IOC INVERSION OF CONTROL

to inject the one class object in another one

**layers** -

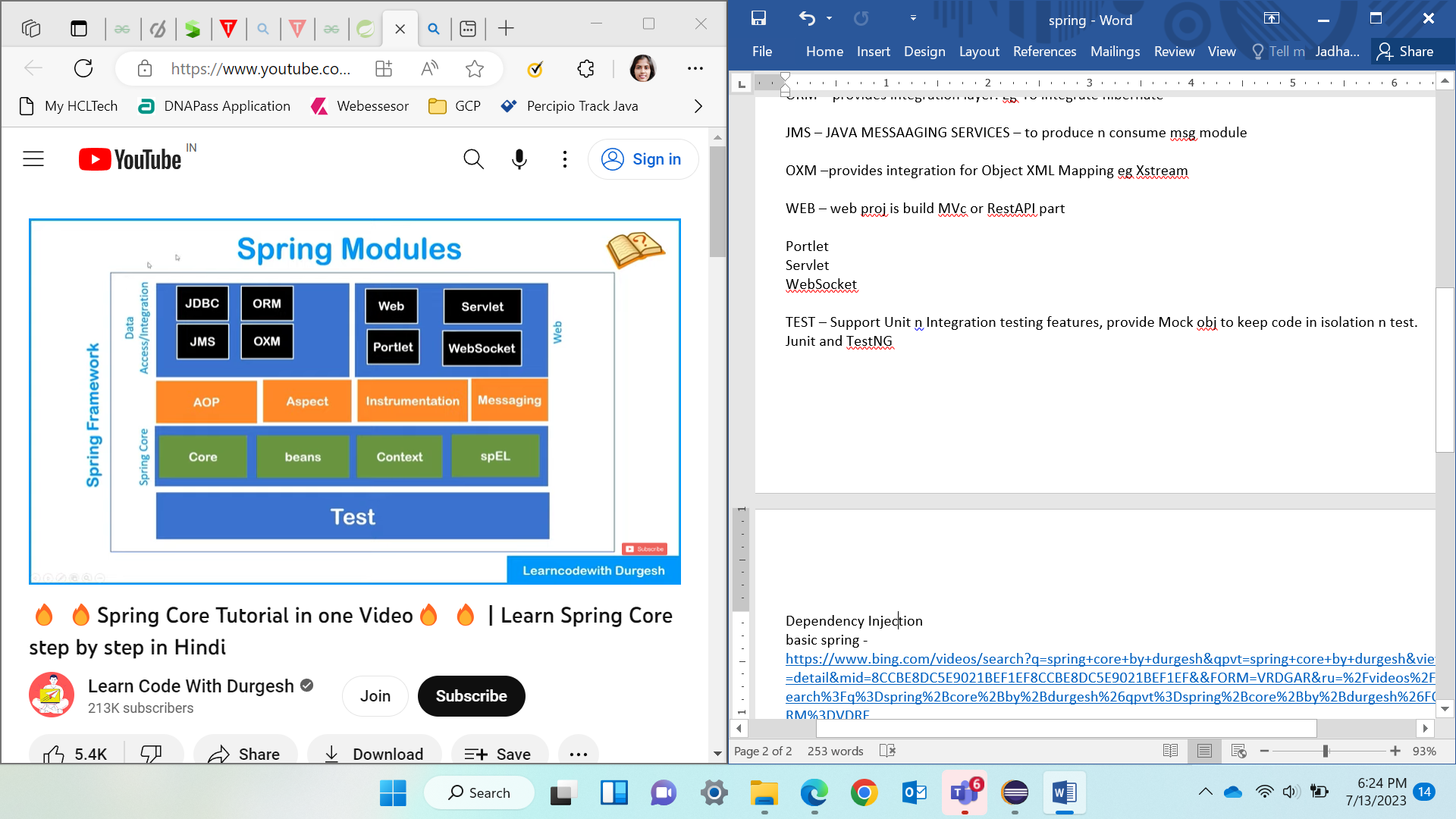
UI/Presentation layer

Business/Service layer

Data Access Layer

Database

**SPRING FRAMEWORK**



CORE – fundamental features provides ioc n dependencies functions

BEANS - fundamental provides ioc n dependencies functions

CONTEXT – inherit features from bean module like factory design and add internationalization, event propagation, resource loading, transparent creation of context. Provide J2EE features.

SPEL SPRING EXPRESSION LANGUAGE – query n manipulates Object Graph at runtime.

AOP ASPECT ORIENTED PROGRAMMING – define method interceptors and point cuts to decouple the code to make functionality independent.

ASPECT –

INSTRUMENTATION – class loader implementation is supported

MESSAGING – serve as foundation to msg appl. Has annotations that maps methods to msg

JDBC – provide extra abstraction layer to contact db

ORM – provides integration layer. Eg-To integrate hibernate

JMS – JAVA MESSAAGING SERVICES – to produce n consume msg module

OXM –provides integration for Object XML Mapping eg Xstream

WEB – web proj is build MVc or RestAPI part

Portlet Servlet WebSocket

TEST – Support Unit n Integration testing features, provide Mock obj to keep code in isolation n test. Junit and TestNG

**Spring BOOT**– module of spring which speed up development. Easy create stand-alone, production-grade Spring bases appl. u just run. Provides easier n faster way to set-up, configure, run both simple n web based appl.

Spring boot = Spring framework + embedded server (tomcat server) – configuration (no need to do it manually)

Convention over configuration s/w design style. Opinionated Default – automatically configure.

If JPA dependency added - scan class path n find dependency and auto configure.

**SPRING IOC**

IOC is predefined program. managing the objects of an application. It uses dependency injection to achieve inversion of control.Creation of object, hold in memory, inject 1obj into another as required.

Info provided to IOC – Beans file(Java pojo class), Config files(XML) which bean is dependent which bean.

Application – get the obj as uses it

ApplicationContext – interface that extend bean factory and additional properties.

The interfaces [BeanFactory](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/BeanFactory.html)and [ApplicationContext](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/ApplicationContext.html)**represent the Spring IoC container**. Here, *BeanFactory* is the root interface for accessing the Spring container having basic functionalities.On the other hand, the *ApplicationContext* is a sub-interface of the *BeanFactory*. Therefore, it offers all the functionalities of *BeanFactory.*

Furthermore, it **provides** **more enterprise-specific functionalities**. The important features of *ApplicationContext* are **resolving messages, supporting internationalization, publishing events, and** application-layer specific contexts. This is why we use it as the default Spring container.

