INDUSTRIAL TRAINING REPORT

ON

ATM SIMULATION SYSTEM

Submitted in partial fulfillment of the requirements For the award of the degree of

**BACHELOR OF TECHNOLOGY IN**

**INFORMATION TECHNOLOGY**

Submitted By

**MOHD SAMIUDDIN**

**(00815607721)**

 **To**

Under Guidance of Associate Professor, IT department

**Department of Information Technology**

**Dr. Akhilesh Das Gupta Institute of Technology & Management (Guru Gobind Singh Indraprastha University, Dwarka, Delhi.) Delhi -110053.**

**December 2023**

DECLARATION

I hereby declare that I have completed my 18 weeks Industrial training with Anudip Foundation from, 2nd August 2022 to 17 November 2022 under the guidance of

Ms. Sudha Kadakol.

I have declared that I have worked with full dedication during these 18 weeks of training and my learning outcomes fulfill the requirements of training for the award of degree of Bachelor of Technology (B.Tech) in Information Technology, Dr Akhilesh Das Gupta Institute of Technology & Management, New Delhi.

Mohd Samiuddin

00815607721

Information Technology

**ACKNOWLEDGEMENT**

It is my proud privilege and duty to acknowledge the kind of help and guidance received from several people in preparation of this report. It would not have been possible to prepare this report in this form without their valuable help, cooperation and guidance.

First and foremost, I wish to record our sincere gratitude to Anudip Coordinators for their constant support and encouragement in preparation of this report and for making available materieals and Lectures facilities needed to prepare this report.

The seminar on ―Java Full Stack Development was very helpful to us in giving the necessary background information and inspiration in choosing this topic for the seminar. Their contributions and technical support in preparing this report are greatly acknowledged.

I am also very grateful to the entire Information Technology department of Dr.Akhilesh Das Gupta Institute of Technology and Management for their constant guidance.

Mohd Samiuddin

00815607721

Information Technology



# ABSTRACT

An ATM simulation system replicates the functionalities of an Automated Teller Machine (ATM) in a virtual environment. It typically involves software designed to mimic the operations of a real ATM, providing users with a platform to perform various banking transactions. The system usually includes features such as account balance inquiries, cash withdrawals, fund transfers between accounts, bill payments, and sometimes even services like mobile recharge or check deposits.

Its abstract would likely outline the system's structure, including modules for user authentication, transaction processing, database management, and interface design. Security measures, like encryption protocols and authentication methods, would be described to safeguard user information and transactions. Additionally, the abstract might touch on the user experience aspects, aiming for an intuitive interface to ensure ease of use for customers.

Overall, the abstract of an ATM simulation system would encapsulate the core functionalities, security measures, and user interface design aspects of the software, emphasizing its ability to simulate real-world ATM operations in a secure and user-friendly manner.

# CONTENTS

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Chapters** | **Page No.** |
| **1.** | **Title** | **I** |
| **2.** | **Declaration** | **II** |
| **3.** | **Certificate from Organization** | **III** |
| **4.** | **Acknowledgement** | **IV** |
| **5.** | **Abstract** | **V** |
| **6.** | **Contents** | **VI** |
| **7.** | **Chapter 1 – Introduction:**   * 1. What is Programming Language   2. Java Archetecture   3. Installing Eclipse IDE   4. Java as oops   5. Java Security | **10-12** |
| **8.** | **Chapter 2 - Training Contents**  2.1. Introduction to Java   * 1. Data Types and Variables in Java   2. Basics of Programming in Java   3. Collection in java   4. Database (MySql)      1. CRUD Operation      2. Match different tables with Join      3. Group by,Having      4. Where Clause   2.6 Connecting Database using JDBC and Hibernate   * + 1. Operations using JDBC   2.6.2 Operations using Hibernate   * 1. Introduction to Spring | **13-40** |
| **9.** | **Chapter 3 - Designing And Result Analysis**   * 1. Collecting Raw Data and working on it.   2. Problem Analysis      1. Product Defination      2. Feasibility Analysis | **40-54** |
| **11.** | **Chapter 4**  4.1 Conclusion  4.2 Future Scope | **55** |
| **12.** | **References** | **56** |

**Chapter 1 – Introduction**

* 1. **What is Programming Language**

A programming language is a formal set of instructions used to communicate with computers. It's a structured way for humans to give commands and express algorithms that computers can understand and execute. These languages are designed with specific syntax and semantics, allowing programmers to create software, applications, websites, and more.

Programming languages vary in complexity, purpose, and application. They can be high-level, like Python, Java, or C++, which offer more abstraction and readability but need translation by a compiler or interpreter before execution. Low-level languages, such as assembly language, are closer to machine code and directly understandable by hardware but are more challenging to write and understand.

* 1. **Java Archetecture**

**Java Architecture** is a collection of components, i.e., **JVM, JRE,** and **JDK**. **It** integrates the process of interpretation and compilation. It defines all the processes involved in creating a Java program. **Java Architecture** explains each and every step of how a program is compiled and executed.

**Java Architecture** can be explained by using the following steps:

* There is a process of compilation and interpretation in Java.
* Java compiler converts the Java code into byte code.
* After that, the JVM converts the byte code into machine code.
* The machine code is then executed by the machine.

The following figure represents the **Java Architecture** in which each step is elaborate graphically.



* 1. **Installing Eclipse IDE**

**Eclipse IDE** is one of the most popular integrated development environments (IDEs). It is mostly used by Java developers, but it can also support almost every other programming language like C/C++, PHP, Scala, Groovy, Clojure, and many more. It has a rich set of features that include support for various programming languages, code analysis, graphical debugging, and unit testing.

In this shot, we will look at the installation process of Eclipse IDE.



