introduction

1.1 Introduction Profiles

1.1.1 Project Profile

1.1.2 Company Profile

1.1.3 About Course

1.2 Introduction of Project

1.3 Problem Statement

1.4 Project Objective

1.5 Feasibility Study

Chapter 2 Literature Review

Chapter 3 Methodology

Chapter 4 Project Setup

Chapter 5 Implementation

Chapter 6 Screenshots of the project

Chapter 7 Conclusions and Future Work

CHAPTER 1 – INTRODUCTION

|  |  |
| --- | --- |
| **Title** | ATM Simulator |
| Organization | Future Finders Organisation |
| Category | JAVA Application |
| Duration | 6 Months |
| Guide | <guide name> (Internal Guide) |
| Submitted by | <your name>  Roll no. : <your roll number> |
| Submitted to | Department of CSE, <College Name> |



ABOUT US

# Future Finders is a ground-breaking platform that develops young Indian talent that is motivated to advance and forge successful careers in IT. We provide a variety of courses to help you launch your career and locate the employer that will assist you as you rise to the top. Our expertise with the latest tools and techniques, and the experience of our professional experts help us deliver high-end services to our esteemed clients. Future Finders educates students and developers about the most recent technologies that are now popular. We are working on distributing informational know-how and offering clients specialized services following global best practices. At Future Finders, we recognize your unrivalled skill and help you realize your creative ideas. Our sole goal is to provide students with cutting-edge practical skills that will enable them to swiftly and effectively adapt to the constantly evolving technologies found in the business world. At Future Finders, our goal is to raise educational standards via innovation in both quality and practical knowledge.



**MISSION**

* At Future Finders, our goal is to improve quality and practical skills while raising the bar for education. Future Finders' principal objective is to bridge the knowledge gap between what is being taught in schools and what the industry needs.

**VISION**

* Future Finders' vision is to provide students with cutting-edge practical skills so they can easily handle and swiftly acclimate to the constantly evolving technology in the corporate world.

**PHILOSOPHY**

# To impart hardcore practical quality training among students/developers about latest technologies trending today.

To share knowledge of information security and create awareness in the market. The solution to clients' as per the International standard practices and governance.

# To support good business practices through continual employee training and education

To equip a local team with a strong knowledge of international best practices and international expert support so as to provide practical advisories in the best interests of our clients

* JAVA
* FULL STACK
* C LANGUAGE
* C++ LANGUAGE
* CYBER SECURITY
* MERN STACK
* PYTHON
* DIGITAL MARKETING
* PHP
* NETWORKING
* ANDROID
* IOS
* .NET
* ORADE DBA
* SOFTWARE TESTING
* ETHICAL HACKING
* MACHINE LEARNING
* BIG DATA HADOOP
* NODE JS
* ANGULAR JS
* CLOUD COMPUTING
* AUTO CAD
* NETWORKING
* ROBOTICS
* VLSI – VHDL
* PCB
* MATLAB
* EMBEDDED SYSTEM
* IOT
* PLC / SCADE
* ARDUINO
  + NX- CAM
  + AUTOCAD ME
  + SOLID WORKS
  + CATIA
  + CREO
  + CNC PROGRAMMING
  + ANSYS
    - MARKETING
    - DIGITAL MARKETING
    - FINANCE
    - HR MANAGEMENT
  + AUTOCAD CE
  + PRIMA VERA
  + MX ROAD
  + 3DS MAX
  + REVIT ARCHITECTURE
  + STAAD PRO

CSE & IT

ECE & EE

MECHANICAL

CIVIL

MANAGEMENT

Why choose us:

 Experience

 Goal Oriented

 A Streamlined / Quality-Driven Process  Talented Designers & Expert Developers

 Our Websites & E-marketing Platforms are Easy to Manage  We Are Dedicated to Our Client’s Success

## We focus on imparting practical skills to the trainees & not just theoretical knowledge. The courses are designed in this way at FUTURE FINDERS correspond to the standards of the corporate divisions and industries. Only through the acquisition of practical skills you can handle the everlasting technologies that venture out in real-time situations**.**

* At **FUTURE FINDERS,** Future Finders has committed staff members with organised learning curricula that can assist you in beginning your career in the most cutting-edge and successful industries.

* **Quality of the Product:** Our software service sector has been maintaining the highest international standards of quality.

## * **Live projects:** Working on active projects and working at a job aid in experiencing growth. In keeping with this, we assist you in developing greater self-assurance, which helps you succeed as quickly as feasible.

* **Global certification:** The courses that Future Finders offers make sure that you are certified and eligible to grab the best opportunities in your career.

## * **Partnership:** FUTURE FINDERS, considers every client a partner. From the initial stages, you are closely involved into the procedure of technical classification, development, and testing.

* **Jobs & Career Prospective:** Future Finders views education and learning as investments in oneself. Consequently, the paths we have set forth ensure that your investment in us is a success



* In addition to a panel of eminent consultants and advisors, we have a dedicated pool of trained Developers and Trainer, investigators, working under the guidance of professional managers. **“A Ship is as good as the crew who sail her**.**”** Our Technical team of professionals handing, designing & delivering of projects has a strong presence in the North India & the US. Our engineers are already working on the latest technologies like **I-Phone & Android** Applications, **Robotics**, **VLSI-VHDL**, Embedded System, Networking and **Cloud computing.** Some of the key professionals and advisors are listed

### Mr. Bonish Singla: (Director)



* He is the backbone of FUTURE FINDERS, manage the company’s day to day affairs and a man with more than 9 years rich practical experience who believes in taking up new ventures and projects. He has been awarded many times for his exemplary work in process improvement for IT Service Delivery Domains. MASTERS in Computer applications and Certified from CU Certification. Holds total of 9 Years of rich experience including 5 Years in Information Security Implementation, Maintenance and Auditing and initial over 4 years of experience in Project Management, Client Relationship Management and Server, Desktop, and IT Service Delivery web designing.

### 

### Miss. Harjit Kaur :(Branch Manager)



* She has more than 5 years solid industrial experience in software companies and she is very innovative in her technical approach. She has completed her masters in MBA. She takes all the responsibilities and maintains staff by recruiting , selecting , orienting , and training employees and Accomplishes staff results by communicating job expectations , planning ,monitoring , and appraising job results.

### Miss. Harsimran: (HR)



* Human resources (HR) are the division of a business responsible for finding, recruiting, screening, and training job applicants. MBA in HR and marketing from (CU) Total of 5+ Years of rich experience HR departments also handle employee compensation, benefits, and terminations. HR departments must keep up to date with laws that can affect the company and its employees. She also assists with payroll management so employees receive their paychecks on time.

### 

### Miss. Isha Bala: (Technical Head)



* A technical lead, or tech lead, oversees the technical aspects of a software team. M. tech and diploma in (CSE) . She helps making architectural and design decisions, guiding team members, and supervising system modifications. Identify potential risk and forming contingency plan as fast as possible. Efficiently liaise with the team members, clients , and also the management .