[bean](https://www.baeldung.com/spring-bean) is **an object that the Spring container instantiates, assembles, and manages** define beans for service layer objects, data access objects (DAOs), presentation objects, infrastructure objects such as Hibernate *SessionFactories,* JMS Queues, and so forth.

shouldn't configure fine-grained domain objects in the container. It's usually the responsibility of DAOs and business logic to create and load domain objects.

**public** **class** **AccountService** { @Autowired **private** AccountRepository accountRepository; // getters and setters }

*@Bean***-annotated methods within a***@Configuration***class**. The *@Bean* annotation on a method indicates that the method creates a Spring bean. Moreover, a class annotated with *@Configuration* indicates that it contains Spring bean configurations.

@Configuration

public class AccountConfig {

@Bean

public AccountService accountService() {

return new AccountService(accountRepository());

}

@Bean

public AccountRepository accountRepository() {

return new AccountRepository();

}

}

enable annotation-based configuration via XML configuratiouse a set of annotations on our Java classes, methods, constructors, or fields to configure beans. examples of these annotations are *@Component*, *@Controller*, *@Service*, *@Repository*, *@Autowired*, and *@Qualifier*.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<context:annotation-config/>

<context:component-scan base-package="com.baeldung.applicationcontext"/>

</beans>

the annotation-config tag enables annotation-based mappings. The component-scan tag also tells Spring where to look for annotated classes.

do all bean mappings in an XML configuration file.

types of ApplicationContext containers

[AnnotationConfigApplicationContext](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/AnnotationConfigApplicationContext.html)classtake classes annotated with @Configuration, @Component, and JSR-330 metadata as input.

ApplicationContext context = new AnnotationConfigApplicationContext(AccountConfig.class);

AccountService accountService = context.getBean(AccountService.class);

[**AnnotationConfigWebApplicationContext**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/context/support/AnnotationConfigWebApplicationContext.html)**is a web-based variant** of *AnnotationConfigApplicationContext*. configure Spring's *ContextLoaderListener* servlet listener or a Spring MVC *DispatcherServlet* in a *web.xml* file.

**XML based configuration in a web application**, we can use the [XmlWebApplicationContext](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/context/support/XmlWebApplicationContext.html) class. configure it in web.xml, or implement the WebApplicationInitializer interface:

Dependency Injection

basic spring - <https://www.bing.com/videos/search?q=spring+core+by+durgesh&qpvt=spring+core+by+durgesh&view=detail&mid=8CCBE8DC5E9021BEF1EF8CCBE8DC5E9021BEF1EF&&FORM=VRDGAR&ru=%2Fvideos%2Fsearch%3Fq%3Dspring%2Bcore%2Bby%2Bdurgesh%26qpvt%3Dspring%2Bcore%2Bby%2Bdurgesh%26FORM%3DVDRE>

[Spring Boot Tutorial in Hindi - YouTube](https://www.youtube.com/playlist?list=PL0zysOflRCelmjxj-g4jLr3WKraSU_e8q)

playlist -- <https://www.youtube.com/playlist?list=PL0zysOflRCekYnhLjQGwpdJYzr38QXdhl>

1. SPRING BOOT REST API

<https://youtu.be/sdDDuQuX2cg>

@RestController - presentation layer.. takes the request

@GetMapping - to map the method to GET function so that on trigger it will show desired page

@Autowired - Injects the object of the child(implemented class) of interface class into the variable being declared

2. SPRING BOOT DB

1. [Spring boot 🔥🔥💯💯| Creating REST API using spring boot in one video in Hindi - YouTube](https://www.youtube.com/watch?v=sdDDuQuX2cg&list=PL0zysOflRCekYnhLjQGwpdJYzr38QXdhl&index=2)

<https://youtu.be/V2p5rVIBT5M>

<https://www.youtube.com/watch?v=uQWYAA7hpVU>

refer this - <https://www.youtube.com/watch?v=3-5e5cXfwrU>

using workbench

<https://www.youtube.com/watch?v=p2vOeLSDvyg>

website for CRUD - <https://codebots.com/crud>

**Appl -> controller -> entities -> services -> serviceimpl**

**Controller madhe kay karycha req alyavr the lihycha -- @RestController – display karycha -- @Autowired lavla karan interface cha object ahe the samjyala hava ani the inject karta mg impl. ---**

**@GetMapping – get request sathi.. req send karta 3 -- @PostMapping – post req sathi.. response deta**

**@@GetMapping("/courses/{courseId}") – dynamic value hein ghenyasathi method madhe lihycha -**@PathVariable

**Entities madhe kay data ahe dakhvycha the lihycha**

**Service interface karto loose coupling sathi**

**Serviceimpl karto karan interface madhe nhi impl. Lihu shakt ---- impl. Weli class la @Service annotate karycha jyane Spring frmk samjel ki hein service class ahe service provide karnara**

**Application**

package com.springdb.sdemo.springcrud;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringcrudApplication {

public static void main(String[] args) {

SpringApplication.run(SpringcrudApplication.class, args);

}

}

**Controller**

package com.springdb.sdemo.springcrud.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

//import org.springframework.web.bind.annotation.GetMapping;

//import org.springframework.web.bind.annotation.PathVariable;

//import org.springframework.web.bind.annotation.PostMapping;

//import org.springframework.web.bind.annotation.PutMapping;

//import org.springframework.web.bind.annotation.RequestBody;

//import org.springframework.web.bind.annotation.RestController;

import com.springdb.sdemo.springcrud.entities.Course;

import com.springdb.sdemo.springcrud.services.Servicess;

@RestController

public class firstDisplay {

@Autowired

private Servicess service\_obj;

@GetMapping("/home")

public String hello() {

return "Hello";

}

@GetMapping("/courses")

public List<Course>getCourses()

{

return this.service\_obj.getCourses();

}

@GetMapping("/courses/{id}")

public Course getCourse(@PathVariable long id)

{

return this.service\_obj.getCourse(id);

}

@PostMapping("/courses")

public Course addCourse(@RequestBody Course course)

{

return this.service\_obj.addCourse(course);

}

@PutMapping("/courses/{id}")

public Course updateCourse(@RequestBody Course course)