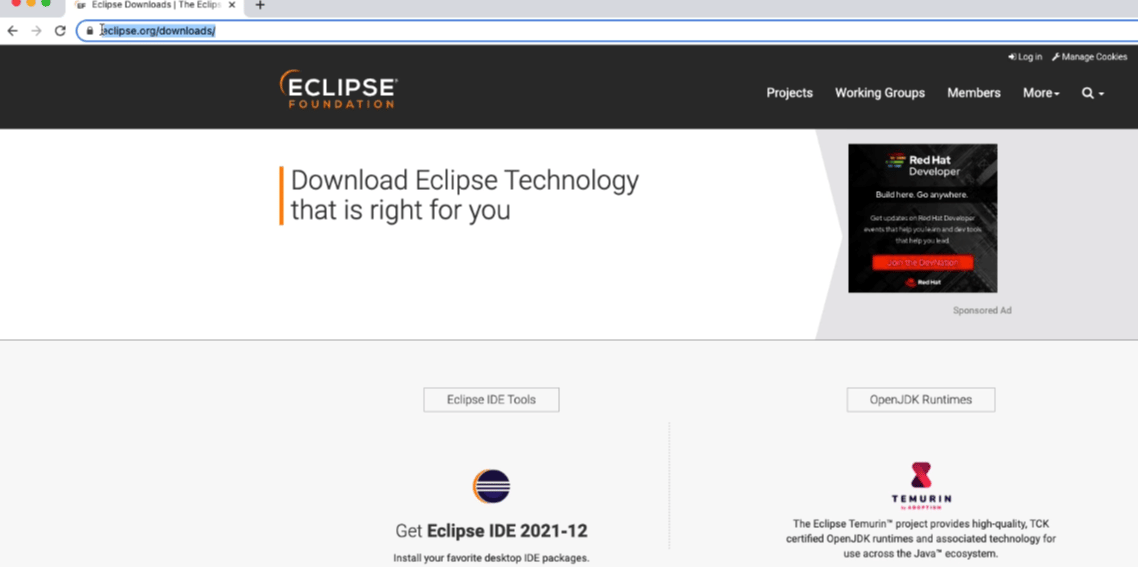
Eclipse logo

### Installation steps

Below is the step-by-step process for installing the Eclipse IDE:

### Step 1 - Download the Eclipse installer

The easiest way to install eclipse IDE is to download and run the installer. To download the Eclipse installer, go to [this url](https://www.eclipse.org/downloads/) and select the installer executable as per your operating system platform.



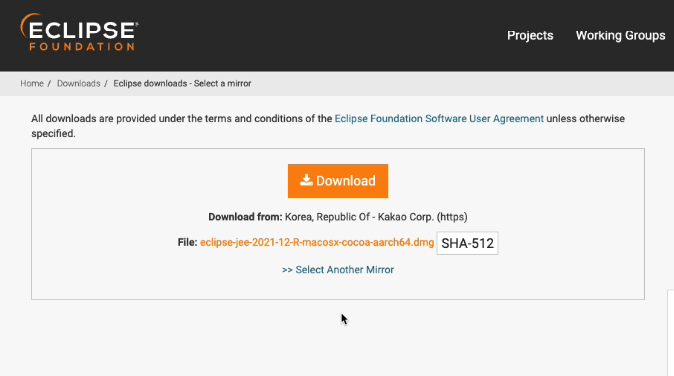
Eclipse download site

Then, select the closest mirror site to download the required package.

For Windows users, select the folder, e.g., C:/Users/[username]/Downloads to download the installer.

For Mac users, select the folder /Users/[username]/Downloads.

Here, “[username]” represents the username of the operating system you are using.

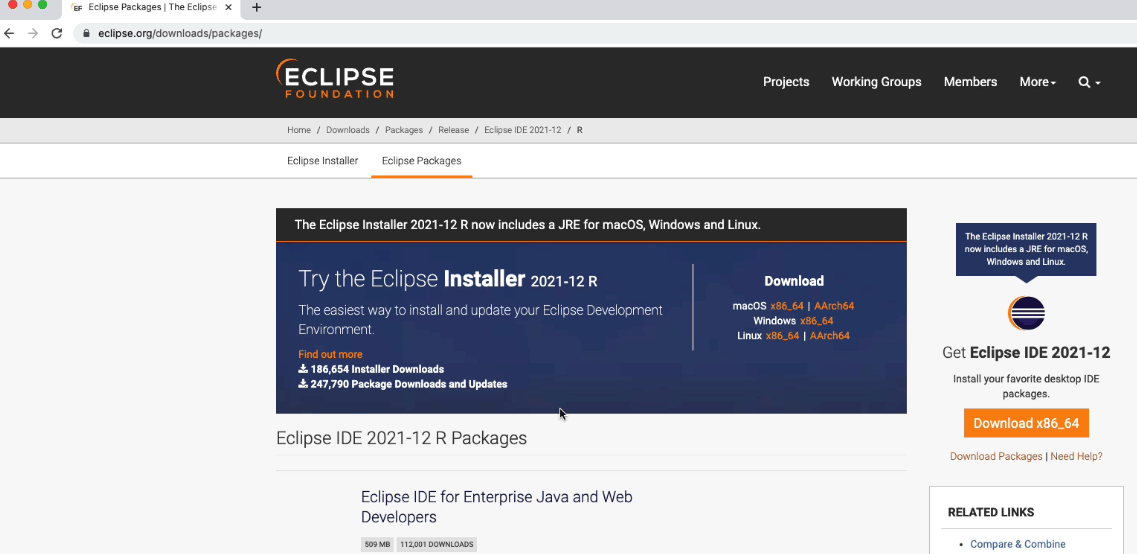


Choosing the closest download mirror site

### Step 2 - Running the installer

After the download is completed, execute the Eclipse installer. You may need to extract the content if you use Mac or Linux.

You would also get a security warning for executing a file downloaded from the Internet. Verify if the publisher is the Eclipse Foundation and choose to continue with the file’s execution as you can trust this source.



