|  |  |  |
| --- | --- | --- |
| 1 | JAVA ACCESS MODIFIERS |  |
| 2 | AGILE METHODOLOGY |  |
| 3 | Method: |  |
| 4 | Static And Non-Static Methods |  |
| 5 | Static Blocks And Constructors |  |
| 6 | CONVERT JAVA CLASSES TO SPRINGBOOT |  |
| 7 | CONTROL STATEMENTS |  |
| 8 | POLYMORPHISM |  |
| 9 | MAVEN UPDATE |  |
| 10 | INHERITANCE |  |
| 11 | ABSTRACTION |  |
| 12 | IMPORTANT POINTS |  |
| 13 | TASK: SONAR QUBE QUALITY GATE CHECKING |  |
| 14 | OBJECT ORIENTED PROGRAMMING(OOPS) |  |
| 15 | ENCAPSULATION |  |
| 16 | CLONE THE PROJECT FROM GITHUB |  |
| 17 | STRINGS |  |
| 18 | STRINGBUFFER AND STRINGBUILDER |  |
| 19 | ARRAYS |  |
| 20 | COLLECTIONS |  |
| 21 | ADDING LOCAL CODE TO THE REMOTE REPOSITORY AT THE FIRST TIME |  |
| 21 | SPRINGBOOT |  |
| 22 | FUNCTONAL FLOW OF HOME INSURANCE |  |
| 23 | MySQL Data Definition Language (DDL) |  |
| 24 | JIRA AND CONFLUENCE |  |
| 25 | GITHUB & GIT |  |
| 26 | PROJECT DEVELOPMENT |  |
| 27 | PROJECT MIGRATION |  |
| 28 | for Loop and foreach Loop in Java | 7,8,9 |
| 29 | **do-while** | 8 |
| 30 |  |  |
|  |  |  |

# JAVA ACCESS MODIFIERS

In Java, access modifiers are keywords that set the access level for classes, variables, methods, and constructors. They determine whether other classes can use a particular field or invoke a particular method. Java provides four main access modifiers:

**Public** (public):

* Scope: Everywhere.
* Description: The member is accessible from any other class.

example:

public class Example {

public int publicVar;

public void public Method() {

// Code here

}

}

**Private** (private):

* Scope: Within the same class.
* Description: The member is accessible only within the class it is declared. It is the most restrictive access level.

example:

public class Example {

private int privateVar;

private void privateMethod() {

// Code here

}

}

**Protected** (protected):

* Scope: Within the same package and subclasses (even if they are in different packages).
* Description: The member is accessible within its own package and by subclasses. It provides a broader level of access than default access.

example:

public class Example {

protected int protectedVar;

protected void protectedMethod() {

// Code here

}

}

**Default** (no modifier, also known as package-private):

* Scope: Within the same package.
* Description: If no access modifier is specified, the member is accessible only within classes in the same package.

example:

public class Example {

int defaultVar;

void defaultMethod() {

/ / Code here

}

}

### Summary Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Modifier | Class | Package | Subclass | World |
| private | Yes | No | No | No |
| Default (none) | Yes | Yes | No | No |
| protected | Yes | Yes | Yes | No |
| public | Yes | Yes | Yes | Yes |

### Key Points

* **Private** members are the most restrictive, accessible only within the same class.
* **Default (package-private)** members are accessible only within the same package.
* **Protected** members are accessible within the same package and by subclasses in other packages.
* **Public** members are accessible from any other class, regardless of the package.

These access modifiers help in encapsulating the data, improving security, and managing the visibility of class members in Java.

# AGILE METHODOLOGY

Agile methodology involves several key ceremonies designed to facilitate communication, collaboration, and iterative progress within a team. which is methodology which helps us to project execution goals. Here’s a brief overview of each.

* **SPRINT PRE-PLANNING**
* **SPRINT PLANNING**
* **STANDUP CALLS**
* **SPRINT DEMO**
* **RETROSPECTIVE MEETING**
* **SPRINT BACKLOG**

SPRINT PRE-PLANNING:

Sprint pre-planning, unlike sprint planning, isn't an official part of the Scrum framework. It's an additional meeting some Agile teams use to improve the efficiency of the actual sprint planning session.

SPRINT PLANNING:

Sprint planning is a crucial meeting in Agile frameworks, particularly Scrum. It marks the beginning of a sprint, which is a short, time-boxed period (usually 1-4 weeks) where a development team works on a set of prioritized tasks.

**Participants:**

* **Scrum Master:** Facilitates the meeting, ensures everyone participates, and keeps the discussion focused.
* **Product Owner:** Represents the stakeholders and prioritizes the product backlog items.
* **Development Team:** The engineers, designers, and other team members who will do the actual work.

STANDUP CALLS:

* **Purpose:**
  + Briefly discuss what each team member accomplished yesterday.
  + Share what they plan to work on today.
  + Identify any blockers or roadblocks that might hinder progress.
* **Benefits:**
  + Improved team communication and awareness.
  + Early identification and resolution of roadblocks.
  + Enhanced focus and accountability for individual tasks.

SPRINT DEMO:

Developer or tester provide demo for the task to be assigned .Demo will be provided to the product owner after his confirmation completed to the status.

RETROSPECTIVE MEETING:

A retrospective meeting, often simply called a "retrospective," is a regular gathering in which a team reflects on their recent work and processes to identify what went well, what didn't, and how to improve in the future.

SPRINT BACKLOG:

A sprint backlog is a list of work items your team plans to complete during a project sprint. These items are usually pulled from the product backlog during the sprint planning session.

Method:

Method:

methods are blocks of code that perform specific tasks and are defined within a class.. Methods are used to perform certain actions, and they are also known as functions.

Syntax:

[access\_modifier] [optional\_specifier] return\_type method\_name([parameters]) {

// Method body

}

### Components of a Method

1. **Access Modifiers**:
   * public: Accessible from any other class.
   * protected: Accessible within the same package and subclasses.
   * default (no modifier): Accessible within the same package.
   * private: Accessible only within the defined class.
2. **Optional Specifiers**:
   * static: Belongs to the class, rather than instances of the class.
   * final: Cannot be overridden by subclasses.
   * abstract: Has no body and must be implemented in subclasses.
3. **Return Type**:
   * The data type of the value the method returns. Use void if the method does not return a value.
4. **Method Name**:
   * Should be a valid identifier and follow the naming conventions.
5. **Parameters**:
   * A comma-separated list of input parameters, each preceded by its data type.

Example of a Method:

public int addNumbers(int a, int b) {

return a + b;

}

# MAIN METHOD:

The main method in Java is the entry point for any standalone application. When you run a Java program, the runtime environment calls the main method, starting the execution of the application. Here's a detailed overview of the main method:

Syntax of the Main Method:

**public static void main(String[] args)**

### Components of the Main Method

1. **Access Modifier:** public
   * The main method must be public so that the Java runtime can access it.
2. **Static Modifier:** static
   * The main method is static so that it can be called without creating an instance of the class. The Java runtime calls this method without needing to instantiate the class.
3. **Return Type:** void
   * The main method doesn't return any value. It simply serves as the entry point to the program.
4. **Method Name:** main
   * The name main is fixed. The Java runtime environment looks specifically for this method name as the entry point.
5. **Parameter:** String[] args
   * The parameter args is an array of String objects. This array holds command-line arguments passed to the program when it is run. If no arguments are passed, the array is empty.

### Example of a Main Method:

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

If one method trying to access the value from another method steps:

In Java, if one method needs to access a value or call another method within the same class, you typically follow these steps:

1. **Ensure the method you want to access is non-private** (unless you're accessing it from within the same class).
2. **Call the method directly** if it's within the same class and accessible.
3. **Use the returned value** (if any) from the called method.