### Miss. Nihirika: (Head Counselor)

### 

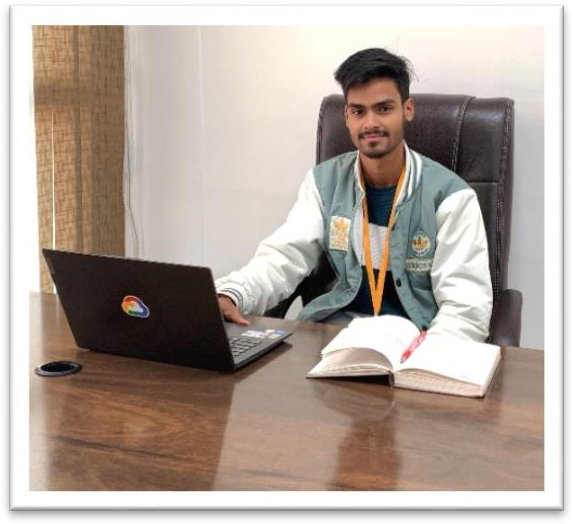
* Head Counselor, under the general direction of the Head - Sales and Marketing, provides leadership and direction to the Counseling Department and assumes responsibilities in developing, implementing, and evaluating the Company counseling and guidance program that includes academic, career, personal/social development. She completed her degree in B .Tech

### 

### Mr. Jaspal Singh: (Civil &Mechanical Head)

* He is leader of the team of civil, mechanical, and electrical engineers and responsible for the planning and analysis of the aspect of the construction that involves mechanical works . He has more than 37 years of experience in industrial field. He is providing services as a technical trainer for more than 8 years. He did his B. Tech in Mechanical Engineering from PEC (Punjab Engineering College). He has been awarded many times for his brilliant services.

### Mr. Chetan Kalra: (Digital Marketing Head)



* B.tech (CSE) – IKG-PTU, Experienced digital marketing manager with extensive experience building, maintaining, and running successful digital marketing campaigns from past 4 years. Bringing forth broad marketing knowledge, coupled with focused campaign experience. Adept at creating and implementing client- centered, successful campaigns, aimed at improving brand awareness and presence. Collaborative and creative manager accomplished at managing digital marketing presence content. Experienced in leading teams of marketing professionals to meet and exceed digital marketing goals.



Mr. Shivam: (Java Developer)

### 

Having 5+ years of experience in analysis, design, development, testing & implementation of complex software applications. B. Tech in CSE (CGC College) Experience and involvement in designing, implementing, and evaluating end-to-end systems using several Java frameworks and technologies like J2EE.

### Miss. Archana: (Full Stack Developer)



* Full Stack Developer with 6+ years of hands-on experience designing, developing, and implementing applications and solutions using a range of technologies and programming languages. B.tech (CSE) PU Certification Seeking to leverage broad development experience and hands-on technical

### 



Plot NO F-465, K&B Tower, Second Floor, Phase 8B, Mohali I (+91) 98559-08009 I [www.futurefinders.in](http://www.futurefinders.in/)

### Mr. Ashwani: (Automation Executive)

### 

* (EE) – CGC College Automation engineer with 4+ years of experience in a variety of industries. Passionate for developing and implementing process improvements through the use of robotics, PLCs, and HMIs. Demonstrated ability to lead cross-functional teams in the design, development, and deployment of manufacturing and process automation solutions.

### Mr. Ayush: (PHP Developer)



* PHP developer to manage our back-end services and ensure a seamless interchange of data between the server and our users. Bachelor's degree in computer programming, development and Certified from CU Certification PHP developer, responsible for developing and coding all server-side logic and required to maintain the central database and respond to requests from front-end developers.

**About the course**

**MODULE 1:**

**History of Android**

Android is a mobile operating system (OS) based on the Linux kernel, designed primarily for touchscreen mobile devices such as smartphones and tablets. It was developed by Google and the Open Handset Alliance (OHA), a consortium of hardware, software, and telecommunication companies.

Android was first announced in November 2007 and the first commercial Android device, the HTC Dream, was released in September 2008. Initially, Android was designed to be a competitor to other mobile operating systems such as Symbian, Blackberry OS, and Windows Mobile. However, Android quickly became the most popular mobile OS in the world, with a market share of over 85% in 2021.

One of the key features of Android is its open-source nature, which allows developers to customize the OS and create their own applications. The Android SDK (software development kit) provides developers with the tools and APIs (application programming interfaces) necessary to build Android applications.

Android applications are written in Java programming language and run on the Dalvik virtual machine, which is designed specifically for mobile devices with limited memory and processing power. Android also supports other programming languages such as Kotlin and C++, and provides support for native code through the Android NDK (native development kit).

Android has evolved significantly since its initial release, with major updates and new features being added regularly. Some of the notable releases include:

* Android 1.5 Cupcake (2009): introduced support for third-party keyboards, video recording, and automatic screen rotation
* Android 2.2 Froyo (2010): introduced support for Adobe Flash, improved performance, and new enterprise features
* Android 4.0 Ice Cream Sandwich (2011): introduced a new user interface design, improved multitasking, and support for NFC (near field communication)
* Android 5.0 Lollipop (2014): introduced the Material Design language, improved battery life, and new security features
* Android 6.0 Marshmallow (2015): introduced improved app permissions, Google Now on Tap, and a new battery-saving feature called Doze
* Android 7.0 Nougat (2016): introduced support for split-screen multitasking, improved notifications, and a new virtual reality platform called Daydream
* Android 8.0 Oreo (2017): introduced picture-in-picture mode, improved battery life, and new notification channels
* Android 9.0 Pie (2018): introduced gesture navigation, adaptive battery, and a new feature called Digital Wellbeing, which helps users manage their screen time
* Android 10 (2019): introduced a new system-wide dark mode, improved privacy and security features, and support for foldable devices
* Android 11 (2020): introduced improvements to messaging, privacy controls, and media controls

Android 12 (2021): introduced a new design language called Material You, improved privacy features, and new notification features.

In addition to smartphones and tablets, Android has also been used in other devices such as smartwatches, smart TVs, and even cars. Android's open-source nature and large developer community have contributed to its success, making it one of the most widely-used operating systems in the world.

Android Studio is an Integrated Development Environment (IDE) specifically designed for developing Android applications. It is based on JetBrains' IntelliJ IDEA software and was first released in 2013. Since then, it has become the standard development environment for Android app development.

Android Studio is a powerful tool that provides a wide range of features to help developers create high-quality Android applications. It supports a wide range of programming languages, including Java and Kotlin, and provides a range of tools to help developers test and debug their applications. Here are some of the key features of Android Studio:

Gradle-based build system: Android Studio uses the Gradle build system, which is highly customizable and provides a flexible build process. This helps developers to create and maintain complex projects with ease.