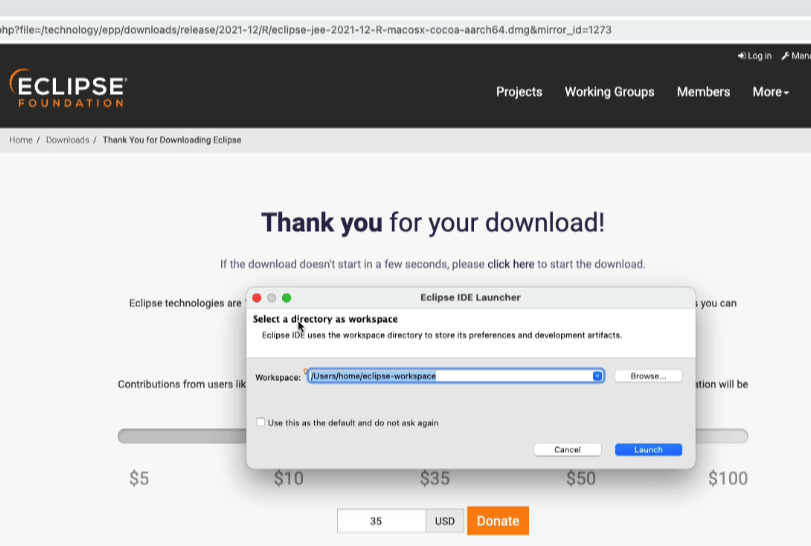
Eclipse IDE package download

### Step 3 - Choosing the installation directory/folder

When the installer prompts you to choose the installation location, keep this as your default user directory and begin the installation process. Windows users can also create start menu entries and desktop shortcuts.

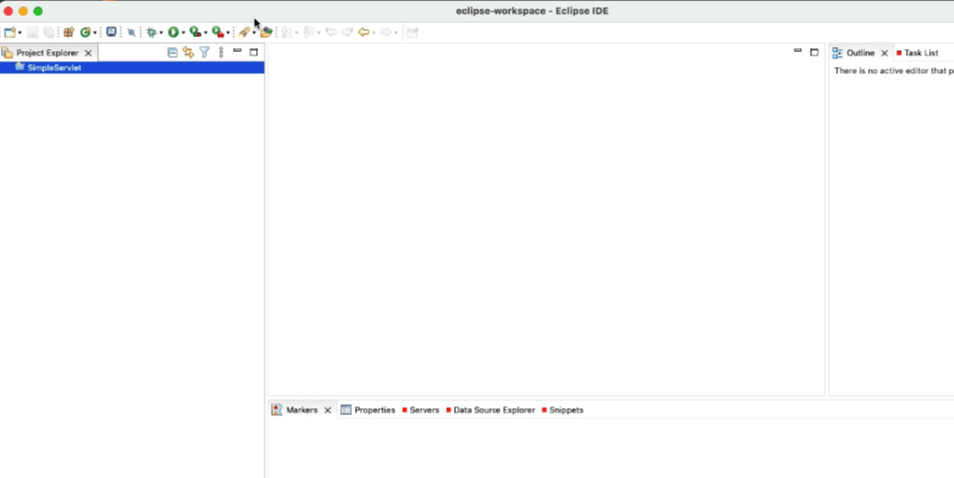
### Step 4 - Launching the Eclipse IDE

After the installation is complete, go to the local folder on your system where you installed the application in the previous step. Find the executable file for the Eclipse IDE and run it to launch the IDE.



Selecting workspace directory

During the first run, Eclipse asks users to specify a folder to create the workspace for their projects. After this is provided, the Eclipse IDE gets launched. This completes the installation process.



Eclipse IDE

Now you can enjoy coding using the Eclipse IDE!

* 1. **Java as oops**

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter development time

## **Java - What are Classes and Objects?**

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:

## **class**

Fruit

## **objects**

Apple

Banana

Mango

* 1. **Java as a Secure Language**

Java is the most popular **object-oriented programming language**. It provides a variety of salient features that are preferred by the developers. It is the reason that a billion of devices runs on Java. In this section, we are going to discuss **why Java is secure**.

Java is secure due to the following reasons:

* Java programs run inside a virtual machine which is known as a sandbox.
* Java does not support explicit pointer.
* Byte-code verifier checks the code fragments for illegal code that can violate access right to object.
* It provides java.security package implements explicit security.
* It provides library level safety.
* Run-time security check takes place when we load new code.

[Java](https://www.javatpoint.com/java-tutorial) provides some other features that make Java more secure.

* JVM
* Security API's
* Security Manager
* Auto Memory Management
* No Concept of Pointers
* Compile-time Checking
* Cryptographic Security
* Java Sandbox
* Exception Handling
* ClassLoader

AD

**JVM**

[JVM](https://www.javatpoint.com/jvm-java-virtual-machine) plays a vital role to provide security. It verifies the byte-code. The JVM provides guarantees that there is no unsafe operation going to execute. It also helps to diminish the possibilities of the programmers who suffer from memory safety flaws.

**Security API's**

Java class libraries provide several API that leads to security. These APIs contain cryptographic algorithms and authentication protocols that lead to secure communication.

**Byte Code**

Every time when a user compiles the Java program, the Java compiler creates a class file with Bytecode, which are tested by the JVM at the time of program execution for viruses and other malicious files.

**Security Manager**

The security manager is responsible for checking the permissions and properties of the classes. It monitors the system resources accessed by the authorized classes. It also controls socket connections.

**No Concept of Pointers**

Java does not provide support for pointers concept. It is the main security [features of Java](https://www.javatpoint.com/features-of-java). The use of pointers may lead to unauthorized read or write operations. Therefore, the user cannot point to any memory locations.

**Memory management**

Java automatically manages memory which is known as garbage collection. The JVM manages memory itself. The programmers are free from memory management. Hence, there is no chance to fault in memory management.

**Compile-time checking**

Compile-time checking also makes the Java secure. Consider a scenario in which an unauthorized method is trying to access the private variable, in this case, the JVM gives the compile-time error. It prevents the system from the crash.

AD

**Cryptographic Security**

Java provides a class named **java.secrurity.SourceCode** that also provides security. If we get code from other sources, we should check from where the code is coming. The class maintains the source information and provides guarantees to keep a digital signature and cryptographic security.

**Java Sandbox**

Java Sandbox is a major component of security consideration. It is a restricted area where applets are run. Java does not provide system resources without check if an applet is to be run.