### Example: Accessing a Value from Another Method

Let's consider a class with two methods where one method accesses the value or calls the other method:

### Steps Explained

1. Define the Method that Provides a Value:
   * **public int add():** This method simply returns an integer value (30 in this case).
2. Define the Method that Accesses the Other Method:
   * **public void mul():** This method calls **add()** to obtain its return value and then prints it.
3. Call the Methods:
   * In the main method, create an instance of Example.
   * Call **mul()** on the instance, which in turn calls **add()** and prints the value.

Static And Non-Static Methods

In Java, methods can be categorized into two types: static methods and non-static methods (also known as instance methods). Understanding the differences between them is crucial for effective object-oriented programming in Java.

### Static Methods

1. **Definition**: Static methods belong to the class rather than any instance of the class. They can be called without creating an instance of the class.
2. **Declaration**: Static methods are declared using the static keyword.

Example:

public class Example {  
public static void staticMethod() {  
System.out.println("This is a static method.");  
}  
}

1. **Calling Static Methods**: They can be called using the class name directly.

Example:

Example.staticMethod();

1. **Accessing Members**: Static methods can access static variables and other static methods directly but cannot access instance variables and methods directly. They do not have a reference to this.

Example:

public class Example {

static int staticVariable = 10;

public static void staticMethod() {

System.out.println(staticVariable);

// System.out.println(instanceVariable); // This would cause an error

}

}

1. **Usage**: Static methods are often used for utility or helper methods that perform operations not dependent on instance variables.

### Non-Static Methods (Instance Methods)

1. **Definition**: Non-static methods belong to an instance of the class. They require an object of the class to be created before they can be called.
2. **Declaration**: These methods do not use the static keyword.

Example:

public class Example {

public void instanceMethod() {

System.out.println("This is an instance method.");

}

}

1. **Calling Instance Methods**: They are called using an instance of the class.

Example example = new Example();

example.instanceMethod();

1. **Accessing Members**: Instance methods can access both instance variables and methods, as well as static variables and methods.

Example:

public class Example {

int instanceVariable = 20;

static int staticVariable = 10;

public void instanceMethod() {

System.out.println(instanceVariable); // Access instance variable

System.out.println(staticVariable); // Access static variable

}

}

1. **Usage**: Non-static methods are used when behavior is dependent on the state of an object, that is, when instance variables or methods need to be accessed or modified.

### Key Differences

* **Belonging**: Static methods belong to the class itself, while non-static methods belong to instances of the class.
* **Access**: Static methods can only access static data and methods directly, while non-static methods can access both static and instance data and methods.
* **Invocation**: Static methods can be called without an instance, whereas non-static methods require an instance.
* **Reference to** this: Static methods do not have access to the this keyword, which refers to the current object, while non-static methods do.

Static Blocks And Constructors

In Java, static blocks (also known as static initialization blocks) are used to initialize static variables. A static block is executed when the class is loaded into memory, before the main method or any other static methods are invoked.

Here’s a quick overview of how static blocks work and how to use them:

### Syntax:

Static{

//code

}

**Example:**

public class MyClass {

// Static variable

static int myStaticVar;

// Static block

static {

// Initialization code

myStaticVar = 10;

System.out.println("Static block executed.");

}

public static void main(String[] args) {

System.out.println("Main method executed.");

System.out.println("myStaticVar: " + myStaticVar);

}

}

### Key Points

1. **Execution Order**: Static blocks are executed in the order they appear in the class, before any static methods or the main method are executed.
2. **Multiple Static Blocks**: You can have more than one static block in a class, and they will be executed in the order they are defined.
3. **Purpose**: Static blocks are typically used for static variable initialization, especially if the initialization is complex or requires logic that cannot be handled with a simple assignment.
4. **Execution Timing**: Static blocks are executed only once when the class is loaded into memory, which typically happens the first time the class is referenced.
5. **Use Cases**: They are useful for setting up static resources, such as establishing database connections or initializing complex static variables.

**Example with Multiple Static Blocks:**

public class MyClass {

static int myStaticVar1;

static int myStaticVar2;

static {

myStaticVar1 = 5;

System.out.println("First static block executed.");

}

static {

myStaticVar2 = 15;

System.out.println("Second static block executed.");

}

public static void main(String[] args) {

System.out.println("Main method executed.");

System.out.println("myStaticVar1: " + myStaticVar1);

System.out.println("myStaticVar2: " + myStaticVar2);

}

}

**Output:**

First static block executed.

Second static block executed.

Main method executed.

myStaticVar1: 5

myStaticVar2: 15

## **Constructors:**

Constructors in Java are special methods that are called when an object is instantiated. They are used to initialize objects. Here’s a comprehensive overview of constructors in Java:

### Types of Constructors

1. **Default Constructor**: Provided by Java if no constructor is explicitly defined. It has no parameters and initializes the object with default values.
2. **Parameterized Constructor**: Allows initializing an object with given values. It has parameters.

### Characteristics of Constructors

* The name of the constructor is the same as the class name.
* Constructors do not have a return type, not even void.
* They can have access modifiers to control the visibility (public, private, protected, or package-private).

**Syntax:**

class MyClass {

// Default constructor

MyClass() {

// Initialization code

}

// Parameterized constructor

MyClass(int param1, String param2) {

// Initialization code with parameters

}

}

# CONVERT JAVA CLASSES TO SPRINGBOOT

To convert a Java application that uses web.xml to a Spring Boot application, you need to follow several steps. Spring Boot applications typically use Java-based configuration rather than web.xml for configuration. Below is a general guide to help you with the conversion:

### 1. Create a Spring Boot Project

You can use Spring Initializer to create a new Spring Boot project. Select the necessary dependencies such as Spring Web.

**2. Replace web.xml Configuration**

In a traditional Java web application, web.xml is used to configure servlets, filters, and listeners. In Spring Boot, you can achieve the same configuration using Java annotations and configuration classes.

**Example web.xml:**

<web-app>

<servlet>

<servlet-name>exampleServlet</servlet-name>

<servlet-class>com.example.ExampleServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>exampleServlet</servlet-name>

<url-pattern>/example</url-pattern>

</servlet-mapping>

<filter>

<filter-name>exampleFilter</filter-name>

<filter-class>com.example.ExampleFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>exampleFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<listener>

<listener-class>com.example.ExampleFilter</listener-class>

</listener>

</web-app>

### 3. Main Application Class

Create a main application class with @SpringBootApplication annotation.

**Example:**

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

**4.Convert Your Existing Classes**

Make sure your servlets, filters, and listeners are compatible with Spring Boot. Here is an example for a servlet:

**Example:**

public class ExampleServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException { resp.getWriter().write("Hello, World!");

}

}

### 5. Update pom.xml

Ensure your pom.xml (or build.gradle if you are using Gradle) includes necessary Spring Boot dependencies.

#### Example pom.xml:

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Add other dependencies as needed -->

</dependencies>

### Summary

* Create a new Spring Boot project.
* Replace web.xml with Java configuration classes.
* Ensure your main application class is annotated with @SpringBootApplication.
* Migrate your existing servlet, filter, and listener classes.
* Update your build configuration (Maven/Gradle) to include Spring Boot dependencies.

By following these steps, you can convert your traditional Java web application to a Spring Boot application. If you need more specific guidance, please provide more details about your existing web.xml configuration and your Java classes.

CONTROL STATEMENTS

Control statements in Java are used to control the flow of execution in a program. They enable the program to make decisions, execute different blocks of code conditionally, or iterate over code blocks multiple times. Here’s an overview of the main types of control statements in Java:

### 1. Decision-Making Statements

These statements allow the program to choose different paths of execution based on certain conditions.

* if statement: Executes a block of code if the specified condition is true.

**Example:**

if (condition) {

// code to be executed if condition is true

}

* if-else statement: Executes one block of code if the condition is true, and another block if the condition is false.