{

return this.service\_obj.updateCourse(course);

}

@DeleteMapping("/courses/{courseId}")

public ResponseEntity<HttpStatus> deleteCourse(@PathVariable String courseId){

try {

this.service\_obj.deleteCourse(Long.parseLong(courseId));

return new ResponseEntity<>(HttpStatus.OK);

}catch(Exception e) {

return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

}

**DAO**

**package com.springdb.sdemo.springcrud.dao;**

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

**import com.springdb.sdemo.springcrud.entities.Course;**

**public interface CourseDao extends JpaRepository<Course,Long> {**

**}**

**ENTITIES**

**package com.springdb.sdemo.springcrud.entities;**

**import javax.persistence.Entity;**

**import javax.persistence.Id;**

**@Entity**

**public class Course {**

**@Id**

**private long id;**

**private String title;**

**private String description;**

**public Course(long id, String title, String description) {**

**super();**

**this.id = id;**

**this.title = title;**

**this.description = description;**

**}**

**public Course() {**

**super();**

**// TODO Auto-generated constructor stub**

**}**

**public long getId() {**

**return id;**

**}**

**public void setId(long id) {**

**this.id = id;**

**}**

**public String getTitle() {**

**return title;**

**}**

**public void setTitle(String title) {**

**this.title = title;**

**}**

**public String getDescription() {**

**return description;**

**}**

**public void setDescription(String description) {**

**this.description = description;**

**}**

**@Override**

**public String toString() {**

**return "Course [id=" + id + ", title=" + title + ", description=" + description + "]";**

**}**

**}**

**SERVICES**

package com.springdb.sdemo.springcrud.services;

import java.util.List;

import com.springdb.sdemo.springcrud.entities.Course;

public interface Servicess {

public List<Course> getCourses();

public Course getCourse(long courseId);

public Course addCourse(Course course);

public Course updateCourse(Course course);

public void deleteCourse(long parseLong);

}

**SERVICES IMPLEMENTATION**

package com.springdb.sdemo.springcrud.services;

//import java.util.ArrayList;

import java.util.List;

//import java.util.stream.Collectors;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.springdb.sdemo.springcrud.dao.CourseDao;

import com.springdb.sdemo.springcrud.entities.Course;

@Service

public class ServicesImpl implements Servicess {

@Autowired

private CourseDao coursedao;

@Override

public List<Course> getCourses() {

return coursedao.findAll();

}

@Override

public Course getCourse(long courseId) {

Optional<Course> c = coursedao.findById(courseId);

try {

Course c1 = c.get();

return c1;

}

catch(Exception e) {

return null;

}

}

@Override

public Course addCourse(Course course) {

coursedao.save(course);

return course;

}

@Override

public Course updateCourse(Course course) {

coursedao.save(course);

return course;

}

@Override

public void deleteCourse(long parseLong) {

Course entity = coursedao.getReferenceById(parseLong);

coursedao.delete(entity);

}

}

////package com.springdb.sdemo.springcrud.services;

////import java.util.ArrayList;

////import java.util.List;

////import java.util.stream.Collectors;

////import org.springframework.stereotype.Service;

////import com.springdb.sdemo.springcrud.entities.Course;

////@Service

////public class ServicesImpl implements Servicess {

//// List<Course> list;

//// public ServicesImpl() {

//// list = new ArrayList<>();

//// list.add(new Course(145,"Java Core","basic of java"));

//// list.add(new Course(135,"spring boot","rest api using spring boot"));

//// }

//// @Override

//// public List<Course> getCourses() {

//// return list;

//// }

////

//// @Override

//// public Course getCourse(long courseId) {

//// Course c = null;

//// for(Course course:list) {

//// if(course.getId()==courseId) {

//// c=course;

//// break;

//// }

//// }

//// return c;

//// }

////

//// @Override

//// public Course addCourse(Course course) {

//// list.add(course);

//// return course;

//// }

////

//// @Override

//// public Course updateCourse(Course course) {

//// list.forEach(e ->{

//// if(e.getId()==course.getId()) {

//// e.setTitle(course.getTitle());

//// e.setDescription(course.getDescription());

//// }

//// });

//// return course;

//// }

////

//// @Override

//// public void deleteCourse(long parseLong) {

//// list=this.list.stream().filter(e -> e.getId()!=parseLong).collect(Collectors.toList());