Code templates: Android Studio provides a range of code templates that developers can use to create their Android applications. These templates can help developers save time by providing pre-written code for common tasks, such as creating a new activity or fragment.

Layout editor: Android Studio's layout editor is a powerful tool that allows developers to create complex user interfaces visually. It provides a drag-and-drop interface that allows developers to add, remove, and modify UI components quickly and easily.

Code analysis: Android Studio includes a range of code analysis tools that can help developers identify potential bugs and performance issues in their code. This can help to improve the overall quality of the application and reduce the time spent on debugging.

Emulator: Android Studio includes an emulator that allows developers to test their applications on a virtual device. This is useful for testing the application on different screen sizes, resolutions, and versions of Android.

Integration with other tools: Android Studio integrates with a range of other tools, including Git, Jenkins, and JUnit. This can help developers to automate their build and test processes and streamline their workflow.

Android Studio provides a wide range of features that can help developers to create high-quality Android applications quickly and efficiently. Here are some of the key components of Android Studio:

Android SDK Manager: The Android SDK Manager is a tool that allows developers to download and manage the Android SDK, which includes the Android API, emulator images, and development tools.

Android Virtual Device (AVD) Manager: The AVD Manager allows developers to create and manage virtual devices that can be used to test their applications.

Gradle Build System: Gradle is a build system that is used to automate the building, testing, and deployment of Android applications.

Layout Editor: The Layout Editor is a visual editor that allows developers to create and modify user interfaces for their Android applications.

Code Editor: The Code Editor is a powerful tool that provides syntax highlighting, code completion, and other features to help developers write high-quality code.

Debugging and Profiling Tools: Android Studio provides a range of debugging and profiling tools that can help developers to identify and fix issues in their code.

Android Emulator: The Android Emulator is a virtual device that allows developers to test their applications on different versions of Android and different screen sizes.

Project Structure: The Project Structure dialog allows developers to configure various aspects of their Android projects, including dependencies, build types, and flavors.

Overall, Android Studio is a powerful tool that can help developers to create high-quality Android applications quickly and efficiently. Its wide range of features and tools make it an ideal choice for both experienced and novice developers alike.

**MODULE 2:**

**INTRODUCTION TO JAVA:**

Java is a popular programming language that was originally developed by Sun Microsystems (now owned by Oracle Corporation) in the mid-1990s. It is designed to be platform-independent, meaning that Java code can run on any system that has a Java Virtual Machine (JVM) installed.

Java is an object-oriented language, meaning that it is based on the concept of objects, which are instances of classes that contain data and code. The language is known for its simplicity, readability, and ease of use. It is also widely used for developing desktop and mobile applications, as well as server-side web applications.

Some of the key features of Java include:

Platform independence: As mentioned earlier, Java is designed to be platform-independent, meaning that code written in Java can run on any system that has a JVM installed.

Object-oriented programming: Java is based on the concept of objects, which are instances of classes that contain data and code. This makes it easier to write reusable and modular code.

Memory management: Java has automatic memory management, meaning that the programmer does not need to manually allocate and deallocate memory.

Robustness and security: Java is known for its robustness and security features, such as its exception handling and automatic garbage collection.

Multi-threading: Java supports multi-threading, meaning that a Java program can have multiple threads of execution running concurrently.

To develop Java applications, programmers use an integrated development environment (IDE) such as Eclipse or IntelliJ IDEA, or a text editor such as Notepad++. They write Java code using syntax and semantics specified by the Java language specification, and compile it into bytecode, which can be executed by the JVM.

Overall, Java is a versatile and widely used programming language that is suitable for a wide range of applications, from desktop and mobile applications to server-side web applications. Its simplicity, readability, and ease of use make it a popular choice among developers, while its platform independence and security features make it a popular choice among businesses and organizations.



**OBJECTS AND CLASSES:** In Java, an object is an instance of a class. A class is a blueprint or template that describes the characteristics and behavior of a group of objects. Objects have states and behaviors. State refers to the values of the object's attributes or properties, while behavior refers to the actions or operations that can be performed on the object.

To create an object in Java, you first need to define a class. A class definition includes the class name, the data fields or instance variables, and the methods or functions that operate on those variables. Here's an example of a simple class definition:

public class Person {

String name;

int age;

public void sayHello() {

System.out.println("Hello, my name is " + name + " and I am " + age + " years old.");

}

}

This class represents a person, with a name and an age. It also has a method sayHello() that prints out a greeting using the person's name and age. To create an object of this class, you use the new keyword:

Person john = new Person();

This creates a new Person object and assigns it to the variable john. You can then set the person's name and age:

john.name = "John";

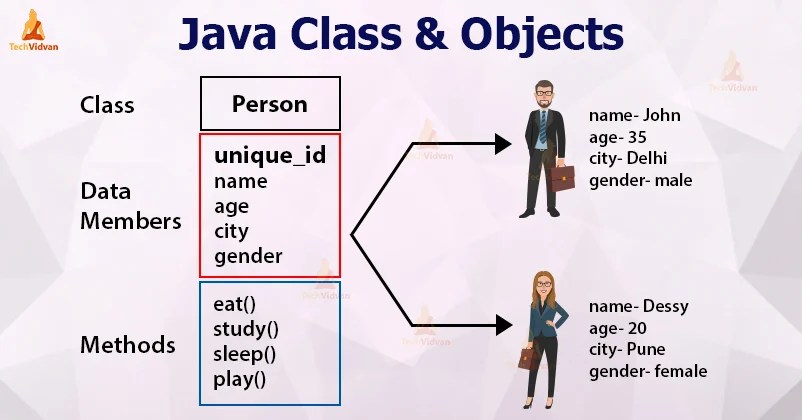
john.age = 30;

And you can call the sayHello() method on the john object:

john.sayHello();

This will print out the greeting "Hello, my name is John and I am 30 years old."

Classes in Java can also inherit from other classes, allowing you to create hierarchies of classes with increasing levels of specialization. This is known as inheritance, and it allows you to reuse code and build on existing functionality. For example, you might have a Student class that inherits from the Person class, adding properties like a student ID number and a list of courses they are enrolled in.



In summary, objects and classes are the building blocks of object-oriented programming in Java. A class defines a blueprint for a group of objects, and an object is an instance of a class. State refers to the values of the object's properties, and behavior refers to the actions or operations that can be performed on the object.

Java is an object-oriented programming (OOP) language, which means that it uses the concept of objects and classes to organize code.

An object is an instance of a class, which is a blueprint for creating objects.

In Java, a class is a template that defines the properties and methods of a particular type of object. It describes the attributes and behaviors of objects of that class.

An object is created from a class using the "new" keyword followed by the name of the class.

Objects in Java have state (properties) and behavior (methods). The state of an object is defined by its instance variables, which are declared in the class. The behavior of an object is defined by the methods, which are also declared in the class.