**Example:**

if (condition) {

// code to be executed if condition is true

}

else {

// code to be executed if condition is false

}

* if-else-if ladder: Allows for multiple conditions to be checked sequentially.

**Example:**

if (condition1) {

// code to be executed if condition1 is true

}

else if (condition2) {

// code to be executed if condition2 is true

}

else {

// code to be executed if all conditions are false

}

* switch statement: Selects a block of code to execute from multiple options based on the value of an expression.

**Example:**

switch (expression) {

case value1: // code to be executed if expression equals value1 break;

case value2: // code to be executed if expression equals value2 break;

// you can have any number of case statements

default: // code to be executed if expression doesn't match any case

}

POLYMORPHISM

Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different classes to be treated as objects of a common superclass. In Java, polymorphism is achieved through method overriding and method overloading. It enables one interface to be used for a general class of actions, promoting flexibility and integration of objects in complex systems. Here’s a deeper look into polymorphism in Java:

### Types of Polymorphism

1. **Compile-Time Polymorphism (Static Polymorphism):**
   * Achieved by method overloading.
   * Determined at compile time.
   * Method overloading allows a class to have more than one method with the same name, but different parameters (different type, number, or both).

**Example:**

**EX1:**

class MathOperations {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

int add(int a, int b, int c) {

return a + b + c;

}

}

**EX2:**

public class Main {

public static void main(String[] args) {

MathOperations math = new MathOperations();

System.out.println(math.add(2, 3)); // Output: 5

System.out.println(math.add(2.5, 3.5)); // Output: 6.0

System.out.println(math.add(1, 2, 3)); // Output: 6

}

}

1. **Run-Time Polymorphism (Dynamic Polymorphism):**

* Achieved by method overriding.
* Determined at runtime.
* Method overriding allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

**Example:**

**EX1:**

class Animal {

void makeSound() {

System.out.println("Animal makes a sound");

}

}

**EX2:**

class Dog extends Animal {

@Override

void makeSound() {

System.out.println("Dog barks");

}

}

**EX3:**

class Cat extends Animal {

@Override

void makeSound() {

System.out.println("Cat meows"); '

}

}

**EX4:**

public class Main {

public static void main(String[] args) {

Animal myAnimal = new Animal();

Animal myDog = new Dog();

Animal myCat = new Cat();

myAnimal.makeSound(); // Output: Animal makes a sound

myDog.makeSound(); // Output: Dog barks

myCat.makeSound(); // Output: Cat meows

}

}

### Key Points

* **Method Overloading:**
  + Same method name, different parameter list.
  + Compile-time polymorphism.
* **Method Overriding:**
  + Same method name and parameter list as in the superclass.
  + Runtime polymorphism.
* **Dynamic Method Dispatch:**
  + The mechanism by which a call to an overridden method is resolved at runtime rather than compile-time.
  + Enables runtime polymorphism.
* **Upcasting:**
  + When a reference variable of the superclass refers to an object of the subclass.
  + Enables polymorphism

### Benefits of Polymorphism

* **Flexibility and Reusability:**
  + Allows for code to be more flexible and reusable.
  + Methods can be written that can process objects of different types and classes.
* **Maintainability:**
  + Reduces the complexity and increases the maintainability of the code.
  + Easier to introduce new classes with minimal changes to existing code.
* **Extensibility:**
  + Easier to extend the application with new functionality.

Polymorphism is a powerful concept that, when used correctly, can greatly enhance the flexibility and maintainability of your Java programs.

MAVEN UPDATE

When we perform Maven clean operation there will be target directory . It will be cleaned.

Maven install is one of maven command .which will be translating all java class files into dot(.) class files. We should be access them by generating var or Jar files.

**How to update maven project:**

1. right click on project-> Update maven project
2. Run as-> java application

**TASK:**

**Error:(MAIN METHOD ERROR)**

Exception in thread

“main java.lang.NoClassDefFoundError:org/Springframework/core/io/support/SpringFactoriesLoader” in Spring Tool suit

**Solution:**

This error because of dependencies mess up. So, We have to remove the nested dependencie

# INHERITANCE

**INHERITANCE:**

Extending or inheriting one class to another class is called inheritance. We can achieve inheritance with help of extends or implements keywords.

We can use extends keyword Class to Class ( or ) Interface to Interface.

We can use implements keyword Class to Interface.

**Multi-level inheritance:**

Assume we have 3 classes A,B,C . Class B extends A, Class C extends B.

* Here class B is extending class A features directly.
* Here class C is extending class B features directly. Here class C is extending class A features indirectly.

**Multiple inheritance:**

Extending or Inheriting multiple class features into another class is nothing but multiple inheritance. Java does not support the multiple inheritance.

**Explanation(Multiple inheritance):**

Assume we have 3 classes A,B,C. A extends B,C.

* Here B & C has same method which is “add” while creating object for “A” and calling to add method. JVM will confuse which method has to be used.

ABSTRACTION

**ABSTRACTION:**

Abstraction is nothing but hiding the implementation . We can achieve the abstraction using the abstract class and interface.

**INTERFACE:**

Blueprint of the class is nothing but interface which you have only method declaration and doesn’t have the method implementation .We used to called them as abstract class.

**\*\*\*ABSTRACT METHOD:**

The method which is having the abstract keyword and doesn't have the implementation is called abstract method.

**\*\*\*Note1:**

When we write abstract method in interface we no need to add the abstract keyword As interface includes abstract keyword for all the methods.

**\*\*\*Note2:**

If a class implements the interface all the methods which is present inside the interface has to be implemented.

**ABSTRACT CLASS:**

The class which is having the abstract keyword is called abstract class. It contains both abstract and non-abstract methods. Normal classes will not allow to write abstract methods inside the class.

**\*\*\* Note:**

If a class extends the abstract class it is mandatory to implement all abstract methods in the implemented class.

# IMPORTANT POINTS

**Final keyword:**

* Once you declare a variable as final we cannot modify the variable.
* Once you declare a method as final we cannot override the method.
* Once you declare a class as final we cannot extend the class.

**Non-static method:**

* You can call static or non-static methods.
* If you wish to call non-static methods from static, it will not accept, it allows only static methods.

**Static block:**

* The moment you keeping the code inside the static block that content is executed.
* Static block is more powerful, it execute the before main method.

**else:**

* If you don't have if condition , you can not write else.

**else-if:**

* To execute the statements based upon different conditions.
* You can write multiple else-if statements.

**1.can we overload the static method?**

a. static methods can be overloaded.

**ABSTRACT METHOD:**

The method which is having the abstract keyword and doesn't have the implementation is called abstract method.

* **Note1:**

When we write abstract method in interface we no need to add the abstract keyword As interface includes abstract keyword for all the methods.

* **Note2:**

If a class implements the interface all the methods which is present inside the interface has to be implemented.

**ABSTRACT CLASS:**

* **Note:**

If a class extends the abstract class it is mandatory to implement all abstract methods in the implemented class

**METHOD OVER LOADING:**

In method overloading return type also different. We can use same return type also based upon the requirement.

**METHOD OVER RIDING:**

1. **why we are not able to achieve method overriding in single class?**

A. When we write same method names and same arguements .it error out during the compile-time because JVM don't know which method has to be called.

**\*\*\***

subclass obj 1 = new superclass(); -----> is not possible

superclass obj 2 = new subclass(); -----> is possible( it is method overriding concept)

TASK: SONAR QUBE QUALITY GATE CHECKING

**SONAR QUBE:**

Which is popular static code analysis tool. which will help us to identify the different kind of issues from the applications. Mostly we will use to find out below issues.

1. Out dated dependencies
2. Conflicting dependencies
3. Unused dependencies
4. Licencing issue

**1.Out dated dependencies:**

Whenever we use outdated dependencies which we can lead the security vulnerabilities and compatibility issues.