//// }

//// }

Application.properties

#server.port = 8123

#logging.level.com.springdb.sdemo.springcrud=DEBUG

spring.jpa.hibernate.ddl-auto = update

spring.datasource.url= jdbc:mysql://localhost:3306/demoschema

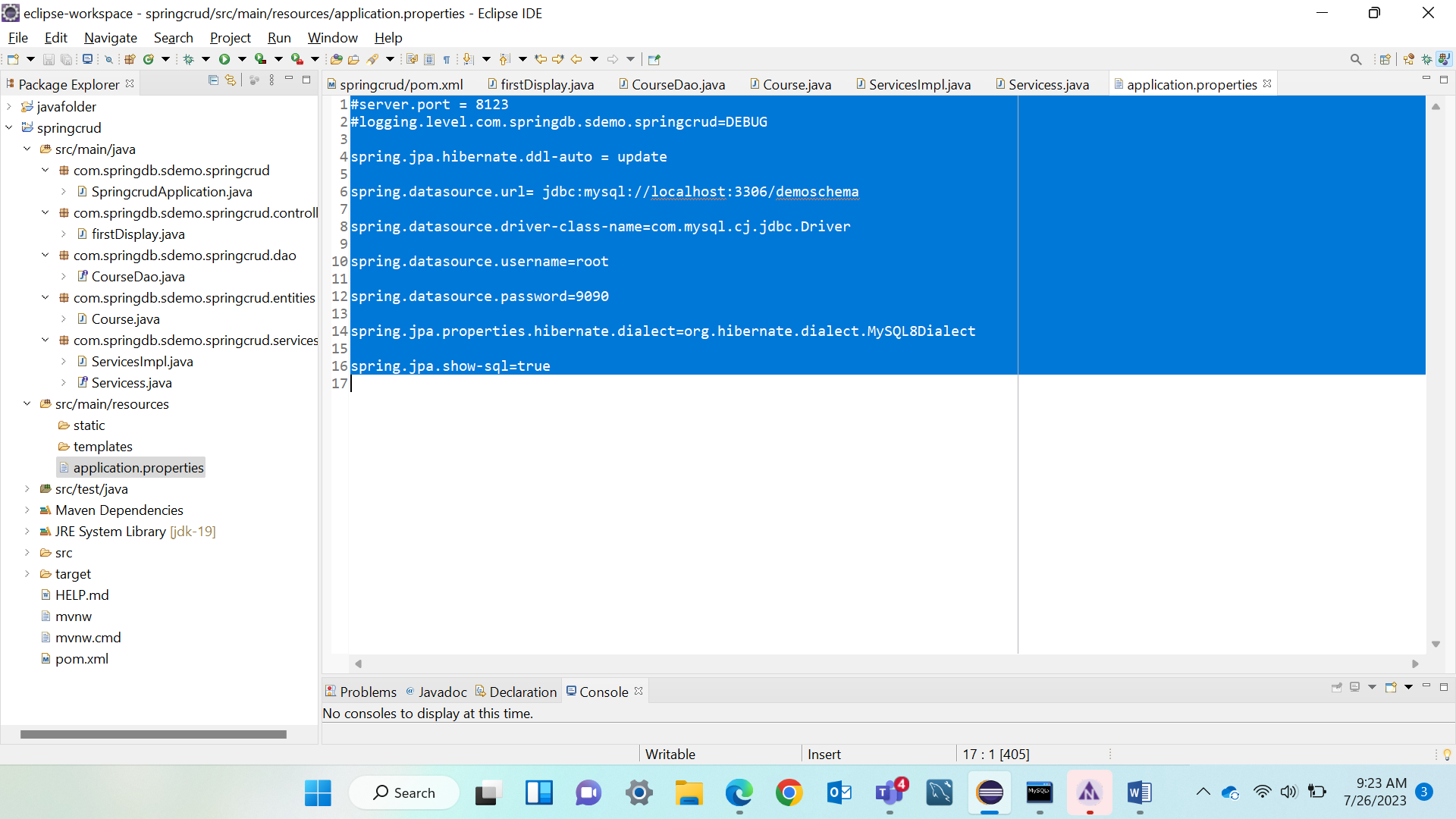
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.username=root

spring.datasource.password=9090

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.show-sql=true



**SPRING BOOT + BATCH**

[🔥 Creating Excel file from database using Spring Boot | Creating API to download excel from DB - YouTube](https://www.youtube.com/watch?v=9fwHA9s3QGs)

[Spring Batch - Database to CSV File - YouTube](https://www.youtube.com/watch?v=sVl3UTnSuFw)