**Exception Handling**

AD

The exception handling feature adds more security in Java. The feature reports the error to the programmer during the runtime. The code will not run until the programmer will not rectify it.

**Java ClassLoader**

There are a number of class loaders present in JVM. It provides and maintains namespaces for specific classes. The advantage of the [ClassLoader](https://www.javatpoint.com/classloader-in-java) is that the untrusted classes would not behave like a trusted one.

**Chapter 2**

* 1. **Introduction to Java**

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995. James Gosling is known as the father of Java. Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

**Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

# Features of Java

The primary objective of [Java programming](https://www.javatpoint.com/java-tutorial) language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

A list of the most important features of the Java language is given below.



1. [Simple](https://www.javatpoint.com/features-of-java#Simple)
2. [Object-Oriented](https://www.javatpoint.com/features-of-java#Object-Oriented)
3. [Portable](https://www.javatpoint.com/features-of-java#Portable)
4. [Platform independent](https://www.javatpoint.com/features-of-java#Platform-independent)
5. [Secured](https://www.javatpoint.com/features-of-java#Secured)
6. [Robust](https://www.javatpoint.com/features-of-java#Robust)
7. [Architecture neutral](https://www.javatpoint.com/features-of-java#Architecture-neutral)
8. [Interpreted](https://www.javatpoint.com/features-of-java#Interpreted)
9. [High Performance](https://www.javatpoint.com/features-of-java#High-Performance)
10. [Multithreaded](https://www.javatpoint.com/features-of-java#Multithreaded)
11. [Distributed](https://www.javatpoint.com/features-of-java#Distributed)
12. [Dynamic](https://www.javatpoint.com/features-of-java#Dynamic)

**2.2Data Types and Variables in Java**

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

## **Java Primitive Data Types**

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in [Java language](https://www.javatpoint.com/java-tutorial).

Java is a statically-typed programming language. It means, all [variables](https://www.javatpoint.com/java-variables) must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

* boolean data type
* byte data type
* char data type
* short data type
* int data type
* long data type
* float data type
* double data type



# Java Variables

A variable is a container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of [data types in Java](https://www.javatpoint.com/java-data-types): primitive and non-primitive.

## **Variable**

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.



1. **int** data=50;//Here data is variable

### **Types of Variables**

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

* local variable
* instance variable
* static variable



**2.3Basics of Programming in Java**

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, [Java](https://www.javatpoint.com/java-tutorial) provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

**Java provides three types of control flow statements**.

1. **Decision Making statements**
   * if statements
   * switch statement
2. **Loop statements**
   * do while loop
   * while loop
   * for loop
   * for-each loop
3. **Jump statements**
   * break statement
   * continue statement

### **Switch Statement:**

In Java, [Switch statements](https://www.javatpoint.com/java-switch) are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:

* The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
* Cases cannot be duplicate
* Default statement is executed when any of the case doesn't match the value of expression. It is optional.
* Break statement terminates the switch block when the condition is satisfied.  
  It is optional, if not used, next case is executed.
* While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

1. }

### **Loop Statements**

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop
2. while loop
3. do-while loop

Let's understand the loop statements one by one.

### **Java for loop**

In Java, [for loop](https://www.javatpoint.com/java-for-loop) is similar to [C](https://www.javatpoint.com/c-programming-language-tutorial) and [C++](https://www.javatpoint.com/cpp-tutorial). It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

1. **for**(initialization, condition, increment/decrement) {
2. //block of statements
3. }

The flow chart for the for-loop is given below.



### **Java for-each loop**

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

1. **for**(data\_type var : array\_name/collection\_name){
2. //statements
3. }

Consider the following example to understand the functioning of the for-each loop in Java.

**Calculation.java**

1. **public** **class** Calculation {
2. **public** **static** **void** main(String[] args) {
3. // TODO Auto-generated method stub
4. String[] names = {"Java","C","C++","Python","JavaScript"};
5. System.out.println("Printing the content of the array names:\n");
6. **for**(String name:names) {
7. System.out.println(name);
8. }
9. }
10. }

**Output:**

Printing the content of the array names:

Java

C

C++

Python

JavaScript

### **Java while loop**

The [while loop](https://www.javatpoint.com/java-while-loop) is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

1. **while**(condition){
2. //looping statements
3. }

The flow chart for the while loop is given in the following image.



### **Java do-while loop**

The [do-while loop](https://www.javatpoint.com/java-do-while-loop) checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

1. **do**
2. {
3. //statements
4. } **while** (condition);

The flow chart of the do-while loop is given in the following image.



### **Jump Statements**

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

### **Java break statement**

As the name suggests, the [break statement](https://www.javatpoint.com/java-break) is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

# Objects and Classes in Java

1. [Object in Java](https://www.javatpoint.com/object-and-class-in-java#object)
2. [Class in Java](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Instance Variable in Java](https://www.javatpoint.com/object-and-class-in-java#objectinstancevariable)
4. [Method in Java](https://www.javatpoint.com/object-and-class-in-java#objectmethod)
5. [Example of Object and class that maintains the records of student](https://www.javatpoint.com/object-and-class-in-java#objectex2)
6. [Anonymous Object](https://www.javatpoint.com/object-and-class-in-java#objectannonymous)

In this page, we will learn about Java objects and classes. In object-oriented programming technique, we design a program using objects and classes.

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

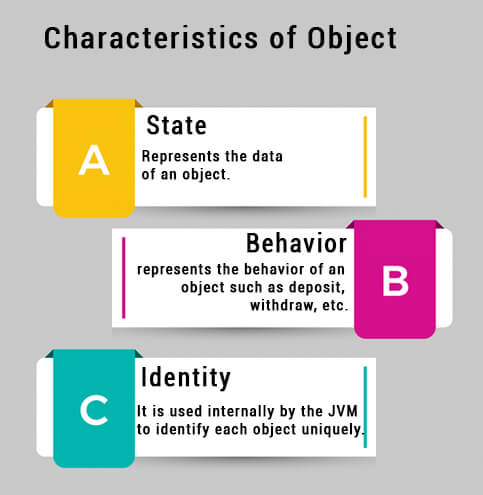
### **What is an object in Java**



An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

* **State:** represents the data (value) of an object.
* **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
* **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.