**2.Conflicting dependencies:**

Different parts of projects required different version of the same dependencies. which also leads the dependencies conflicts. same dependencies but different versions.

**3. Unused dependencies:**

Which we are not using we have to removing dependencies.

**4. Licencing issue:**

Couple of dependencies are licensed issues if the license are expired or ready to expired sonar will identified or noticed the issue.

**Project Environment:**

1. Development Environment
2. Testing
3. UAT( User Acceptance Testing)( Pre-production)
4. Production

OBJECT ORIENTED PROGRAMMING(OOPS)

**OOPS:**

There are four types oops principles

1. Inheritance
2. Polymorphism
3. Abstraction
4. Encapsulation

**1.Inheritance**:

Extending or inheriting one class to another class is called inheritance. We can achieve inheritance with help of extends or implements keywords.

We can use extends keyword Class to Class ( or ) Interface to Interface.

We can use implements keyword Class to Interface.

**Multi-level inheritance:**

Assume we have 3 classes A,B,C . Class B extends A, Class C extends B.

* Here class B is extending class A features directly.
* Here class C is extending class B features directly. Here class C is extending class A features indirectly.

**Multiple inheritance:**

Extending or Inheriting multiple class features into another class is nothing but multiple inheritance. Java does not support the multiple inheritance.

**Explanation(Multiple inheritance):**

Assume we have 3 classes A,B,C. A extends B,C.

* Here B & C has same method which is “add” while creating object for “A” and calling to add method. JVM will confuse which method has to be used.

**2.Polymorphism:**

One interface will have many implementations is called polymorphism. There are two different ways

1. Method overloading
2. Method overriding

* **Method overloading:**

Method overloading can be achieved withinthe class. We will have the two or more than two methods. Which is having the same method name and different implementations. We will have different inline parameters.

\*\* We have to write same method name ,same functionality with different parameters

**Q1:** Can we overload the main method?

**A:** You can overload the main method with different parameters, but these methods wont be called by JVM as the entry point.

**Q2:** Can we overload the static method?

**A:** Static methods can be overloaded.

**Q3:** Can we overload the final method?

**A:** Yes, We can do it.

* **Method overriding:**

We have same method name, same inline parameters and different implementation.

* We should have two classes.
* Method overriding is mechanism of calling the overriding new method or function whatever the function that we have to call from the child class.

**Q1:** Can we override static method?

**A:** Yes, We can override the static methods.

**Q2:** Can we override final methods?

**A:** We cannot override final method.

**3.Abstraction:**

Abstraction is nothing but hiding the implementation . We can achieve the abstraction using the abstract class and interface.

**INTERFACE:**

Blueprint of the class is nothing but interface which you have only method declaration and doesn’t have the method implementation .We used to called them as abstract class.

**\*\*\*ABSTRACT METHOD:**

The method which is having the abstract keyword and doesn't have the implementation is called abstract method.

**\*\*\*Note1:**

When we write abstract method in interface we no need to add the abstract keyword As interface includes abstract keyword for all the methods.

**\*\*\*Note2:**

If a class implements the interface all the methods which is present inside the interface has to be implemented.

**ABSTRACT CLASS:**

The class which is having the abstract keyword is called abstract class. It contains both abstract and non-abstract methods. Normal classes will not allow to write abstract methods inside the class.

**\*\*\* Note:**

If a class extends the abstract class it is mandatory to implement all abstract methods in the implemented class.

**SENARIOS:**

* \*\*\* one interface can extends another interface
* // A is one interface B is another interface which extends A ,   
  // whatever the class implementing interface B , that should implements methods which present in A , B interfaces
* //\*\*\* if an abstract class implements interface which no need to have the implementation ,  
  //if you wish to do also we can do that.
* //\*\*\* if an abstract class can only extends the abstract class  
  //abstract class which no need to have the implementation ,  
  //if you wish to do also we can do that
* // \*\*\*when abstract class implements interface ,if you wish to do implementation we can do that,  
  //else u can ignore it
* //\*\* single normal can can implements multiple interfaces , multiple inheritance
* //\*\* A class can extends abstract and implements interfaces at the same time ,  
  //That class should implements all the methods present in the both interface and abstract class

**4.Encapsulation:**

Holding the data into single entity is nothing but encapsulation.

**How to achieve:**

we can achieve Data (or) Model (or) POJO class . The class which is having the data members and does not have the declaration. POJO stands for Plain Old Java Object

* We should have the setters and getters for the data members.
* [this.id](http://this.id/) ---->” this” is one of the reserved word in java. which helps to set the value to the data members (with in the class only).

ENCAPSULATION

**ENCAPSULATION:**

Holding the data into single entity is nothing but encapsulation.

**How to achieve:**

we can achieve Data (or) Model (or) POJO class . The class which is having the data members and does not have the declaration. POJO stands for Plain Old Java Object

* We should have the setters and getters for the data members.
* [this.id](http://this.id/) ---->” this” is one of the reserved word in java. which helps to set the value to the data members (with in the class only).

CLONE THE PROJECT FROM GITHUB

* If you wish to add repository navigate to Eclipse IDE

Window → Show view → other → root repository

**Remote Repository:**

The code which is accessible entire team that is called remote repository

**Local Repository:**

Whatever the code that implemented by individual developer is called local repository. when we perform commit and push that code move to remote repository

**steps of clone the project from github:**

* Click new branch in GitHub and create the new branch.
* Click the code and copy the url.
* Then we open the Eclipse IDE click Window → Show view → other → Git → Git repository → open

**CLONE OLD PROJECT FROM GITHUB:**

* Go to code option the click download zip.
* Go to download and the extract the download file.
* Moved the extracted file into Eclipse workspace folder.
* Go to Eclipse then following below steps

Click File → import → Maven → Existing maven → Root Repository(browse the file space) → finish

**NOTE:**

\*\*\* Whenever you request for any of the software installation we use service tickets. Service tickets may be

1. service now tool

2. Wees tool

\*\*\* If the project name same, IDE will not allow to create (or) import the same project

# STRINGS

**Primitive datatype:**

The datatype which is not allowed to create object is called primitive datatype. All primitive datatypes we have to declare with literal approach.

**STRING:**

String is immutable object in java and which is non-primitive datatype.

* Declaration by using the new keyword.

String s= new String(“Hello”);

* Only non-primitive datatypes can be declare by using new keyword.

**Q1: What is immutable?**

**A:** Once you declare value we cannot modify the value is called immutable object.

\*\*\* “==” is one of the java operator which helps to compare two objects at the same time it compares memory location of the objects.

\*\*\* We can declare a string in two ways

1. **Literal approach:** While declaring literal approach the value will store it into string pool memory.
2. **New Keyword:** While declaring new keyword is stored in heap memory.

**Methods:**

There are some of methods in the string class which can reduce the implementation logic from the developer.

1. **charAt:**

It is one of the pre-defined method in string which needs integer (index) as inline parameter and it return characters.

* Index always starts from zero.

**Ex:** String s=”hello”;

s.charAt(1);

**output: e**

1. **concat:**

It is pre-defined method which helps to combine two or more strings.

**Ex:** s.concat(“world”);

**output: hello world**

1. **contains:**

It is pre-defined method which helps to check given character in present string or not. which needs input as char (or) string and it returns boolean value true or false.]

1. **endsWith:**

It is pre-defined method in string which helps to check given string is ending with mentioned char or String . It returns boolean value true or false.

1. **equals:**

It is pre-defined method in string which helps to compare two strings is that strings same or not. It returns boolean value true or false. It is case sensitive.

**Note: Diff b/w “==” and “equals” method**

“==” will compare objects at the same time will check memory location also. But equals method only check both objects same or not.

1. **equalsIgnoreCase:**

It is pre-defined method in string which helps to check given strings are equal or not . And it will exclude case sensitivity. It returns boolean value(true or false).