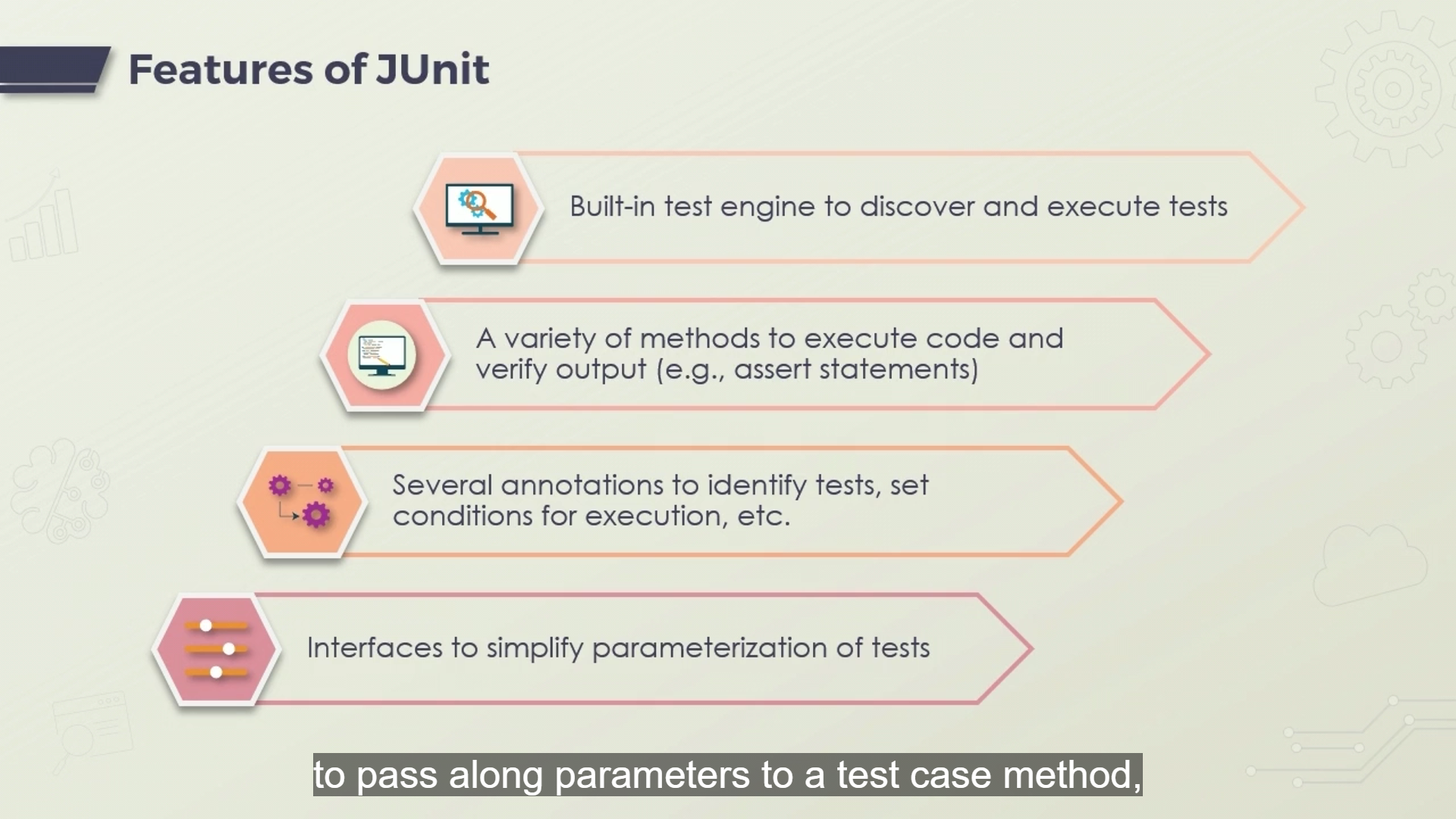
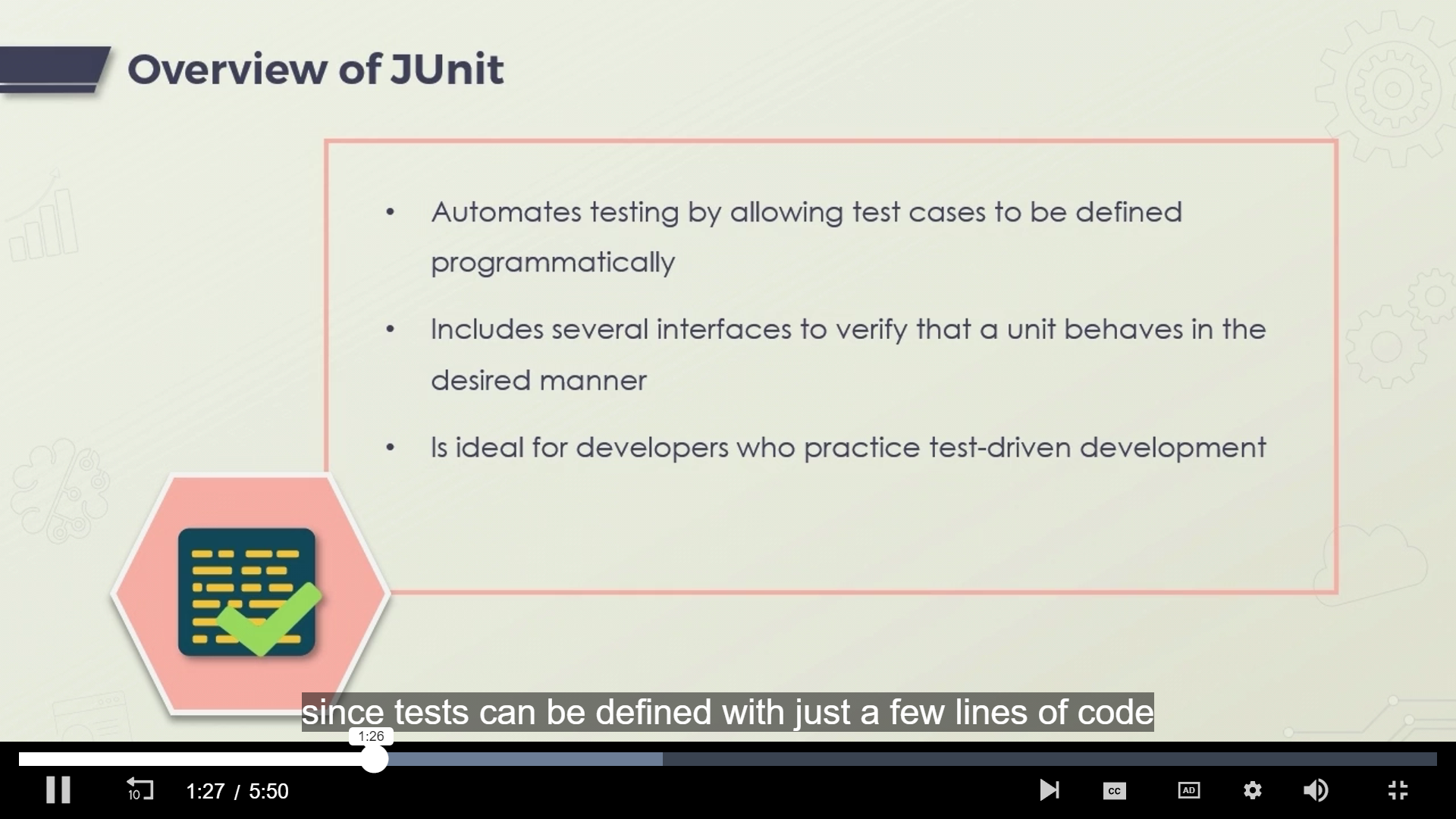
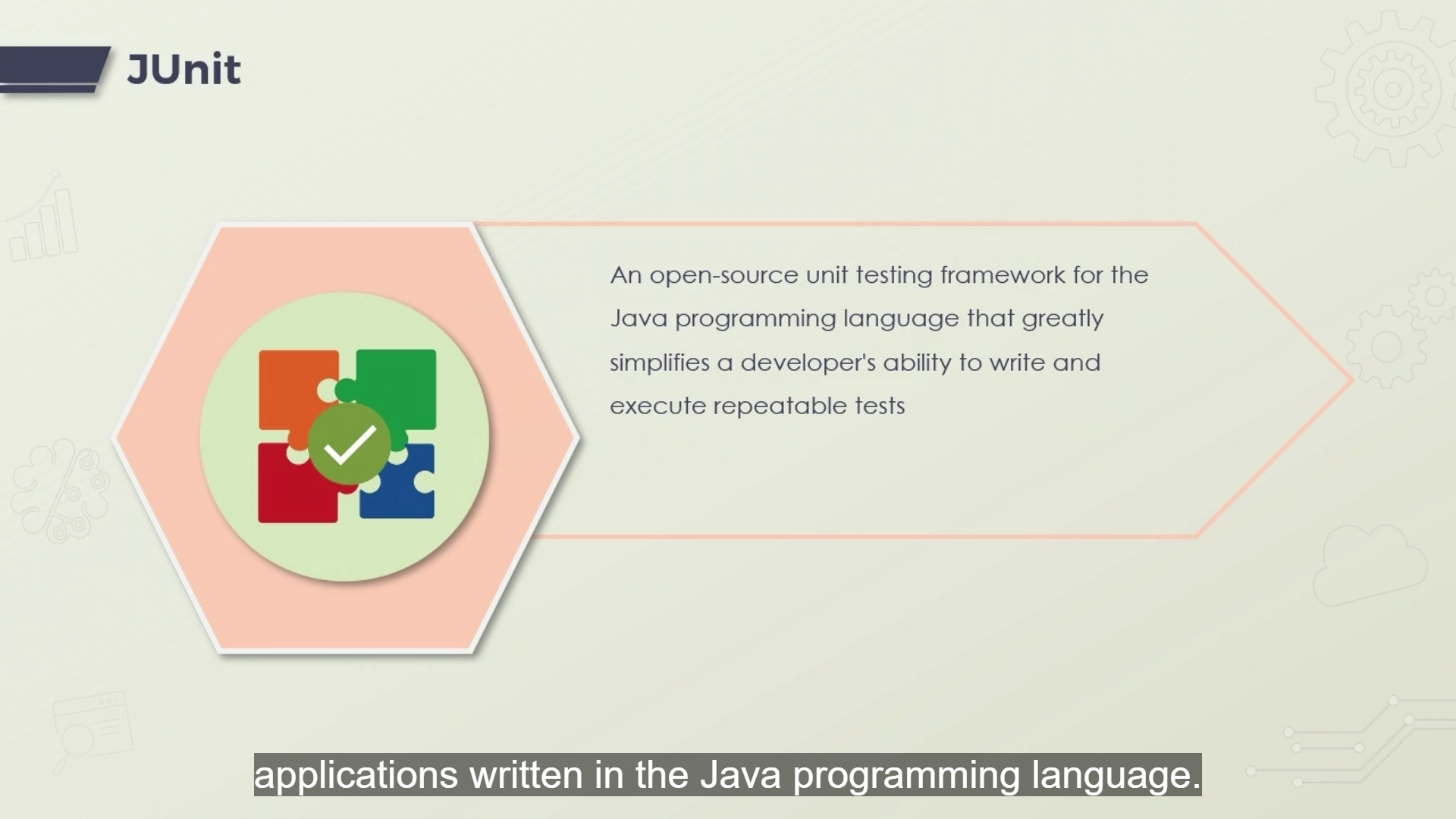
[Set Data To Database By Reading Data From CSV FILE USING SPRING BOOT & JPA - YouTube](https://www.youtube.com/watch?v=G8zS4V0gqf8)