For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

**An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

**Object Definitions:**

* An object is a real-world entity.
* An object is a runtime entity.
* The object is an entity which has state and behavior.
* The object is an instance of a class.

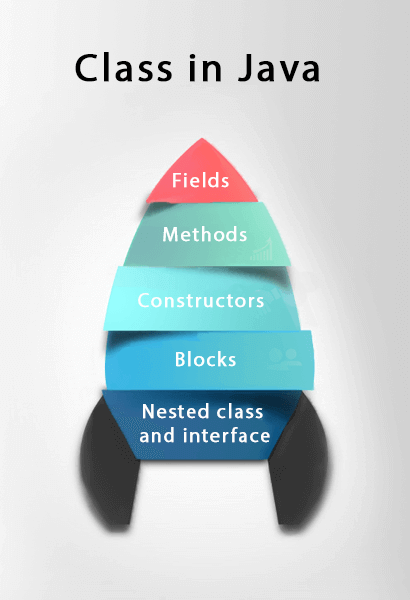
AD

## **What is a class in Java**

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

* **Fields**
* **Methods**
* **Constructors**
* **Blocks**
* **Nested class and interface**



### **Instance variable in Java**

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

### **Method in Java**

In Java, a method is like a function which is used to expose the behavior of an object.

#### **Advantage of Method**

* Code Reusability
* Code Optimization

### **new keyword in Java**

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

**2.4Collection in java**

1. [Java Collection Framework](https://www.javatpoint.com/collections-in-java)
2. [Hierarchy of Collection Framework](https://www.javatpoint.com/collections-in-java#collectionhierarchy)
3. [Collection interface](https://www.javatpoint.com/collections-in-java#collectionmethods)
4. [Iterator interface](https://www.javatpoint.com/collections-in-java#collectioniterator)

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](https://www.javatpoint.com/java-arraylist), Vector, [LinkedList](https://www.javatpoint.com/java-linkedlist), [PriorityQueue](https://www.javatpoint.com/java-priorityqueue), HashSet, LinkedHashSet, TreeSet).

#### **What is Collection in Java**

A Collection represents a single unit of objects, i.e., a group.

#### **What is a framework in Java**

* It provides readymade architecture.
* It represents a set of classes and interfaces.
* It is optional.

#### **What is Collection framework**

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

### **Hierarchy of Collection Framework**

Let us see the hierarchy of Collection framework. The **java.util** package contains all the [classes](https://www.javatpoint.com/object-and-class-in-java) and [interfaces](https://www.javatpoint.com/interface-in-java) for the Collection framework.



### **Iterator interface**

|  |
| --- |
| Iterator interface provides the facility of iterating the elements in a forward direction only. |

#### **Methods of Iterator interface**

There are only three methods in the Iterator interface. They are:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean hasNext() | It returns true if the iterator has more elements otherwise it returns false. |
| 2 | public Object next() | It returns the element and moves the cursor pointer to the next element. |
| 3 | public void remove() | It removes the last elements returned by the iterator. It is less used. |

## **Iterable Interface**

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

1. Iterator<T> iterator()

It returns the iterator over the elements of type T.

## **Collection Interface**

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

Some of the methods of Collection interface are Boolean add ( Object obj), Boolean addAll ( Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

## **List Interface**

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

**2.5Database (MySql)**

**What is Database?**

Database is a collection of interrelated data.

**What is DBMS?**

DBMS (Database Management System) is software used to create, manage, and organize databases.

**What is RDBMS?**

● RDBMS (Relational Database Management System) - is a DBMS based on the concept of tables (also called relations).

● Data is organized into tables (also known as relations) with rows (records) and columns (attributes).

● Eg - MySQL, PostgreSQL, Oracle etc.

**2.5.1CRUD Operation**

We use SQL for CRUD Operations :

● CREATE - To create databases, tables, insert tuples in tables etc

● READ - To read data present in the database. ● UPDATE - Modify already inserted data.

● DELETE - Delete database, table or specific data point/tuple/row or multiple rows.

* + 1. **Matche different tables with Join**

In MySQL, the JOIN operation is used to combine rows from two or more tables based on a related column between them. This operation allows you to retrieve data that spans across multiple tables by specifying how the tables are related.

There are different types of JOIN operations:

**INNER JOIN**: Returns rows when there is a match in both tables.

**SELECT \* FROM table1 INNER JOIN table2 ON table1.column = table2.column;**

**LEFT JOIN (or LEFT OUTER JOIN):** Returns all rows from the left table and matching rows from the right table.

**SELECT \* FROM table1 LEFT JOIN table2 ON table1.column = table2.column;**

**RIGHT JOIN (or RIGHT OUTER JOIN):** Returns all rows from the right table and matching rows from the left table.

**SELECT \* FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;**

**FULL JOIN (or FULL OUTER JOIN):** Returns all rows when there is a match in either left or right table.

**SELECT \* FROM table1 FULL JOIN table2 ON table1.column = table2.column;**

* + 1. **Group by,Having**

In SQL, the GROUP BY and HAVING clauses are used together to perform operations on groups of rows and filter the results based on specified conditions.

**GROUP BY**: This clause is used to group rows that have the same values in a specified column or columns into summary rows, like applying aggregate functions (SUM, COUNT, AVG, etc.) on those groups.

**SELECT column\_name, aggregate\_function(column\_name)**

**FROM table\_name**

**GROUP BY column\_name;**

**HAVING:** The HAVING clause filters the grouped rows based on a specified condition after the GROUP BY operation has been performed. It's similar to the WHERE clause but is specifically used with aggregated data.

**SELECT column\_name, aggregate\_function(column\_name)**

**FROM table\_name**

**GROUP BY column\_name**

**HAVING condition;**

For example:

**SELECT department, AVG(salary) as avg\_salary**

**FROM employees**

**GROUP BY department**

**HAVING AVG(salary) > 50000;**

This query would compute the average salary per department in the employees table and return only those departments where the average salary is greater than $50,000.