1. **isBlank:**

It is pre-defined method in string which does not need any inline arguements. It returns true when the string is blank otherwise it returns false when the string is not blank.

1. **isEmpty:**

It is pre-defined method in string. when the string is empty it returns true, when the string is not empty it returns false. Inline arguements not applicable. It checks spaces also.

1. **length:**

It is pre-defined method in string which helps to check given string length which does not need any inline arguements. It returns integer.

\*\*\* length always starts from **“1”**

1. **matches:**

It is pre-defined method in string which helps to check given regular expressions matches or not. If it matches returns true, not matches it returns false.

1. **replace:**

It is pre-defined method in string which helps to replace the new characters in the given string . here we have two inline arguements first represents which one replaces, second represents with whom replaced. It returns updated string.

1. **startsWith:**

It is pre-defined method in string which helps to check given string starts with given char (or) not. We need inline arguements String or char. It returns boolean value.

1. **subString:**

It is pre-defined method in string which helps to create substring .

Inline arguements: subString(4) → It will be the starting index of given string.

subString(4,9) → first will be the starting index, and second will be the ending index. It returns string.

1. **toCharArray:**

It is pre-defined method in string which helps to convert string into char array. Inline arguements not applicable. It will be return like this {“h“, “e“ , “l“ , “l“ , “o” }.

1. **toUpperCase:**

It is pre-defined method in string which helps to given string convert to upper case. It returns string . Inline arguements not applicable.

1. **toLowerCase:**

It is pre-defined method in string which helps to given string into lower case. Inline arguements not applicable. It returns string.

1. **trim:**

It is pre-defined method in string which helps to remove spaces both sides in strings starting and ending positions. Inline arguements not applicable. It returns string.

1. **split:**

It is pre-defined method in string which helps to split given string to string array. Inline arguements is regular expression and return type is string array.

# STRINGBUFFER AND STRINGBUILDER

StringBuffer and StringBuilder are mutable. Mutability nothing but modification. Which will helps us to the string modification.

\*\*\* StringBuffer , StringBuilder and String these three are the final classes in java.

**DIFF B/W STRINGBUFFER AND STRINGBUILDER:**

StringBuffer is synchronized and StringBuilder is not synchronized.

**Synchronized:**

Synchronized is one of the java reserved word which helps to achieve the synchronization.

**Synchronization:**

It is a process of allowing single task to be completed then it allows another task to perform.

**DRAWBACKS:**

Performance wise **StringBuffer** is very slow . The reason why it is synchronized and it is safe.

**StringBuilder** is faster because it is not synchronized and it is not safe.

**Append Method:**

Which helps to manipulate and assign values to the StringBuffer and StringBuilder.

**StringBuffer Methods:**

charAt

subString

length

replace

reverse(only used in StringBuffer and StringBuilder)

# ARRAYS

* Arrays are fixed size datatypes.
* Arrays will help us to group of multiple data together but it allows only one of the type of datatypes. It may be string , int, char whatever it is.

**syntax1:**

int [] arr = new int [10];

“ arr” is represent integer array and it contains 10 integers.

**syntax2:**

String [] arr1={ “ganesh”, “govind”, “ bujji” ,”chinna” }

This represents the string array which is also fixed size and we cannot add other indexes.

* arr[0]=9; → which helps to store the value in the 0th index.

**UPDATE:**

arr1[0] =”gani”; → whatever the latest value to be assigned in existing value.

# FOR AND FOREACH LOOPS

There are two ways of retrieving

1.For Loop

2.Foreach Loop

For Loop:

For loop is one of the control statement which will used to iterate the objects.

Syntax:

for(initialisation; condition; updation){

//statements

}

Ex:

for(i=0; i<=10; i++)

{

system.out.println(i);

}

output: 0 1 2 3 4 5 6 7 8 9 10

Foreach Loop:

Follow below steps for retreiving the array or collection

1. Which type of object are using it
2. Object name
3. Whom you are iterating

Syntax:

for(Datatype variable: array or collection variable)

{

// statements

}

Ex:

int [ ] arr = {5,6,23,84,56};

for(int obj : arr)

{

system.out.println(obj);

}

output: 5 6 23 84 56



Add labe

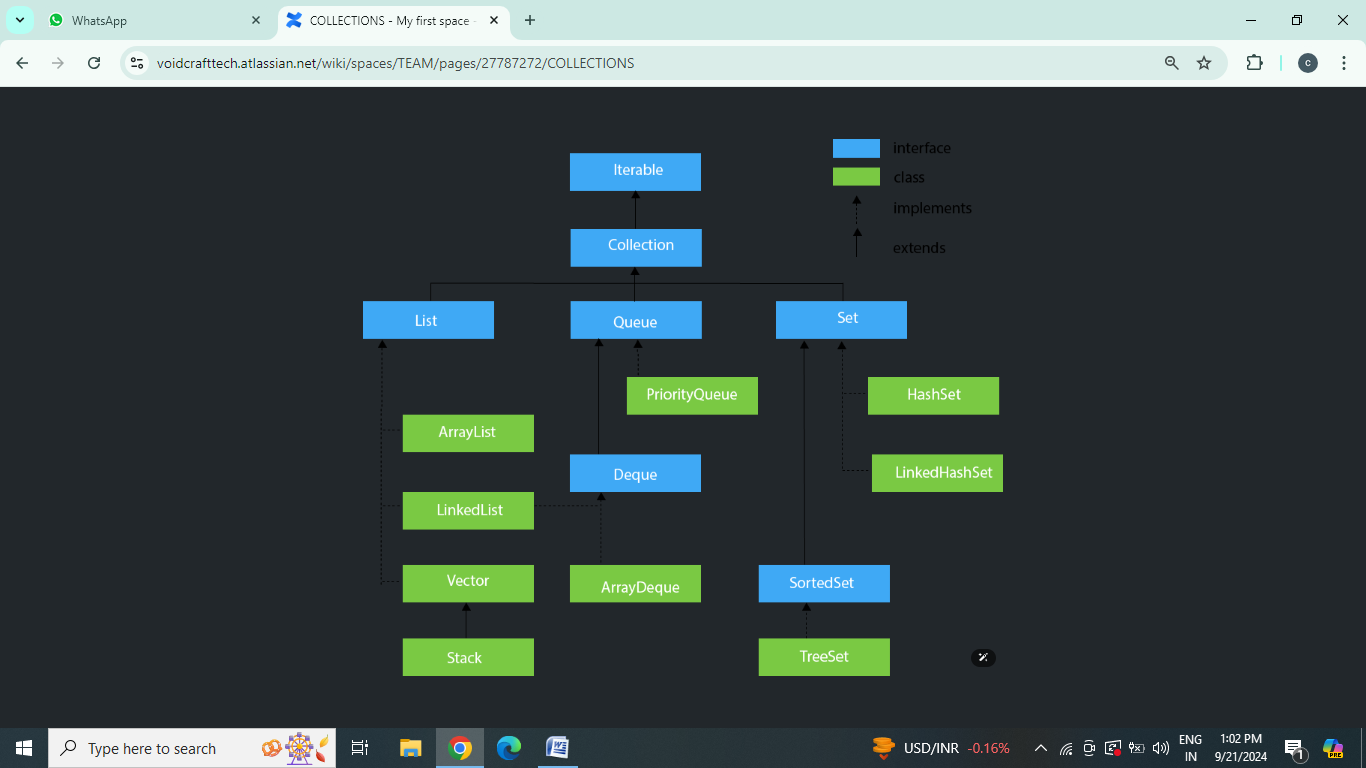
# COLLECTIONS

COLLECTIONS:

Collections are used to the data manipulation. We perform CURD operations is called data manipulation.

COLLECTION:

Collection is an interface. It is used to data manipulation.



Iterable:

Iterable is one of the interface which belongs to java.lang package. this interface help us to extract the object.