[Creating API for Dumping Excel data to MYSQL using Spring Boot || Hindi - YouTube](https://www.youtube.com/watch?v=qflq2xxweUw)

[Spring Batch | read records from Database and process - YouTube](https://www.youtube.com/watch?v=lU44oS2Y3IQ)

[🔥Dumping data from CSV to MYSQL using Spring Boot Batch |Spring batch Tutorial step by step in Hindi - YouTube](https://www.youtube.com/watch?v=po6wM4O1Yxw)

JUNIT



**@Test -** Represents Junit test/Test Case.. method must be public and void.. Should not return any value

**@AfterAll @AfterEach**

**@BeforeAll @BeforeEach**

**@AssertEquals @AssertTrue @AssertFalse**

**Exceptions - @InvalidException**

**Display Names to Test Cases- @DisplayName(“”)**

**EnabledOnOs(OS.MAC) DisabledOnOs()**

Batch links - [What Is a Batch Job? – BMC Software | Blogs](https://www.bmc.com/blogs/batch-jobs/?)

[Basics of Batch Scripting - GeeksforGeeks](https://www.geeksforgeeks.org/basics-of-batch-scripting/?)

Batch Jobs –

batch job’ [originated](https://en.wikipedia.org/wiki/Batch_processing) in the days of mainframe computers when users entered programs onto punch cards and then pass it to system operator to feed it in comp.

Though def and usage changed over years but the background processing is alive.

Whether for routine organizational processes, like payroll, or to keep servers from being tied up during the workday, batch jobs will be essential for years to come.

batch job is a scheduled program that is assigned to run on a computer without further user interaction. It gets executed when the system is ideal. All jobs on submission waits in queue to be processed but if there are more no. of jobs already in queue then it is executed in chronological/ priority order.

Whether for routine organizational processes, like payroll, or to keep servers from being tied up during the workday, batch jobs will be essential for years to come.

In some computer systems, batch jobs run in the background while interactive programs run in the foreground, giving them priority over batch programs as these need huge amount of memory than those of foreground prgs.

USECASES – image processing, payroll, report generation, data warehouse management, billing, inventory processing.

**Batch Job Steps-**

1. Chunk Oriented – data read processed one item at time. After going through business logic of particular appl results are grouped and stored in chunk after reaching configurable size
2. Task oriented - Utilizing batch frameworks, these chunk steps are able to then bookmark their progress using checkpoints. Using the input retrieval and output writing parts of a chunk step, the current position is saved after it is processed. So if chunk interruption occurs it restarts from the last checkpoint

**SPRING BATCH**

* Spring batch is a lightweight framework used to enable the development of robust **batch applications** vital for the daily operations of enterprise systems. Spring Batch is not a scheduling framework like Quartz, Control-M but it is intended to work in conjunction with a scheduler rather than replace a scheduler.

**Framework provides functions for −**

Spring Batch provides reusable functions that are essential in processing large volumes of records, including logging and tracing, transaction management, job processing statistics, job restart, skip, and resource management and features that enable extremely high-volume and high performance batch jobs through optimization and partitioning technique