In Java, classes are defined using the "class" keyword followed by the name of the class. The name of the class should start with a capital letter and should be in CamelCase format.

A Java class can contain instance variables, constructors, methods, and nested classes or interfaces.

Instance variables are the properties of an object, and they are declared within a class. They can have different access modifiers such as public, private, or protected.

Constructors are special methods that are used to create and initialize objects. They have the same name as the class and do not have a return type.

Methods are functions that perform some action on the object. They have a return type and can accept parameters.

Access modifiers such as public, private, and protected are used to restrict access to instance variables and methods.

In Java, the "this" keyword is used to refer to the current object. It is used to access the instance variables and methods of the object.

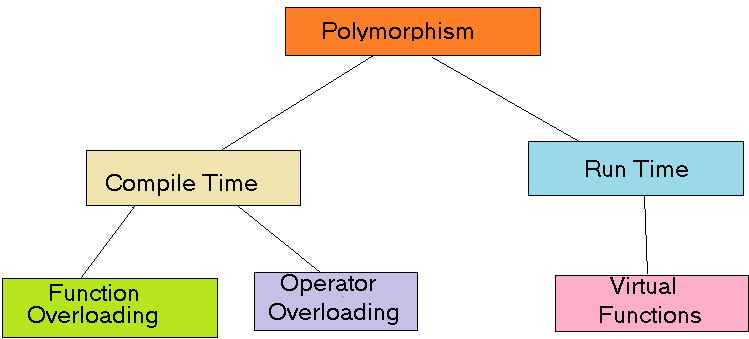
The "static" keyword is used to define a variable or method that belongs to the class rather than an instance of the class.

Inheritance is a mechanism in Java that allows a class to inherit properties and methods from another class. The class that is being inherited from is called the superclass or parent class, and the class that is inheriting is called the subclass or child class.

Polymorphism is a feature in Java that allows objects of different classes to be treated as if they were of the same class. This can be achieved through method overloading or method overriding.

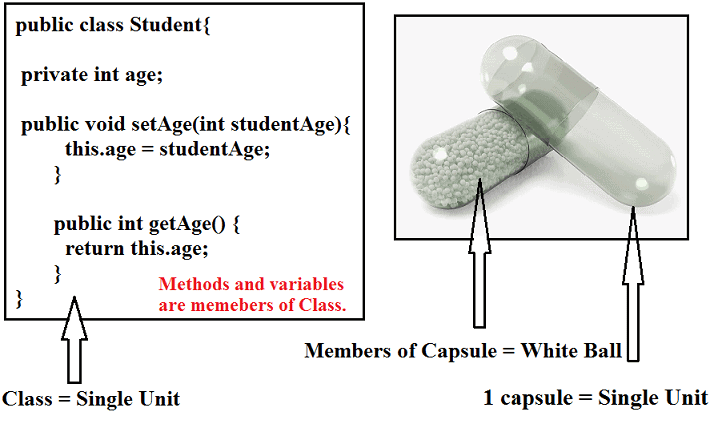
Method overloading is a technique in Java where multiple methods can have the same name but different parameters.

Method overriding is a technique in Java where a subclass provides its implementation of a method that is already defined in its superclass.



Encapsulation is the practice of hiding the implementation details of a class from the outside world. This is achieved by declaring the instance variables as private and providing public getter and setter methods to access and modify the variables.

These are some of the important concepts related to objects and classes in Java. Understanding these concepts is crucial for developing Java applications.



MODULE 3:

INHERITANCE:

Inheritance is an important concept in object-oriented programming (OOP) that allows a class to inherit properties and methods from another class. The class that inherits from another class is called the subclass, while the class being inherited from is called the superclass.

In Java, inheritance is implemented using the extends keyword. Here's an example:

csharp

Copy code

public class Animal {

protected String name;

protected int age;

public Animal(String name, int age) {

this.name = name;

this.age = age;

}

public void makeSound() {

System.out.println("The animal makes a sound");

}

}

public class Dog extends Animal {

public Dog(String name, int age) {

super(name, age);

}

@Override

public void makeSound() {

System.out.println("The dog barks");

}

}

In this example, Dog is a subclass of Animal, which is the superclass. Dog inherits the name and age properties and the makeSound() method from Animal. The @Override annotation indicates that Dog is providing its own implementation of the makeSound() method.

Some key points to keep in mind about inheritance in Java:

A subclass can only inherit from one superclass.

The protected keyword is used to make class members (such as variables and methods) accessible to subclasses.

Constructors are not inherited, but a subclass can call a constructor of the superclass using the super() keyword.

If a subclass provides its own implementation of a method from the superclass, it can use the @Override annotation to indicate this.

A subclass can add its own properties and methods in addition to those inherited from the superclass.

Inheritance is a powerful tool for creating classes that share common properties and behaviors. It allows you to create a hierarchy of related classes that can be organized and managed more easily. By using inheritance in your Java code, you can write more efficient and maintainable software.

File handling is an important concept in Java, as it allows the user to interact with the files stored on the computer's file system. There are several classes available in Java that can be used for file handling, such as FileReader, FileWriter, BufferedReader, BufferedWriter, etc.



To read a file in Java, we can use the FileReader and BufferedReader classes. Here is an example code snippet that reads a text file named "example.txt":

java

Copy code

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class ReadFileExample {

public static void main(String[] args) {

BufferedReader reader = null;

try {

reader = new BufferedReader(new FileReader("example.txt"));

String line = reader.readLine();

while (line != null) {

System.out.println(line);

line = reader.readLine();

}

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

To write to a file in Java, we can use the FileWriter and BufferedWriter classes. Here is an example code snippet that writes some text to a text file named "example.txt":

java

Copy code

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

public class WriteFileExample {

public static void main(String[] args) {

BufferedWriter writer = null;

try {

writer = new BufferedWriter(new FileWriter("example.txt"));

writer.write("This is an example text file.");

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

writer.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

These are just some basic examples of file handling in Java. There are many more advanced concepts and techniques that can be used to handle files in Java, such as working with binary files, using serialization to save and load objects, and more.

Android Studio is an integrated development environment (IDE) used to develop Android applications. It is based on the IntelliJ IDEA platform and provides a comprehensive set of tools and features for building Android apps. The Android SDK (Software Development Kit) is a collection of libraries, tools, and documentation that developers use to create Android apps.

Here are some key features and components of Android Studio and the Android SDK:

User Interface: Android Studio provides an intuitive user interface that makes it easy to create, modify, and test your app. You can use drag-and-drop tools to create your UI and preview changes in real-time.

Code Editor: Android Studio comes with a powerful code editor that supports many programming languages, including Java and Kotlin. You can use code completion, syntax highlighting, and other features to speed up your coding.

Build Tools: Android Studio uses Gradle as its build system. Gradle is a powerful and flexible tool that makes it easy to manage dependencies, build flavors, and create signed APKs.