Iterator:

Iterator is an interface which is having the only pre-defined method helpful to perform collection related operations.

Collection:

Collection is interface which is having java.util package which is having all pre-defined methods add, remove, clear etc. which helps to data manipulation.

List:

List is one of the interface which helps to store multiple values ,duplicate values , null values etc.

Queue:

Queue is one of the interface which helps to store or keep the elements in FIFO(First In First Out) model.

Set:

Set is an interface which helps to store only unique values.

ArrayList:

* ArrayList is the class which implements the List Interface.
* As ArrayList implements the list which store multiple values, duplicate values, null values.
* As implementing collection directly, which helps to perform data manipulation.
* which is is internally uses the array data structure.
* which is follows the index based process for storing the elements
* ArrayList used to perform the frequent retrieve or fetching operations. It wouldn’t recommand for deleting or removing operations.
* \*\* which is not good for delete operation the reason because why you delete an element, in the background all other elements indexes will be modified.
* which is index based and the method retrieve the element is get whenever we provide index it pick of the element which present in the given index.

LinkedList:

* LinkedList is the class which implements the List Interface.
* As LinkedList implements the list which store multiple values, duplicate values, null values.
* As implementing collection directly, which helps to perform data manipulation.
* which is internally uses the double linked list and which is node based.
* \*\* which will be perform frequently delete and update operations.
* It is good for deleting. the reason why when we remove the element the impact will be removed element previous node and next node will be modified.
* It is not good for retrieval the reason why we provide the index through get method that has to converted into the node and return the element.

Vector:

* Vector is the class which implements the List Interface.
* As Vector implements the list which store multiple values, duplicate values, null values.
* As implementing collection directly, which helps to perform data manipulation.
* \*\* vector also internally uses array data structure. It is synchronized.
* We will be used to perform the frequent create operation because which is increases 100% default size where LinkedList and ArrayList increases 50% of default size.

# ADDING LOCAL CODE TO THE REMOTE REPOSITORY AT THE FIRST TIME

* Open the GitHub.
* We create branch in GitHub.
* click on code and copy the URL.
* Open GitHub desktop tool.
* Add → Clone Repository
* Select URL and Paste URL
* Click Clone.
* Main branch( is automatically cloned) ---> take folder place
* take the project from local repository ,which we have worked.
* paste that project from files/explore in GitHub folders.
* clover ,find bugs ,docker ----after creation → go to properties → go to the URL → copy these three properties.
* Go to project folder place and paste above three properties in this folder and delete other unused folders.
* then import this project in IDE.
* Build the project

1. open file( which taken from working file).
2. Do changes.

* CI

1. add pipeline properties.
2. add container pipeline.

* After committing raise pull request(PR).
* After approved → merge PR.

**JENKINS:**

Whatever the manual operation that we doing to make the build we are using **JENKINS**.

**SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC):**

* **JIRA:** user story → detailed description → requirement gathering.
* **ANALYSIS:** Youhave to understand.
* **DEVELOPMENT:** GIT → create new feature.
* Write your code
* PR - Raise ---> Merge.
* Local Testing
* **DEPLOYMENT:**

STAGES: 1. DEVELOP 2. TESTING 3. PRODUCTION

1. JENKINS
2. TRIGGERS → COMPILE , DEV

# SPRINGBOOT

1There are three types of applications

1. Standard alone applications
2. Web based applications
3. Enterprise applications

**1.Standard alone applications:**

The applications which we can access within the desktop is called the standard alone applications.

**2.Web based applications:**

The applications which we can access within the internet. we use web based technologies like servlets, sturts, spring mvc, spring boot. which helps to build the web applications.

**3.Enterprise applications:**

The applications which we can access within the internet large scale web applications.

Ex: IDFC , FLIPKART , AMAZON ETC.

**SPRINGBOOT(Why we use Spring Boot)**

* It is faster.
* Configuration independent
* In-Built server.

# 

**Project Flow(Web-Application):**

* \*\* Web application , First search for web.xml file.
* web.xml file send request to the all classes.

**Spring Boot(web - application)**

* In Spring Boot, We don't have web.xml file. It have spring container, which is a class which is having the main method that will be taking care of doing all the activities done by web.xml file.
* In any web based applications web.xml file is first initializer. The moment this will take care of forwarding the request to the java class.

# 

**How to create Spring Boot applications(ways):**

1. Navigate to start.spring.io
2. Maven → dependencies spring boot
3. You can use STS → create spring boot project
4. SCLI → spring Boot command line interface

**Steps to create using** [**start.spring.io**](http://start.spring.io/)

1. Maven – build tool
2. Spring Boot version
3. Group – on which package you have to create main class
4. ArtifactId – Project name
5. Packaging – you have to packaging which jar/war
6. java version – which version you want
7. Dependencies – you have to provide the all dependencies based upon technologies you are using

**Ex: we have used web i.e, we have to add Springweb , Lombok etc.**

1. click on generate.

**REST Methods:**

1. **POST:** New creation/new version
2. **PUT:** Update / Modify
3. **GET:** Retrieval
4. **DELETE:** If you wish to delete service
5. **PATCH:** Partial operations

**POSTMAN: To perform the backend testing.**

**Annotations:**

1. **@RestController:** Whichis combination of @Controller + @ResponseBody

Which converts simple java file to web class, which can access anytime

1. **@Controller :** which makes this java class as web component
2. **@ResponseBody:** which helps to send response back to the client.
3. **@PostMapping(“/saveemp”):** Which is having the resource name as “/saveemp” whenever you are calling from the client/frontend . It will map to the give resource.

PostMapping used to save the data.

1. **@RequestBody:** which carries the data sending from the client.

# FUNCTONAL FLOW OF HOME INSURANCE

### 1. Start with the Purpose

* "The Hartford house owner's insurance web app is designed to help users easily manage their home insurance policies. From getting a quote to filing claims, the app makes the process simple, efficient, and user-friendly."

### 2. Explain the Core Features in a Logical Flow

* **User Registration and Login:**  
  "First, users can create an account by providing basic information like their name, email, and phone number. Existing users can log in to manage their policies."
* **Eligibility Check:**  
  "Once logged in, users enter their ZIP code to check if they are eligible for home insurance in their area. The app uses ZIP code data to ensure coverage availability."
* **Personal and Property Details:**  
  "If eligible, users proceed to fill out personal information such as their name and address. We’ve integrated Google Maps API to help with accurate address entry. On the second page, users provide property details like the type of house (single-detached, duplex, etc.), its size, and construction type (wood, brick, etc.)."
* **Premium Calculation:**  
  "Based on the property details, the app calculates the insurance premium. Factors like house type, construction materials, and ZIP code risk affect the final premium. Discounts, such as AARP member benefits, are also factored in."
* **Policy Customization:**  
  "Users can further customize their policy by selecting different coverage levels (basic, standard, or premium) and adding additional coverage for valuables."
* **Payment Options:**  
  "Users can choose from different payment plans—monthly, quarterly, or yearly. They can pay using credit cards, bank transfers, or other options."
* **Managing the Policy:**  
  "Once the policy is activated, users can view their policy details, renew it when it’s close to expiring, or make changes like updating their contact information or adjusting their coverage."
* **Filing Claims:**  
  "In case of damage or loss, users can file claims by providing incident details and uploading supporting documents like photos. They can also track the status of their claims online."
* **Customer Support:**  
  "The app also offers easy access to customer support through live chat, phone, or a FAQs section."

### 3. Highlight the Technology & User Experience

* "The app is designed with the user in mind, ensuring the process is intuitive. Integrating Google Maps API for address entry reduces user errors, and the ZIP code check helps with eligibility verification upfront, saving users time. It’s not just about functionality but also about improving the user experience."

### 4. Mention Scalability and Efficiency

* "The system is scalable, meaning it can handle a growing number of users without compromising performance. It’s also efficient because it automates processes like premium calculation and claim tracking, which would otherwise take more time if done manually."