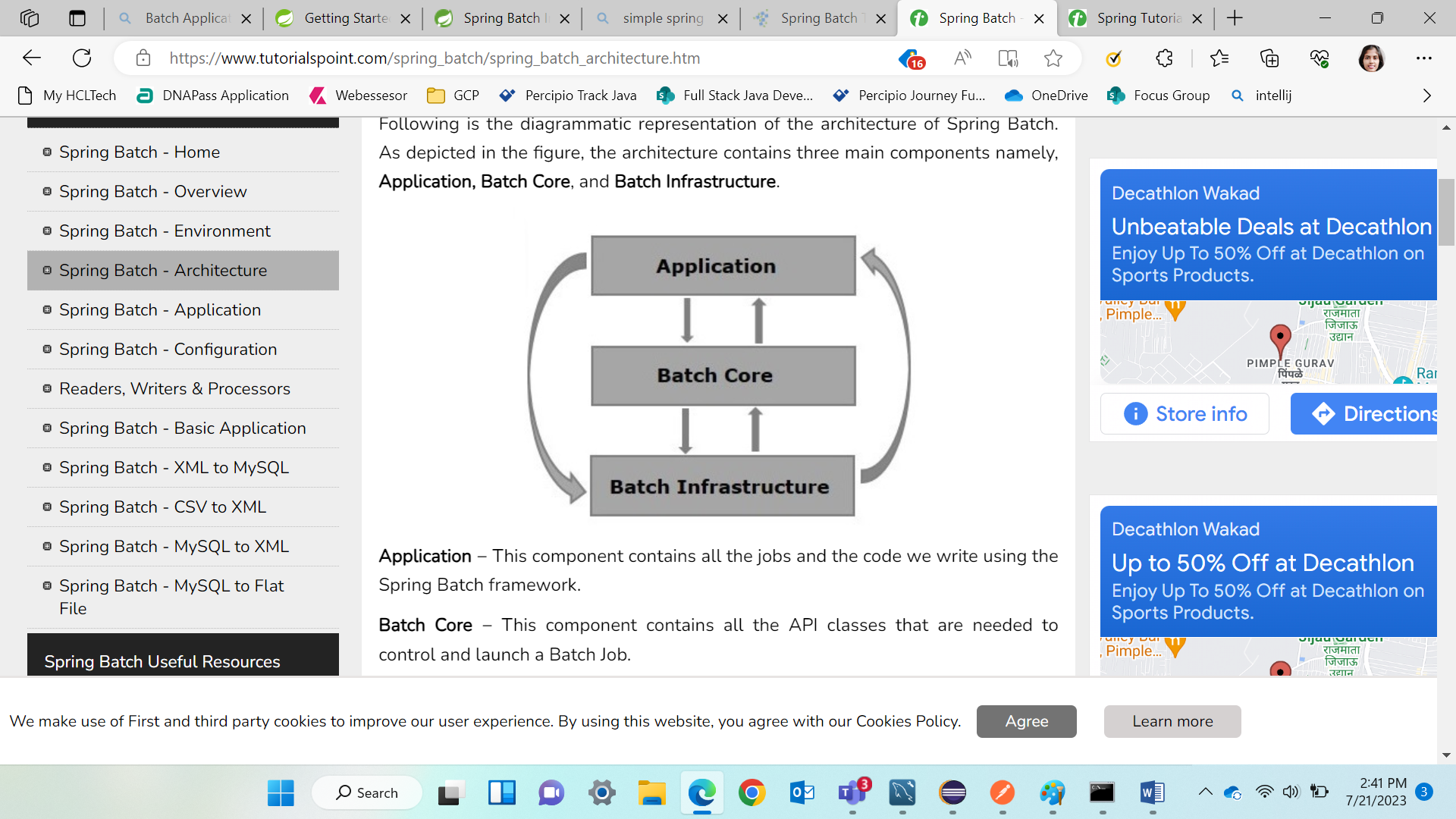
**FEATURES** –

* **FLEXIBILITY** - change an XML file to alter the order of processing in an application.
* **MAINTANABILITY** - includes steps that can be decoupled, tested, and updated, without effecting the other steps.
* **SCALABILITY** - portioning techniques\_execute the steps of a job in parallel and also single thread in parallel
* **RELIABILTY** -  failure, you can restart the job from exactly where it was stopped, by decoupling
* **MULTIPLE FILE FORMAT SUPPORT** - XML, Flat file, CSV, MYSQL, Hibernate, JDBC, Mongo, Neo4j, etc.

**Spring Batch applications support −**

* Automatic retry after failure.
* Tracking status and statistics during the batch execution and after completing the batch processing.
* To run concurrent jobs.
* Services such as logging, resource management, skip, and restarting the processing.

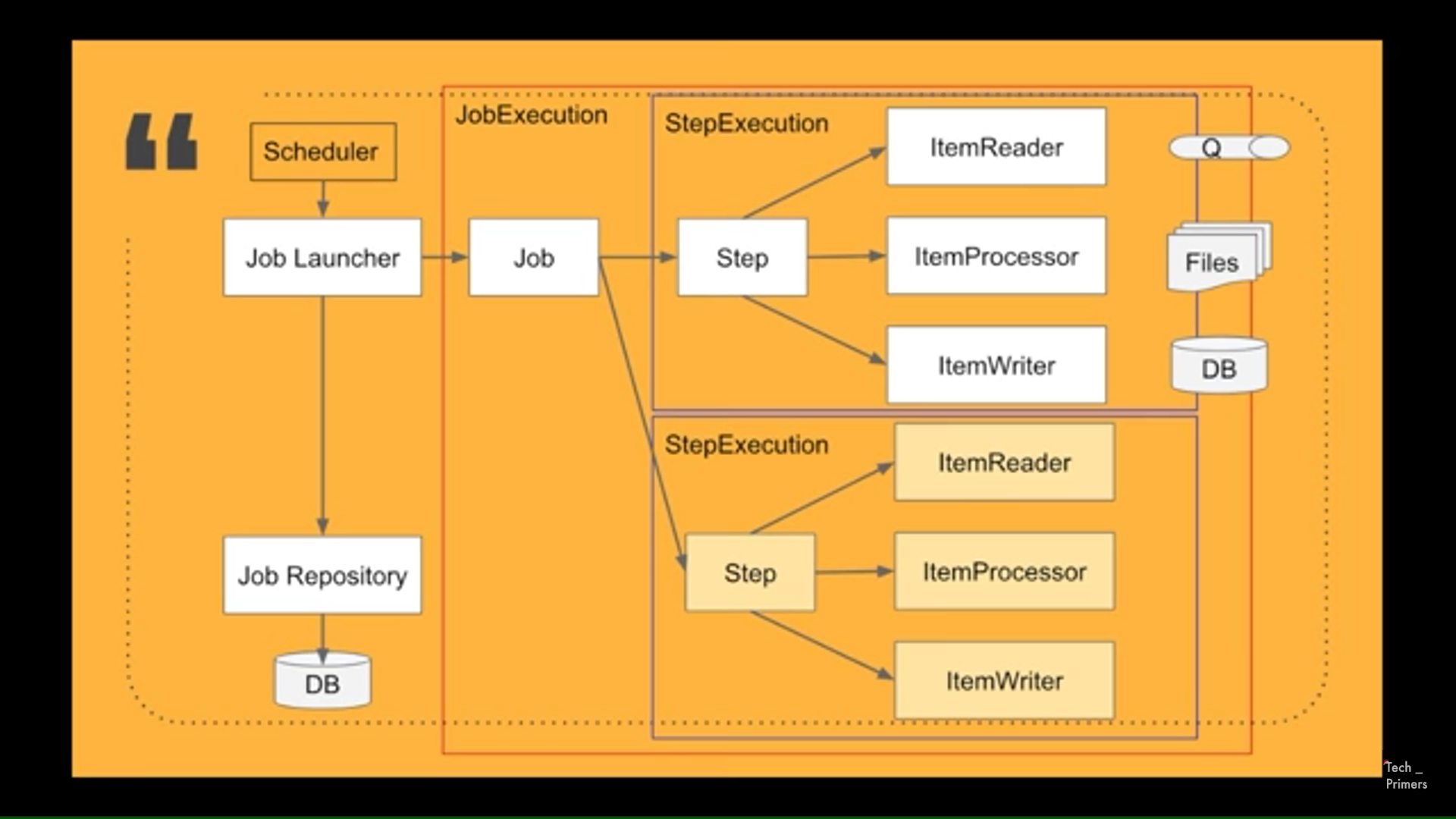
Architecture



Application − contains all the jobs and the code we write using the Spring Batch framework.