* + 1. **Where Clause**

In SQL, the WHERE clause is used to filter records from a table based on specified conditions. It's employed within a SELECT, UPDATE, DELETE, or INSERT statement to retrieve or modify specific rows that meet certain criteria.

Here's a basic syntax for the WHERE clause:

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE condition;**

Some common operators used in the WHERE clause include:

Comparison Operators: =, <> (not equal to), >, <, >=, <=.

Logical Operators: AND, OR, NOT.

For instance:

**SELECT \* FROM employees**

**WHERE department = 'Sales' AND salary > 50000;**

This query would fetch all columns from the employees table where the department is 'Sales' and the salary is greater than $50,000.

The WHERE clause allows you to specify conditions that data must meet for inclusion in the result set, enabling precise retrieval or modification of information from the database.

**2.6 Connecting Database using JDBC and Hibernate**

**JDBC:**

Step 1: Load the JDBC Driver:

Class.forName("com.mysql.cj.jdbc.Driver");

Step 2: Establish Connection:

String url = "jdbc:mysql://localhost:3306/your\_database\_name";

String username = "your\_username";

String password = "your\_password";

Connection connection = DriverManager.getConnection(url, username, password);

Step 3: Create Statements and Execute Queries:

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("SELECT \* FROM your\_table");

// Process the result set or perform other database operations

**Hibernate:**

Hibernate is an ORM (Object-Relational Mapping) framework that abstracts away much of the database interaction and provides a higher-level interface for developers. Here's a basic setup:

**Step 1: Configure Hibernate**:

Set up hibernate.cfg.xml or use annotations for configuration.

**Step 2: Create Entity Classes:**

java

Copy code

@Entity

@Table(name = "your\_table")

public class YourEntity {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

// Other fields and annotations

// Getters and setters

}

**Step 3: Establish SessionFactory:**

SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();

Session session = sessionFactory.openSession();

**Step 4: Perform Database Operations:**

Transaction transaction = session.beginTransaction();

// Save or retrieve entities

YourEntity entity = new YourEntity();

// Set entity properties

session.save(entity);

// Commit the transaction

transaction.commit();

session.close();

Hibernate simplifies database interactions by allowing developers to work with Java objects directly, and it handles the translation between these objects and the database tables.

Both JDBC and Hibernate serve the purpose of database connectivity in Java, but Hibernate offers more abstraction and convenience by managing the database operations through objects and their relationships, while JDBC operates at a lower level, requiring more manual handling of SQL queries and connections. Choosing between them depends on the project's requirements, complexity, and developer preferences.

* + 1. **Operations using JDBC**

**Connecting to a Database:**

Connection connection = null;

try {

Class.forName("com.mysql.cj.jdbc.Driver");

String url = "jdbc:mysql://localhost:3306/your\_database\_name";

String username = "your\_username";

String password = "your\_password";

connection = DriverManager.getConnection(url, username, password);

// Use the connection

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

} finally {

if (connection != null) {

try {

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**Executing Queries:**

Creating a Statement and Executing Query:

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("SELECT \* FROM your\_table");

// Process the result set

while (resultSet.next()) {

// Retrieve data from each row

}

resultSet.close();

statement.close();

Prepared Statements (Parameterized Queries):

PreparedStatement preparedStatement = connection.prepareStatement("SELECT \* FROM your\_table WHERE id = ?");

preparedStatement.setInt(1, 123); // Set parameter values

ResultSet resultSet = preparedStatement.executeQuery();

// Process the result set

resultSet.close();

preparedStatement.close();

**Modifying Data:**

**Inserting Data:**

Statement statement = connection.createStatement();

int rowsAffected = statement.executeUpdate("INSERT INTO your\_table (column1, column2) VALUES ('value1', 'value2')");

// Check the number of rows affected

statement.close();

**Updating Data:**

PreparedStatement preparedStatement = connection.prepareStatement("UPDATE your\_table SET column1 = ? WHERE id = ?");

preparedStatement.setString(1, "new\_value");

preparedStatement.setInt(2, 123); // Set parameter values

int rowsAffected = preparedStatement.executeUpdate();

// Check the number of rows affected

preparedStatement.close();

**Deleting Data:**

Statement statement = connection.createStatement();

int rowsAffected = statement.executeUpdate("DELETE FROM your\_table WHERE id = 123");

// Check the number of rows affected

statement.close();

* + 1. **Operations using Hibernate**

**1. Configuration and Setup:**

SessionFactory Initialization:

SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();

**2. CRUD Operations:**

Create (Save) Entity:

Session session = sessionFactory.openSession();

Transaction transaction = session.beginTransaction();

YourEntity entity = new YourEntity();

// Set entity properties

session.save(entity);

transaction.commit();

session.close();

Read (Retrieve) Entity by ID:

Session session = sessionFactory.openSession();

YourEntity entity = session.get(YourEntity.class, entityId);

session.close();

// Use the retrieved entity

Update Entity:

java

Copy code

Session session = sessionFactory.openSession();

Transaction transaction = session.beginTransaction();

YourEntity entity = session.get(YourEntity.class, entityId);

// Modify entity properties

session.update(entity);

transaction.commit();

session.close();

**Delete Entity:**

Session session = sessionFactory.openSession();

Transaction transaction = session.beginTransaction();

YourEntity entity = session.get(YourEntity.class, entityId);

session.delete(entity);

transaction.commit();

session.close();

**3. HQL (Hibernate Query Language):**

HQL Queries:

Session session = sessionFactory.openSession();

Query<YourEntity> query = session.createQuery("FROM YourEntity WHERE someCondition = :value", YourEntity.class);

query.setParameter("value", someValue);

List<YourEntity> resultList = query.getResultList();

session.close();

// Process the resultList

* 1. **Introduction to Spring**

Spring is a comprehensive framework for building robust, scalable, and maintainable Java applications. It provides a wide range of tools and libraries that simplify various aspects of enterprise application development. Here’s an introduction to key components and concepts within the Spring framework:

**1. Core Features:**

Dependency Injection (DI): Spring's core principle is dependency injection, which manages object dependencies, making code loosely coupled and easier to test.