### 5. Wrap Up with a Real-World Benefit

* "Overall, the app empowers users by giving them control over their insurance policies in an accessible and efficient way. It simplifies what can be a complex process and saves time for both the company and the customers."

MySQL Data Definition Language (DDL)

Certainly! Let’s explore the **Data Definition Language (DDL)** commands in MySQL. DDL is responsible for defining and managing the structure of database objects. These commands allow you to create, modify, and delete database structures. Here are some common DDL commands along with examples:

1. **CREATE**:
   * The CREATE command is used to create new database objects such as tables, indexes, views, and more.
   * Example: Creating a simple table named Students:

**SQL**

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Age INT

);

1. ALTER:
   * The ALTER command modifies the existing structure of database objects.
   * Example: Adding a new column named Email to the Students table:

SQL

ALTER TABLE Students

ADD COLUMN Email VARCHAR(100);

1. **DROP**:

* The DROP command deletes entire database objects (tables, indexes, etc.).
* Example: Dropping the Students table:

DROP TABLE Students;

**TRUNCATE**:

* The TRUNCATE command removes all records from a table, but retains the table structure.
* Example: Truncating the Students table:

TRUNCATE TABLE Students;

**COMMENT**:

* The COMMENT command allows you to add comments to the data dictionary for documentation purposes.
* Example: Adding a comment to the Students table:

COMMENT ON TABLE Students

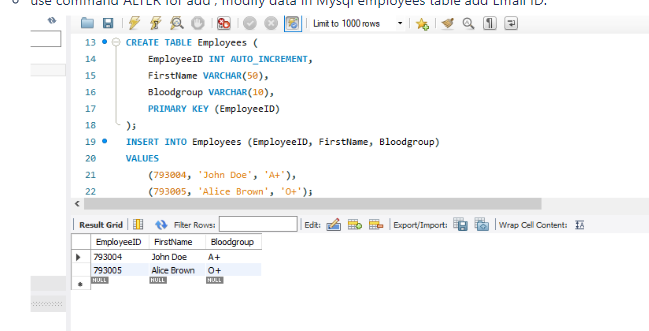
IS 'Contains student information';

**RENAME**:

* The RENAME command changes the name of an existing object.
* Example: Renaming the Students table to ClassMembers:

RENAME TABLE Students TO ClassMembers;

* Remember to adapt these examples to your specific use case. DDL commands are essential for defining your database schema effectively. If you have any further questions or need assistance, feel
* My project **create table and insert data office employee , for user command create ,**
* use command ALTER for add , modify data in Mysql employees table add Email ID.



JIRA AND CONFLUENCE

JIRA:

Jira is a software application developed by Atlassian that allows teams to track issues, manage projects, and automate workflows. It's particularly popular for Agile project management and software development, but its flexibility allows it to be used for various tasks across different teams.

* **Open:** The task is waiting to be started. It's like a sticky note on your "To Do" list.
* **In Progress:** Someone on your team has begun working on the task. The sticky note has moved to the "In Progress" section.
* **Under Review:** The task is finished but needs a check-up. The sticky note is now with someone for review, like a teacher grading a test.
* **Done:** The review is complete, and the task is finished! The sticky note has reached the "Done" pile. You can celebrate!
* **Cancelled:** The task won't be done anymore. Maybe it's no longer needed. The sticky note gets crumpled and thrown away (virtually, of course).
* **Rejected:** The task was reviewed but needs revisions. The sticky note goes back to "In Progress" with feedback for improvement. It's like getting your test back with a red mark and some notes.

ISSUE BOARD:

* **Epic:** Represents a large body of work or a major feature to be developed. Imagine it as a big chapter in your project book.
* **Story:** A user-centric description of a desired functionality. Think of it as a user's perspective on "what" needs to be achieved.
* **Task:** A specific action required to complete a user story. Break down your stories into smaller to-do list items.
* **Sub-task (optional):** An even smaller chunk of work that contributes to a task. Great for further granular breakdown.
* **Bug:** A problem or error in the software that needs to be fixed. This is like a red "Uh oh!" flag.

Why JIRA software:

JIRA is an all-in-one solution for multiple software and team related use cases.

* Creating backlogs
* Creating sprints
* Creating tasks
* Updating status on the board
* Managing releases
* Code integration
* Managing workflows

CONFLUENCE:

Confluence is a team workspace tool designed for collaboration and knowledge sharing. It's like a central hub where teams can:

* **Create and share documents:** Think of wikis, meeting notes, project plans, and more.
* **Work together on projects:** Discuss ideas, assign tasks, and track progress.
* **Organize information:** Keep everything in one place, easily accessible for everyone.

It's a popular choice for Agile teams to improve communication, boost knowledge sharing, and streamline project management.

Here are some key benefits of using Confluence:

* **Improved collaboration:** Breaks down silos and fosters a collaborative work environment.
* **Enhanced knowledge sharing:** Centralizes information and makes it easily accessible to everyone.
* **Increased productivity:** Reduces time spent searching for information and streamlines communication.
* **Better decision-making:** Provides a platform to share ideas, gather feedback, and make informed decisions.
* **Scalability:** Adapts to growing teams and keeps information organized as projects evolve.

If you're looking for a tool to improve collaboration, knowledge sharing, and project management within your team, Confluence is a strong contender.

3

* **Open:** The task is waiting to be started. It's like a sticky note on your "To Do" list.
* **In Progress:** Someone on your team has begun working on the task. The sticky note has moved to the "In Progress" section.
* **Under Review:** The task is finished but needs a check-up. The sticky note is now with someone for review, like a teacher grading a test.
* **Done:** The review is complete, and the task is finished! The sticky note has reached the "Done" pile. You can celebrate!
* **Cancelled:** The task won't be done anymore. Maybe it's no longer needed. The sticky note gets crumpled and thrown away (virtually, of course).
* **Rejected:** The task was reviewed but needs revisions. The sticky note goes back to "In Progress" with feedback for improvement. It's like getting your test back with a red mark and some notes.

ISSUE BOARD:

* **Epic:** Represents a large body of work or a major feature to be developed. Imagine it as a big chapter in your project book.
* **Story:** A user-centric description of a desired functionality. Think of it as a user's perspective on "what" needs to be achieved.
* **Task:** A specific action required to complete a user story. Break down your stories into smaller to-do list items.
* **Sub-task (optional):** An even smaller chunk of work that contributes to a task. Great for further granular breakdown.
* **Bug:** A problem or error in the software that needs to be fixed. This is like a red "Uh oh!" flag.

Why JIRA software:

JIRA is an all-in-one solution for multiple software and team related use cases.

* Creating backlogs
* Creating sprints
* Creating tasks
* Updating status on the board
* Managing releases
* Code integration
* Managing workflows

CONFLUENCE:

Confluence is a team workspace tool designed for collaboration and knowledge sharing. It's like a central hub where teams can:

* **Create and share documents:** Think of wikis, meeting notes, project plans, and more.
* **Work together on projects:** Discuss ideas, assign tasks, and track progress.
* **Organize information:** Keep everything in one place, easily accessible for everyone.

It's a popular choice for Agile teams to improve communication, boost knowledge sharing, and streamline project management.

Here are some key benefits of using Confluence:

* **Improved collaboration:** Breaks down silos and fosters a collaborative work environment.
* **Enhanced knowledge sharing:** Centralizes information and makes it easily accessible to everyone.
* **Increased productivity:** Reduces time spent searching for information and streamlines communication.
* **Better decision-making:** Provides a platform to share ideas, gather feedback, and make informed decisions.
* **Scalability:** Adapts to growing teams and keeps information organized as projects evolve.

If you're looking for a tool to improve collaboration, knowledge sharing, and project management within your team, Confluence is a strong contender.

