TO CREATE A MAVEN PROJECT

🔷 In Eclipse

1. Go to **File → New → Project**
2. Choose **Maven → Maven Project**
3. Check **Create a simple project (skip archetype selection)**
4. Click **Next**
5. Fill in:
    - Group Id: `com.example`
    - Artifact Id: `spring-core-demo`
    - Version: default
    - Packaging: `jar`
6. Click **Finish**
7. Eclipse creates the Maven structure and `pom.xml`

✅ Exercise 1: Configuring a Basic Spring Application

🔧 Goal:

Create a simple Spring application that uses Java-based configuration to:

- Define a Bean
- Configure it using Spring
- Load the Application Context
- Use the bean

📦 Step-by-Step Implementation

🟢 Step 1: Create a Maven Project

🔷 Folder structure:

css

Copy Edit

```css
spring-core-demo/
├── src/
│   └── main/
│       ├── java/
│       │   └── com/example/demo/
│       │       ├── AppConfig.java
│       │       ├── HelloService.java
│       │       └── MainApp.java
│       └── resources/
└── pom.xml
```

🟢 Step 2: Add Spring Dependencies in `pom.xml`

xml

Copy Edit

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>spring-core-demo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!-- Spring Context Dependency -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.20</version>
        </dependency>
    </dependencies>
```

```
</project>
```

💡 Run `mvn clean install` to download dependencies.

🟢 Step 3: Create the Service Class (`HelloService.java`)

java

Copy Edit

```java
package com.example.demo;

public class HelloService {
    public void sayHello() {
        System.out.println("✅ Hello from Spring Bean!");
    }
}
```

🟢 Step 4: Create Spring Configuration Class (`AppConfig.java`)

java

Copy Edit

```java
package com.example.demo;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {

    @Bean
    public HelloService helloService() {
        return new HelloService();
    }
}
```

💡 `@Configuration` tells Spring that this class contains bean definitions.
💡 `@Bean` marks a method that returns a Spring-managed bean.

🟢 Step 5: Create the Main Class (`MainApp.java`)

java

Copy Edit

```java
package com.example.demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        // Load Spring Context using AppConfig class
        ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

        // Get HelloService bean from context
        HelloService service = context.getBean(HelloService.class);

        // Call the method
        service.sayHello();
    }
}
```

✅ Output

csharp

Copy Edit

```
✅ Hello from Spring Bean!
```

📚 Concepts Covered:

Concept	Description
`@Configuration`	Marks the class as a source of Spring bean definitions
`@Bean`	Used to create and return a Spring bean
`ApplicationContext`	The Spring IoC container
`AnnotationConfigApplicationContext`	Used for Java-based configuration loading

🧪 Optional: Add Another Bean with Dependency

Add a `GreetingService` that depends on `HelloService`.

🔷 `GreetingService.java`

java

Copy Edit

```
package com.example.demo;

public class GreetingService {
    private final HelloService helloService;

    public GreetingService(HelloService helloService) {
        this.helloService = helloService;
    }

    public void greet() {
        System.out.println("Greeting the user...");
        helloService.sayHello();
    }
}
```

🔷 Update `AppConfig.java`

java
Copy Edit
```
@Bean
public GreetingService greetingService() {
    return new GreetingService(helloService());
}
```

🔷 Update `MainApp.java`

java
Copy Edit
```
GreetingService gs = context.getBean(GreetingService.class);
gs.greet();
```

✅ Exercise 2: Implementing Dependency Injection in Spring (Core Java-based config)

🎯 Objective:

Implement **Constructor-based Dependency Injection** using **Spring Core (without Spring Boot)** and **Java Config (@Configuration and @Bean).**

📦 Folder Structure:

css
Copy Edit
```
spring-core-di/
├── src/
│   └── main/
│       ├── java/
│       │   └── com/example/
│       │       ├── AppConfig.java
│       │       ├── MessageService.java
│       │       ├── EmailService.java
│       │       ├── NotificationManager.java
│       │       └── MainApp.java
└── pom.xml
```

✅ Step-by-Step Code:

1️⃣ Create Interface: `MessageService.java`

java
Copy Edit
```
package com.example;

public interface MessageService {
    String getMessage();
}
```

2️⃣ Create Implementation Class: `EmailService.java`

java
Copy Edit
```
package com.example;

public class EmailService implements MessageService {
    @Override
    public String getMessage() {
        return "📩 Email sent successfully!";
    }
}
```

3️⃣ Create Client Class: NotificationManager.java

java

Copy Edit

```java
package com.example;

public class NotificationManager {
    private final MessageService messageService;

    // Constructor-based Dependency Injection
    public NotificationManager(MessageService messageService) {
        this.messageService = messageService;
    }

    public void notifyUser() {
        System.out.println("🔔 Notification: " + messageService.getMessage());
    }
}
```

4️⃣ Create Configuration Class: AppConfig.java

java

Copy Edit

```java
package com.example;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {

    @Bean
    public MessageService messageService() {
        return new EmailService();  // You can later switch to SMSService easily
    }

    @Bean
    public NotificationManager notificationManager() {
        return new NotificationManager(messageService());
    }
}
```

5️⃣ Main Class: MainApp.java

java

Copy Edit

```java
package com.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        // Load Spring context using Java Config
        ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

        // Get NotificationManager bean
        NotificationManager manager = context.getBean(NotificationManager.class);

        // Call notify method
        manager.notifyUser();
    }
}
```

🧪 Output:🔔 Notification: 📩 Email sent successfully!

Pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>spring-core-demo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!-- ✅ Spring Core Context Dependency -->
        <dependency>
```

```
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.20</version>
        </dependency>
    </dependencies>
</project>
```

✅ Exercise 4: Creating and Configuring a Maven Project

> 💡 This exercise helps you understand how to manually create a Spring Maven project using proper folder structure, Maven build configuration, and Spring dependency management.