Inversion of Control (IoC): Objects do not create their dependencies but are provided with them. This is achieved through IoC, where control of object creation is inverted from the application to the Spring framework.

**2. Modules in the Spring Framework**:

**Spring Core Container**: Manages Spring beans and their lifecycles through ApplicationContext and BeanFactory.

Spring AOP (Aspect-Oriented Programming): Provides support for aspect-oriented programming and allows implementing cross-cutting concerns.

**Spring JDBC:** Simplifies database access and eliminates the need for boilerplate JDBC code.

**Spring MVC (Model-View-Controller):** A web framework for building web applications following the MVC design pattern.

Spring Security: Offers security features for authentication, authorization, and other security aspects.

**Spring Boot:** Simplifies the setup and development of Spring-based applications by providing a convention-over-configuration approach.

**3. Spring Boot:**

**Auto-Configuration:** Automatically configures the Spring application based on included dependencies.

Standalone Applications: Allows creating production-ready standalone Spring-based applications with embedded servers.

Reduced Configuration: Minimizes boilerplate code and XML configurations by using sensible defaults.

**4. Advantages of Spring:**

**Modularity:** Spring's modular design allows using only the required parts of the framework, promoting lightweight applications.

**Testability**: Dependency injection facilitates easier unit testing by enabling mocking and easier integration testing.

Scalability: Provides support for developing scalable applications due to its loosely coupled nature and support for enterprise-level features.

**Community Support**: An active and large community continuously supports and enhances the framework.

**5. Spring Ecosystem**:

**Spring Data**: Simplifies working with various data storage solutions.

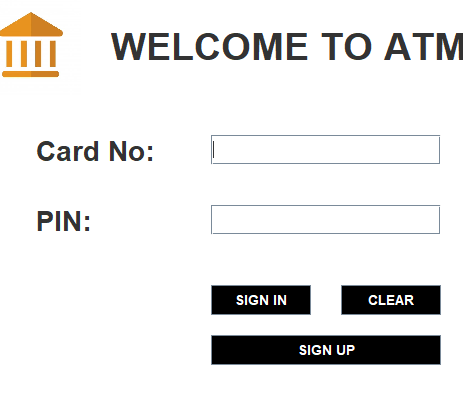
**Spring Integration**: Helps in integrating enterprise systems.

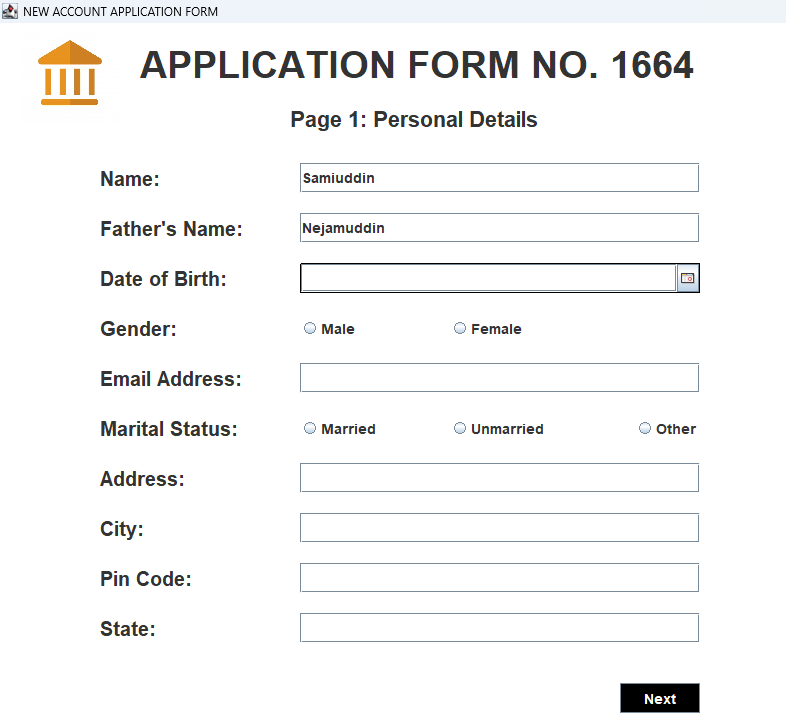
**Spring Cloud**: A set of tools for building cloud-native applications.

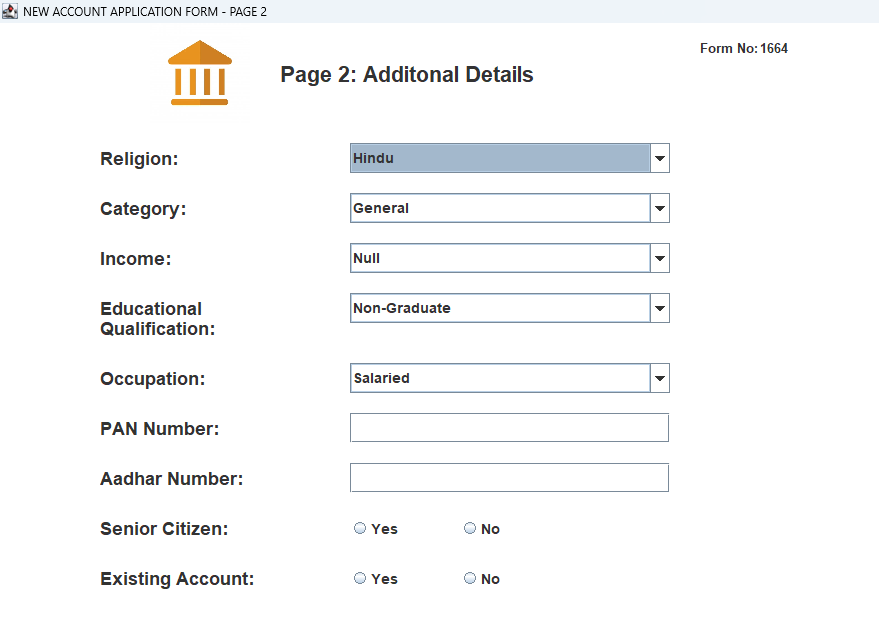
**Spring Batch:** Supports batch processing applications.