Emulator: Android Studio provides an emulator that allows you to test your app on a virtual device. You can customize the device's properties and emulate different hardware configurations to ensure your app works on a wide range of devices.

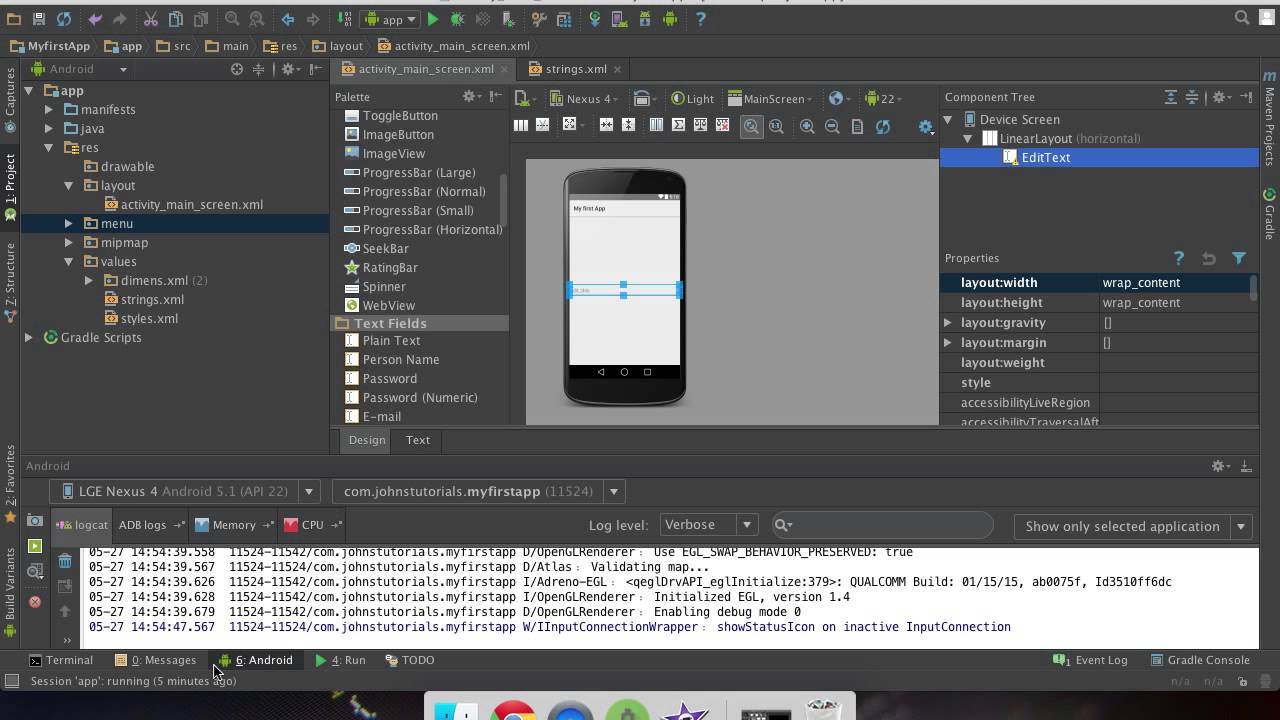
Debugger: Android Studio includes a debugger that helps you identify and fix issues in your code. You can set breakpoints, inspect variables, and step through code to understand how your app works.

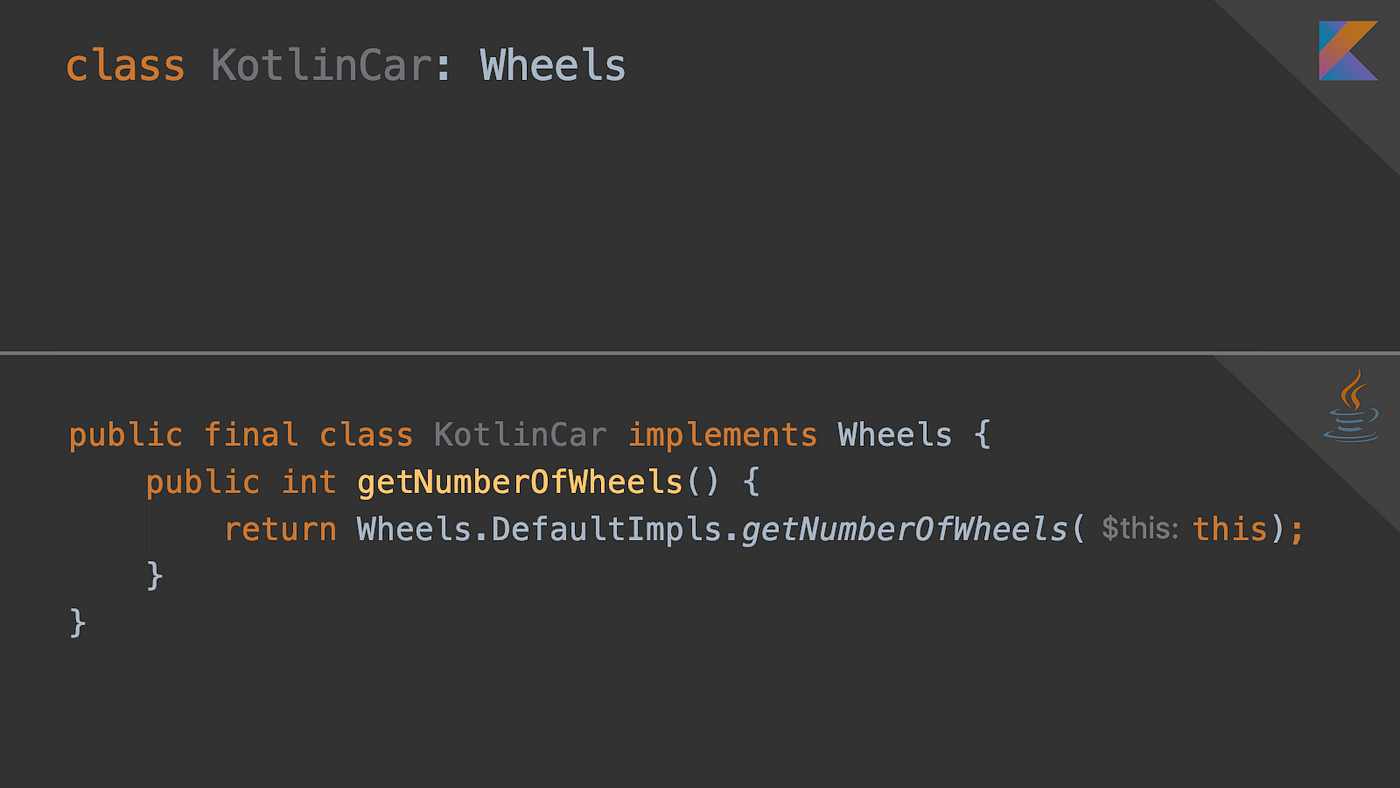
Android SDK: The Android SDK includes a set of tools and libraries that you can use to develop Android apps. It provides APIs for accessing hardware features, such as the camera and sensors, and for building user interfaces with widgets and layouts.