GITHUB & GIT

GitHub:

GitHub is remote repository which will have all the projects. GitHub is a web-based platform for software development that lets developers store, track, and collaborate on code. It's essentially a combination of two things:

* Version control: This refers to the ability to track changes made to code over time. This allows developers to see how the code has evolved, revert to previous versions if necessary, and collaborate on changes without accidentally overwriting each other's work. Git is the most widely used version control system, and GitHub provides a user-friendly interface for working with it.
* Code sharing: GitHub allows developers to share their code with others publicly or privately. This is a fundamental aspect of open-source software development, where code is freely shared and anyone can contribute to it. But GitHub is also used extensively by private companies to share code among their teams.

In addition to version control and code sharing, GitHub offers a number of other features that make it a valuable tool for developers, including:

* Issue tracking: This allows developers to track bugs, feature requests, and other tasks associated with a project.
* Collaboration tools: GitHub provides features like pull requests, which allow developers to propose changes to code and have them reviewed by others before they are merged into the main codebase.
* Social coding: GitHub is also a social network for developers. You can follow other developers and organizations, see what projects they're working on, and contribute to their projects.

So, GitHub is a one-stop shop for software development, offering everything from version control and code sharing to collaboration tools and social networking.

### Steps to Clone a GitHub Project in Eclipse

1. **Open Eclipse**: Launch Eclipse IDE.
2. **Open the Git Perspective**:
   * Go to Window > Perspective > Open Perspective > Other...
   * Select Git and click Open.
3. **Clone the Repository**:
   * In the Git perspective, click on the Clone a Git repository link or use the toolbar: File > Import... > Git > Projects from Git > Clone URI.
   * In the dialog that opens, enter the repository URL from GitHub. For example, if your repository URL is https://github.com/username/repository.git, paste it into the URI field.
   * Eclipse will automatically fill in the Host and Repository Path fields.
   * Click Next.
4. **Authenticate (if necessary)**:
   * If the repository is private, you will need to enter your GitHub username and password (or a personal access token if two-factor authentication is enabled).
5. **Select Branches**:
   * Select the branches you want to clone. Typically, you would clone the master or main branch, but you can choose others if needed.
   * Click Next.
6. **Choose Local Destination**:
   * Choose a local directory where the repository will be cloned.
   * Click Finish.
7. **Import the Project into Eclipse**:
   * After cloning, Eclipse will show the repository in the Git Repositories view.
   * Right-click on the cloned repository and select Import Projects.
   * Choose Import existing Eclipse projects and click Next.
   * Eclipse will search for existing projects in the repository. Select the project(s) you want to import and click Finish.

GIT Commands:

Sure, here is a list of common Git commands along with brief descriptions of what they do:

### Configuration

* git config --global user.name "Your Name": Set the name you want attached to your commit transactions.
* git config --global user.email "your\_email@example.com": Set the email you want attached to your commit transactions.

### Starting a Project

* git init: Initialize a new Git repository.
* git clone <repository>: Clone an existing repository.

### Basic Snapshotting

* git add <file>: Add file contents to the index (staging area).
* git add .: Add all changes in the current directory to the index.
* git commit -m "Commit message": Record changes to the repository with a message.
* git status: Show the working tree status.
* git diff: Show changes between working directory and index.

### Branching and Merging

* git branch: List branches. The current branch will be highlighted with an asterisk.
* git branch <branch-name>: Create a new branch.
* git checkout <branch-name>: Switch to another branch.
* git merge <branch-name>: Merge specified branch’s history into the current branch.
* git branch -d <branch-name>: Delete a branch.

### Inspecting and Comparing

* git log: Show the commit history.
* git log --oneline: Show the commit history in a compact format.
* git show <commit>: Show various types of objects.
* git diff <commit>: Show changes between the working directory and a particular commit.

### Sharing and Updating Projects

* git remote add <name> <url>: Add a remote repository.
* git fetch <remote>: Fetch changes from the remote, but do not merge them.
* git pull <remote>: Fetch changes from the remote and merge them into the current branch.
* git push <remote> <branch>: Push local changes to the remote repository.

### Rewriting History

* git reset <file>: Remove file changes from the index (unstage).
* git reset --hard <commit>: Reset the index and working directory to the specified commit.
* git revert <commit>: Create a new commit that undoes changes made in the specified commit.

### Stashing

* git stash: Stash the changes in a dirty working directory away.
* git stash apply: Re-apply the changes stashed.
* git stash drop: Discard the stashed changes.

### Ignoring Files

* .gitignore: A file specifying intentionally untracked files to ignore.

### Advanced Commands

* git rebase <branch>: Reapply commits on top of another base tip.
* git cherry-pick <commit>: Apply the changes introduced by some existing commits.
* git tag <tag-name>: Create a tag pointing to the current commit.

### Viewing Help

* git help <command>: Get help for a specific Git command.

These commands cover most of the typical tasks you would need to perform when working with Git.

PROJECT DEVELOPMENT

Any application have 3 layers in the project. There are

1. physical Layer
2. Business Layer
3. Data Layer

**1.Physical Layer:**

Physical layer is nothing but which we can see usually called as front-end. We use JSP, HTML, CSS etc.

**2.Business Layer:**

We can use core languages like Java, Python etc.. Which will have the core implementation, all transactions will be written here only.

**3.Data Layer:**

All the data base transactions will be stored either creating either deleting or updating in data layer.

**CRUD OPERATIONS:**

* C stands for Create or Insert
* R stands for Read or Get
* U stands for Update or Edit
* D stands for Delete

**Implementation we did**

* First we can do the troubleshooting and debug
* Maintenance and Monitory
* We can track errors easily
* We can get error history
* We can provide alerts and notifications

PROJECT MIGRATION

Project Migration steps:

1. We must identify which project we need migrate.
2. Navigate to git directory and download the project.
3. Extracts keep into one folder and import that project into IDE.
4. We referred the pom.xml from here and we have created the new spring boot project(start.spring.io).
5. Extract keep into one folder and import that project into IDE.
6. We have identified the packages from the old projects and created the same into new spring boot

for Loop and foreach Loop in Java

**1.** for Loop

* **Definition**: The for loop is used to execute a block of code a specific number of times. It is useful when you know be for hand how many times you need to repeat a task.

for (initialization; condition; update) {

// code to be executed

}

for Loop: Used when you know the exact number of iterations.

for (int i = 0; i < 5; i++) {

System.out.println(i);

}

public class Main {

public static void main(String[] args) {

for (int i = 0; i < 5; i++) {

System.out.println("Value of i: " + i);

}

}

}

* + **Explanation**:
    - int i = 0: Initializes the loop variable i to 0.
    - i < 5: The loop runs as long as i is less than 5.
    - i++: Increments i by 1 after each iteration.
    - The loop prints the value of i from 0 to 4.

**2.** foreach Loop (Enhanced for Loop)

* **Definition**: The foreach loop, also known as the enhanced for loop, is used to iterate over elements in a collection or array. It is simpler and cleaner when you just need to traverse the entire collection or array.

**Syntax**:

for (type element : collection) {

// code to be executed

}

public class Main {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 4, 5};

for (int number : numbers) {

System.out.println("Number: " + number);

}

}

}

* + **Explanation**:
    - int[] numbers = {1, 2, 3, 4, 5}: An array of integers.
    - for (int number : numbers): The loop variable number takes each value from the array numbers one by one.
    - The loop prints each number in the array
* foreach Loop: Used to iterate over elements in a collection or array.

int[] arr = {1, 2, 3, 4, 5};

for (int num : arr) {

System.out.println(num);

}

Both loops are essential for iterating through data, with for loops offering more control over the iteration process and foreach loops providing a more concise way to iterate over collections and arrays.

do-while

1. The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

do {

// Code block to be executed

} while (condition);

1. The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Note:- Do not forget to increase the variable used in the condition, otherwise the loop will never end!