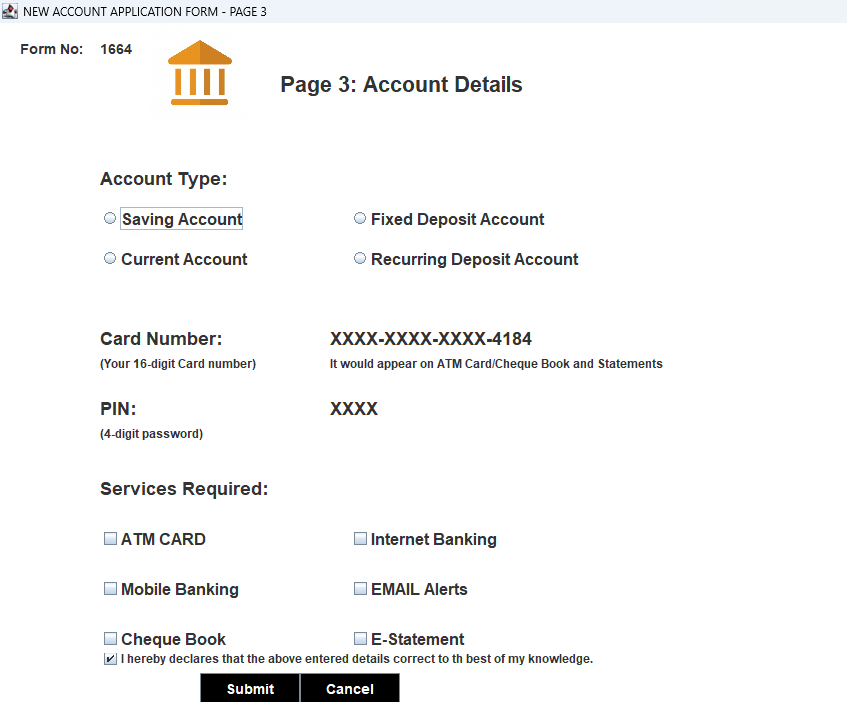
Spring's vast ecosystem and extensive documentation make it a popular choice for Java developers, offering solutions for diverse application development needs. It emphasizes best practices, reduces boilerplate code, and enhances productivity in Java-based enterprise development.

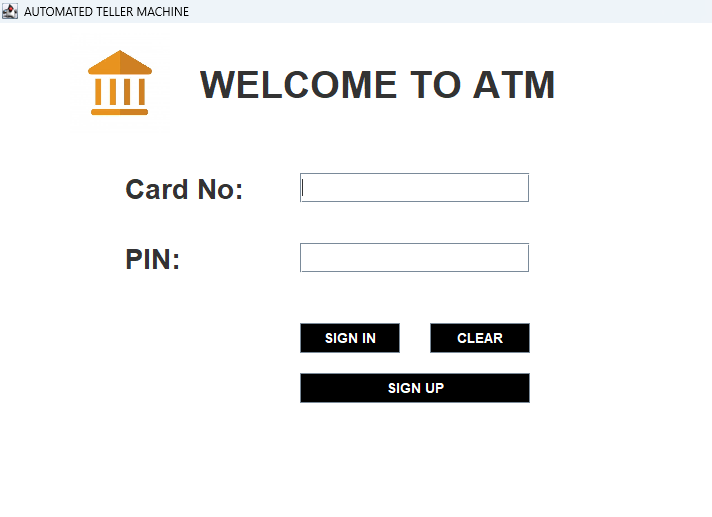
**Chapter 3 - Designing And Result Analysis**

****

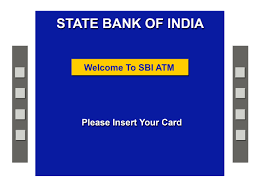
****







* 1. **Collecting Raw Data and working on it.**

****

**Chapter 4**

4.1 Conclusion

The ATM Simulation System represents a pivotal advancement in modern banking technology, offering a comprehensive suite of functionalities, robust security measures, and user-centric design to transform the banking experience. Throughout the feasibility analysis and exploration of various aspects, several critical observations and conclusions have been drawn

### **Functionality and User Experience:**

* The system's array of functionalities, including withdrawals, deposits, fund transfers, and balance inquiries, align closely with user expectations for efficient and convenient banking services.
* The intuitive user interface, coupled with secure authentication methods, ensures a seamless and secure user experience, fostering trust and reliability among users.

**4.2 Future Scope**

**1. Technological Advancements**:

**Biometric Authentication**: Integrate advanced biometric authentication methods (fingerprint, facial recognition) for heightened security and user convenience.

**AI and Machine Learning**: Explore AI-powered features for personalized user experiences, predictive analytics for user behavior, and fraud detection.

**IoT Integration**: Leverage IoT technologies to enable seamless connectivity between ATMs, mobile devices, and other banking channels.

**2. Enhanced User Experience**:

Multilingual Support: Implement multilingual interfaces to cater to a diverse user base, enhancing accessibility and user satisfaction.

**Voice Recognition**: Introduce voice-based commands and responses for an intuitive and hands-free user experience.

**Personalization Features:** Offer tailored recommendations and services based on user preferences and transaction histories.

**3. Security and Fraud Prevention:**

**Blockchain Integration**: Explore blockchain technology for enhanced transaction security, transparency, and resistance to tampering.

**Behavioral Analytics**: Develop systems that analyze user behavior to detect anomalies and prevent fraudulent transactions in real-time.

**Continuous Security Updates**: Stay updated with the latest security protocols and regularly implement patches to counter emerging threats.

**References**

* **For Java installation**

https://www.java.com/en/download/

* **For Database installation**

<https://www.mysql.com/downloads/>

https://www.mysql.com/products/connector/

* **Reference websites**
  + - [www.javatpoint.com](http://www.javatpoint.com)
    - [www.w3schools.com](http://www.w3schools.com)
    - <http://www.tutorialspoint.com/java/index.htm>