Libraries: The Android SDK includes many libraries that you can use to add functionality to your app. For example, you can use the Retrofit library to make HTTP requests, the Picasso library to load images, and the Room library to store data in a SQLite database.

Documentation: The Android SDK includes extensive documentation that explains how to use the tools, libraries, and APIs. You can also find tutorials, sample code, and best practices to help you build high-quality apps.

To develop Android apps, you will need to have Java or Kotlin programming skills. Java is the more widely used language for Android development, but Kotlin is becoming increasingly popular due to its concise syntax and enhanced safety features.





Here is an example of how to create a simple Android app using Java and Android Studio, with file handling capabilities:

Open Android Studio and create a new project.

Give your project a name and select the API level you want to target.

In the project structure, create a new class named "FileHandler" that extends the "Activity" class.

In the "onCreate" method of the "FileHandler" class, create a new file object and specify the file name and location. For example, you could create a file named "data.txt" in the app's internal storage directory:

File file = new File(getFilesDir(), "data.txt");

Use the file object to create a FileWriter object, which you can use to write data to the file. For example, you could write a string to the file:

try {

FileWriter writer = new FileWriter(file);

writer.write("Hello, world!");

writer.close();

} catch (IOException e) {

e.printStackTrace();

}

To read data from the file, create a FileReader object and use a BufferedReader object to read the data. For example, you could read the data you just wrote to the file:

try {

FileReader reader = new FileReader(file);

BufferedReader bufferedReader = new BufferedReader(reader);

String line = bufferedReader.readLine();

bufferedReader.close();

**Main Objective:**

The main objective of Java is to provide a platform-independent programming language that can be used to create software for a variety of devices and operating systems. Java programs are compiled into bytecode, which can be run on any device or platform that has a Java Virtual Machine (JVM) installed. This makes Java ideal for developing cross-platform software, such as mobile applications, desktop applications, and web applications.

The objective of Android Studio is to provide developers with an integrated development environment (IDE) that streamlines the process of creating Android applications. Android Studio provides a range of tools and features to help developers create high-quality apps that are optimized for the Android platform. These features include an intuitive user interface designer, a powerful code editor, debugging and testing tools, and integration with the Android SDK.

The Android SDK provides developers with a set of tools and APIs for building Android applications. The SDK includes a range of libraries and resources that can be used to create apps for a variety of use cases, such as location-based services, multimedia applications, and social networking apps. The objective of the Android SDK is to provide developers with a comprehensive set of tools and resources that can be used to create high-quality, feature-rich Android applications.

Java is a programming language that was first released in 1995. It was developed by Sun Microsystems (later acquired by Oracle) and is currently one of the most popular programming languages in the world. Java is a class-based, object-oriented language that is designed to be platform-independent, meaning that code written in Java can run on any computer or device with a Java Virtual Machine (JVM) installed. Java is used for a wide range of applications, including desktop and web applications, mobile apps, games, and enterprise software.

Android Studio, on the other hand, is an Integrated Development Environment (IDE) for Android app development. It was developed by Google and is based on the IntelliJ IDEA IDE. Android Studio provides a wide range of tools for Android app development, including a code editor, visual layout editor, debugging tools, and performance analysis tools. Android Studio also includes the Android SDK, which provides developers with the necessary tools and libraries to build Android apps.

The main objective of Java and Android Studio is to provide developers with the tools and language constructs necessary to build robust, high-performance, and user-friendly software applications for a wide range of devices and platforms. Java and Android Studio are constantly evolving, with new features and updates being released regularly to meet the changing needs of developers and users alike.

In order to become proficient in Java and Android Studio, it is important to have a solid understanding of the core concepts of programming, such as variables, data types, operators, control structures, and functions. It is also important to have a good understanding of object-oriented programming concepts, such as classes, objects, inheritance, and polymorphism.

To learn Java and Android Studio, there are many resources available online, including tutorials, courses, and documentation. Some popular resources for learning Java include the official Java website, Oracle Academy, Codecademy, Udemy, and Coursera. For Android app development, some popular resources include the official Android developer website, Udacity, Coursera, and Pluralsight.

In conclusion, Java and Android Studio are powerful tools for building software applications for a wide range of devices and platforms. To become proficient in Java and Android Studio, it is important to have a solid understanding of the core programming concepts and to use the available resources to continue learning and expanding your skills.

**INTRODUCTION OF PROJECT**

Automated Teller Machines (ATMs) have become an integral part of modern banking, providing customers with convenient access to a wide range of banking services. The development of ATM systems involves complex software engineering processes to ensure functionality, security, and user-friendliness. The ATM Simulator project in Java aimed to replicate this complexity by designing, implementing, and evaluating a comprehensive simulation of an ATM system.

This report serves as the culmination of the ATM Simulator project, presenting a detailed account of its objectives, methodology, implementation process, and outcomes. It provides insights into the challenges faced, the solutions devised, and the

lessons learned throughout the project's lifecycle.

The primary objective of the ATM Simulator project was to develop a realistic simulation of an ATM system using the Java programming language. This involved creating a user-friendly interface with essential banking functionalities such as cash withdrawal, deposit, balance inquiry, and account management. Additionally, robust backend systems were implemented to handle transaction processing, account management, and security features.

The significance of this project lies in its potential to serve as a valuable educational tool for students and professionals interested in understanding the intricacies of ATM system development. By providing a hands-on experience in designing, implementing, and evaluating an ATM simulation, this project aims to bridge the gap between theoretical knowledge and practical application in the field of software engineering.

Throughout the project's lifecycle, a structured methodology was followed, encompassing phases such as requirement analysis, design, implementation, testing, and evaluation. Each phase was meticulously executed to ensure the quality and reliability of the final product.

In this report, we will delve into the details of each phase, highlighting the methodologies employed, the challenges encountered, and the solutions devised. We will also discuss the outcomes of the project, including the performance, usability, and security aspects of the developed simulator.

Ultimately, the ATM Simulator project in Java represents a significant endeavor in the field of software engineering, providing valuable insights into the development of ATM systems and offering a platform for experimentation, learning, and innovation. Through this report, we aim to share our experiences, findings, and recommendations with the broader community, contributing to the advancement of knowledge in this domain.