🎯 Objective:

- Set up a **Spring Core** project using **Maven** from scratch
- Understand the **standard folder structure**
- Use **pom.xml** to add Spring dependencies
- Write a minimal Spring program

📝 Step-by-Step Guide

🔷 Step 1: Create the Maven Project

Option A: Using CLI

bash
Copy Edit
```bash
mvn archetype:generate -DgroupId=com.example -DartifactId=spring-core-project -DarchetypeArtifactId=maven-archetype-quickstar
```

Option B: Manually (if using IntelliJ or Eclipse)

1. Create a new **Maven Project**
2. Provide:
    - **Group Id:** com.example
    - **Artifact Id:** spring-core-project
3. Finish setup

📁 Step 2: Understand Folder Structure

Your project will look like:

css
Copy Edit
```css
spring-core-project/
├── src/
│   ├── main/
│   │   └── java/
│   │       └── com/example/
│   │           ├── AppConfig.java
│   │           ├── HelloService.java
│   │           └── MainApp.java
├── pom.xml
```

🛠️ Step 3: Add Spring Dependencies in pom.xml

xml
Copy Edit
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="
           http://maven.apache.org/POM/4.0.0
           http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>spring-core-project</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!-- ✅ Spring Context (Core) -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.20</version>
        </dependency>
    </dependencies>
</project>
```

Then do:

bash

Copy Edit

```bash
mvn clean install
```

📦 Step 4: Create Bean Class – HelloService.java

java

Copy Edit

```java
package com.example;

public class HelloService {
    public void greet() {
        System.out.println("🌱 Hello from Spring Core using Maven!");
    }
}
```

🧱 Step 5: Create Spring Configuration – AppConfig.java

java

Copy Edit

```java
package com.example;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {
    @Bean
    public HelloService helloService() {
        return new HelloService();
    }
}
```

🚀 Step 6: Main Application – MainApp.java

java

Copy Edit

```java
package com.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
        HelloService helloService = context.getBean(HelloService.class);
        helloService.greet();
    }
}
```

✅ Final Output:

csharp

Copy Edit

```
🌱 Hello from Spring Core using Maven!
```

✅ Spring Data JPA – Quick Example

🎯 Objective:

Build a minimal Spring Boot app using **Spring Data JPA** that:

- Uses a simple Country entity
- Performs CRUD operations using JpaRepository
- Requires **no custom SQL**

⚙️ Tech Stack:

- Spring Boot
- Spring Data JPA
- H2 Database (in-memory, for simplicity)

📁 Project Structure:

css

Copy Edit

```
spring-data-jpa-quick/
├── src/
│   └── main/
│       ├── java/
│       │   └── com/example/demo/
│       │       ├── Country.java
│       │       ├── CountryRepository.java
│       │       ├── DataLoader.java
│       │       └── Application.java
│       └── resources/
│           └── application.properties
└── pom.xml
```

🧱 Step-by-Step Implementation

🔷 1. pom.xml

xml

Copy Edit

```xml
<project ...>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>spring-data-jpa-quick</artifactId>
    <version>1.0-SNAPSHOT</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.1</version>
    </parent>

    <dependencies>
        <!-- Spring Data JPA -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- H2 Database -->
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
    </dependencies>
</project>
```

🔷 2. Country.java

java

Copy Edit

```java
package com.example.demo;

import jakarta.persistence.*;

@Entity
public class Country {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String code;

    // Getters and setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getCode() { return code; }
    public void setCode(String code) { this.code = code; }
}
```

🔷 3. CountryRepository.java

java

Copy Edit

```java
package com.example.demo;

import org.springframework.data.jpa.repository.JpaRepository;

public interface CountryRepository extends JpaRepository<Country, Long> {
    Country findByCode(String code);
}
```

> 💡 This gives you all CRUD methods like save(), findById(), delete(), etc., **for free.**

🔷 4. DataLoader.java

java

Copy Edit

```java
package com.example.demo;

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class DataLoader implements CommandLineRunner {

    private final CountryRepository repo;

    public DataLoader(CountryRepository repo) {
        this.repo = repo;
    }

    @Override
    public void run(String... args) throws Exception {
        // Create new country
        Country c1 = new Country();
        c1.setName("India");
        c1.setCode("IN");

        repo.save(c1);

        // Fetch and print
        Country found = repo.findByCode("IN");
        System.out.println("Found Country: " + found.getName() + " (" + found.getCode() + ")");
    }
}
```

🔷 5. Application.java

java

Copy Edit

```java
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

🔷 6. application.properties

properties

Copy Edit

```properties
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.h2.console.enabled=true
```

🏁 Output in Console:

pgsql

Copy Edit

```
Hibernate: insert into country (code, name, id) values (?, ?, ?)
Hibernate: select country0_.id as id1_0_, country0_.code as code2_0_, country0_.name as name3_0_ from country country0_ where
Found Country: India (IN)
```