**PROJECT OBJECTIVE**

The primary objective of the ATM Simulator project in Java was to develop a realistic simulation of an Automated Teller Machine (ATM) system. This simulation aimed to replicate the functionalities and user experience of a real-world ATM, providing users with a platform to perform basic banking transactions such as cash withdrawal, deposit, balance inquiry, and account management. Specifically, the project sought to achieve the following objectives: Firstly, the development of a user-friendly interface that closely resembles the layout and functionality of a physical ATM machine. The interface should be easy to navigate, visually appealing, and responsive to user input. Secondly, the implementation of essential banking functionalities, including cash withdrawal, deposit, balance inquiry, and account management, adhering to banking standards and regulations. Thirdly, the creation of robust backend systems to handle transaction processing, account management, and data storage, ensuring scalability, efficiency, and security. Moreover, the project aimed to create a simulation environment that replicates the behavior of a real ATM system, including various scenarios such as network downtime, card insertion errors, and transaction timeouts. Finally, security measures were incorporated to safeguard user transactions and sensitive information through authentication mechanisms, encryption techniques, and transaction validation procedures. Overall, the ATM Simulator project aimed to provide users with an immersive and educational experience of using an ATM system in a simulated environment, while also serving as a valuable tool for learning about ATM system development and banking technologies.

**FEASIBILITY STUDY**

**Technical Feasibility:**

The technical feasibility of the project was thoroughly evaluated to assess whether the required technologies, tools, and resources were readily available and adequate for developing the ATM Simulator in Java. It was determined that Java, being a widely used and versatile programming language, provided a suitable platform for implementing the project. Additionally, the availability of development environments, libraries, and frameworks for Java development ensured that the technical requirements could be met effectively.

**Economic Feasibility:**

The economic feasibility of the project was assessed to determine whether the development costs were justified by the potential benefits and returns. It was found that the project could be developed within a reasonable budget, considering the availability of open-source tools and frameworks for Java development. Moreover, the potential benefits of the project, such as its educational value and potential for future enhancements, outweighed the initial investment required.

**Operational Feasibility:**

The operational feasibility of the project was examined to evaluate whether the ATM Simulator would be practical and usable in real-world scenarios. It was determined that the simulation environment could accurately replicate the functionalities and user experience of a real ATM system, providing users with a realistic and immersive banking experience. Additionally, the project's user-friendly interface and intuitive design ensured ease of operation for users.

**Scheduling Feasibility:**

The scheduling feasibility of the project was analyzed to determine whether the project could be completed within the allotted timeframe and resources. A detailed project plan was developed, outlining the tasks, milestones, and timelines for each phase of the project. It was concluded that the project could be executed according to the proposed schedule, provided that sufficient resources and effort were allocated to each phase.

**Literature Review:**

The development of an ATM Simulator project in Java is informed by a diverse range of literature covering key areas such as ATM system architecture, software engineering principles, Java programming, simulation and modeling techniques, and usability and user experience design. In understanding ATM system architecture, studies by authors like James C. Anderson and Roger L. Grimes provide insights into the hardware and software components of ATM systems, offering a foundational understanding necessary for the simulation's design. Additionally, literature on software engineering principles from Roger S. Pressman and Robert C. Martin offers guidance on best practices for software development, ensuring that the project is developed with a focus on quality, reliability, and scalability. Meanwhile, resources on Java programming, including texts by Cay S. Horstmann and Joshua Bloch, as well as online documentation, serve as invaluable references for leveraging Java's features and libraries in developing the simulator. Furthermore, insights from simulation and modeling literature, such as Lawrence M. Leemis and Averill M. Law, inform the creation of realistic simulations by modeling stochastic processes and analyzing simulation results. Lastly, research on usability and user experience design, exemplified by works from Don Norman and Steve Krug, underscores the importance of intuitive interface design and user-centered approaches, guiding the development of a user-friendly simulation interface. Through a comprehensive review of relevant literature, the ATM Simulator project aims to integrate theoretical insights and practical techniques to create an immersive and authentic ATM simulation experience for users.

**METHODOLOGY**

1. **Requirement Analysis:** The project began with a thorough analysis of requirements gathered from stakeholders and potential users. This phase involved identifying the functional and non-functional requirements of the ATM simulator, including desired features, user interactions, system performance, and security considerations. Requirements were documented and prioritized to serve as a foundation for subsequent phases of development.
2. **Design**: Following requirement analysis, detailed design specifications were developed for both the frontend user interface and backend systems of the ATM simulator. The design phase focused on creating intuitive user interfaces that closely resembled the layout and functionality of real ATM machines. Additionally, robust backend systems were designed to handle transaction processing, account management, and data storage efficiently.
3. **Implementation:** With design specifications in place, the implementation phase involved writing modular and maintainable code in Java programming language to realize the design requirements. Java Swing framework was utilized for developing the graphical user interface (GUI), while JDBC (Java Database Connectivity) was used for database interactions. Object-oriented programming principles were employed to ensure code reusability, extensibility, and maintainability. Throughout implementation, rigorous testing was conducted to detect and address any defects or inconsistencies.
4. **Testing**: Testing played a crucial role in ensuring the correctness, reliability, and usability of the ATM simulator. Various testing techniques, including unit testing, integration testing, system testing, and user acceptance testing, were employed to verify the functionality of the simulator under different scenarios. Test cases were designed to cover all aspects of the ATM system, including transaction processing, error handling, and security features. Feedback from testing was used to iteratively refine the simulator and address any identified issues.
5. **Evaluation:** The final phase of the project involved evaluating the performance, usability, and security aspects of the developed ATM simulator. Performance testing was conducted to assess the responsiveness and efficiency of the simulator under various workloads. Usability testing involved soliciting feedback from users to evaluate the intuitiveness and user-friendliness of the interface. Additionally, security testing was performed to identify and address any vulnerabilities in the system's authentication, encryption, and transaction validation mechanisms.

**PROJECT SETUP**

1. **Development Environment:** The development environment for the ATM Simulator project was set up on each developer's workstation. This typically involved installing the Java Development Kit (JDK), a Java Integrated Development Environment (IDE) such as Eclipse or IntelliJ IDEA, and a database management system (e.g., MySQL) for storing account information and transaction records. Additionally, version control software (e.g., Git) was installed to facilitate collaborative development and version management.
2. **Project Structure:** A structured project directory was created to organize source code, resources, and documentation for the ATM Simulator project. The project directory typically included subdirectories for source code, configuration files, libraries, documentation, and test cases. A README file was also created to provide an overview of the project, including installation instructions, usage guidelines, and contribution guidelines for developers.
3. **Dependency Management:** Dependencies for the ATM Simulator project, such as Java libraries and frameworks, were managed using a build automation tool like Apache Maven or Gradle. Dependency management tools allowed developers to specify project dependencies in a configuration file (e.g., pom.xml for Maven), which were automatically downloaded and included in the project build process. This ensured consistency and ease of dependency management across different development environments.
4. **Version Control:** A version control system, such as Git, was used to track changes to the project codebase and facilitate collaboration among developers. A central repository, hosted on a platform like GitHub or Bitbucket, was set up to store the project's source code and track its history. Developers could clone the repository to their local workstations, make changes to the code, and push their changes to the central repository for review and integration.
5. **Continuous Integration:** Continuous Integration (CI) practices were adopted to automate the build and testing process for the ATM Simulator project. A CI server, such as Jenkins or Travis CI, was configured to monitor changes to the project repository and trigger automated builds whenever new code was pushed. Automated tests, including unit tests and integration tests, were executed as part of the CI process to ensure the integrity and correctness of the codebase.

**IMPLEMENTATION**

1. **Frontend Development:** The frontend development of the ATM Simulator focused on creating an intuitive and user-friendly graphical user interface (GUI) that closely resembled the layout and functionality of a real ATM machine. Java Swing framework was used to develop the GUI components, including buttons, text fields, menus, and dialogs. The GUI was designed to provide users with easy access to essential banking functionalities such as cash withdrawal, deposit, balance inquiry, and account management. Visual cues, icons, and labels were used to enhance usability and guide users through the transaction process.
2. **Backend Systems:** Backend systems were developed to handle transaction processing, account management, and data storage for the ATM Simulator. Java programming language was used to implement the backend logic, including transaction validation, account verification, and database interactions. JDBC (Java Database Connectivity) was utilized to establish connections with the database management system (e.g., MySQL) and execute SQL queries for retrieving and updating account information. Robust error handling mechanisms were implemented to handle exceptions and edge cases, ensuring the reliability and consistency of backend operations.
3. **Security Features:** Security features were integrated into the ATM Simulator to safeguard user transactions and sensitive information. Authentication mechanisms, such as PIN verification and card validation, were implemented to authenticate users and prevent unauthorized access to ATM functionalities. Encryption techniques, such as hashing and salting, were employed to secure user passwords and sensitive data stored in the database. Additionally, transaction validation procedures were implemented to detect and prevent fraudulent activities, such as duplicate transactions or insufficient funds.
4. **Testing and Debugging:** Throughout the implementation phase, rigorous testing and debugging were conducted to identify and resolve any issues or inconsistencies in the code. Unit tests were written to verify the functionality of individual components, while integration tests were performed to ensure that different parts of the system worked together seamlessly. User acceptance testing was conducted to evaluate the usability and user experience of the ATM Simulator, gathering feedback from users to identify areas for improvement. Debugging tools and techniques, such as logging, debugging breakpoints, and error tracking, were used to diagnose and fix errors in the codebase.
5. **Iterative Development:** The implementation process followed an iterative development approach, with regular feedback cycles and incremental updates to the ATM Simulator. Feedback from testing and user evaluations was used to iteratively refine the user interface, enhance functionality, and improve overall system performance. Continuous integration practices ensured that changes were integrated smoothly and that the ATM Simulator remained stable and reliable throughout the development process.

**SCREENSHOTS:**

**CONCLUSION**

The ATM Simulator project in Java represents a significant achievement in the field of software engineering, delivering a realistic and user-friendly simulation of an Automated Teller Machine (ATM) system. Through a structured methodology encompassing requirement analysis, design, implementation, testing, and evaluation, the project successfully met its objectives and provided valuable insights into ATM system development.

The implementation of the ATM Simulator involved the development of a user-friendly graphical user interface (GUI) using Java Swing framework, robust backend systems for transaction processing and account management, and integration of security features to safeguard user transactions and sensitive information. Rigorous testing and debugging processes ensured the reliability, correctness, and security of the ATM Simulator, while iterative development cycles allowed for continuous refinement and improvement based on user feedback and testing results.

The ATM Simulator project not only achieved its technical goals but also demonstrated its educational value as a learning tool for students and professionals interested in software development and banking technologies. By providing a hands-on experience in designing, implementing, and evaluating an ATM simulation, the project bridged the gap between theoretical knowledge and practical application, empowering users to gain insights into ATM system architecture, software engineering principles, and Java programming.

**FUTURE WORK**

1. **Advanced Functionality:** Expand the functionality of the ATM Simulator to include more advanced banking operations such as fund transfers between accounts, bill payments, check deposits, and currency exchange. Implement additional security features such as biometric authentication, multi-factor authentication, and transaction monitoring to enhance security.
2. **Enhanced User Experience:** Improve the user interface design of the ATM Simulator to make it more visually appealing, intuitive, and responsive. Incorporate multimedia elements such as animations, sound effects, and interactive tutorials to enhance the user experience and facilitate learning for users.
3. **Accessibility Features:** Introduce accessibility features into the ATM Simulator to make it more inclusive and usable for individuals with disabilities. Implement features such as screen reader compatibility, keyboard navigation, and customizable font sizes and colors to ensure that the simulator is accessible to users with diverse needs.
4. **Mobile Compatibility:** Adapt the ATM Simulator for mobile devices by developing a responsive web application or a dedicated mobile app. Optimize the user interface for smaller screens and touch input, and leverage device-specific features such as GPS for locating nearby ATMs and push notifications for transaction alerts.
5. **Integration with Real Data:** Integrate the ATM Simulator with real-world banking data and APIs to provide users with access to real account information and transaction history. Partner with financial institutions to securely access and retrieve account data, and implement features such as account aggregation and financial analysis to provide users with valuable insights into their finances.
6. **Educational Resources:** Develop educational resources and tutorials to accompany the ATM Simulator, including documentation, video tutorials, and interactive learning modules. Provide users with guidance on using the simulator effectively, understanding banking concepts, and exploring career opportunities in the financial industry.
7. **Community Collaboration:** Foster a community of users and developers around the ATM Simulator project by establishing forums, discussion groups, and collaborative development platforms. Encourage users to contribute feedback, suggestions, and code contributions to enhance the project and address emerging needs and challenges.

**REFERNCES**

1. Anderson, James C. "Understanding ATM Technology." CRC Press, 1999.
2. Bloch, Joshua. "Effective Java." Addison-Wesley Professional, 2018.
3. Grimes, Roger L. "Hacking ATM Machines." Wiley, 2014.
4. Horstmann, Cay S. "Core Java Volume I - Fundamentals." Pearson, 2019.
5. Law, Averill M., and W. David Kelton. "Simulation Modeling and Analysis." McGraw-Hill Education, 2015.
6. Leemis, Lawrence M., and Stephen K. Park. "Discrete-Event Simulation: A First Course." Pearson, 2006.
7. Martin, Robert C. "Clean Code: A Handbook of Agile Software Craftsmanship." Prentice Hall, 2008.
8. Norman, Don. "The Design of Everyday Things." Basic Books, 2013.
9. Pressman, Roger S. "Software Engineering: A Practitioner's Approach." McGraw-Hill Education, 2014.
10. Krug, Steve. "Don't Make Me Think." New Riders, 2014.
11. Java Documentation. Oracle Corporation. [<https://docs.oracle.com/javase/>].
12. MySQL Documentation. Oracle Corporation. [<https://dev.mysql.com/doc/>].

THANK YOU!!!