Batch Core − contains all the API classes that are needed to control and launch a Batch Job.

Batch Infrastructure − contains the readers, writers, and services used by both application and Batch core components.



Scheduler – Scheduler in JVm that looks after / trigger Spring Batch processing workflow.. triggers Job LAUNCHER

Job LAUNCHER – starting pt. for any job to start in Spring Framework

Job repository – Job LAUNCHER triggers holds stat info. How many batches run, status of it, msg processd, how many were escaped

Job – Job LAUNCHER also triggers Job registered with that particular Job Launcher

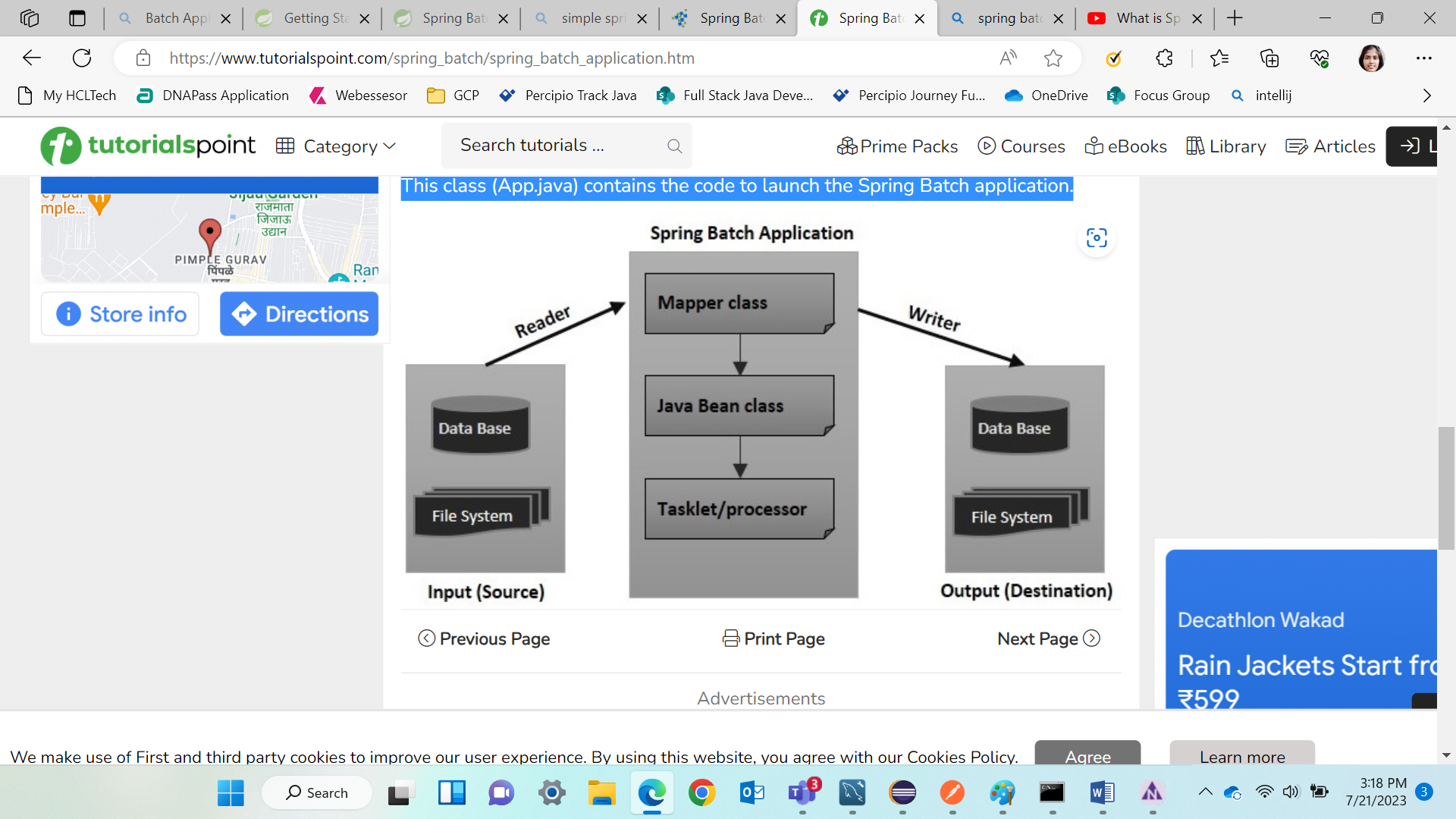
Step – particular job have step – having IteamReader.. Itemprocessor.. ItemWriter

Read the data from particular files say Q.. process the data and write / store it into the DB/particular file.

Step Execution - Steps can be multiple inside a job but has only one Instance per step of RPW.

Job Execution – Jobs execution and multiple steps are then whole called JExec.

On completion of steps/job the steps status is updated in JRepo…



Mapper Class

contains the code to get the data from the reader and to set it to a Java class with setter and getter methods (Java Bean).

Java Bean Class

A Java class with setters and getters (Java bean) represents data with multiple values. acts as a helper class. pass the data from one component (reader, writer, processer) to other in the form of object of this class.

Tasklet/processor

contains the processing code of the Spring Batch application. A processor is a class which accepts an object that contains the data read, processes it, and returns the processed data (in the form object).

Launcher class

This class (App.java) contains the code to launch the Spring Batch application.

Links for localization

[localization - Implementing a translator service in Spring Boot - Stack Overflow](https://stackoverflow.com/questions/58804414/implementing-a-translator-service-in-spring-boot)

[Guide to Internationalization in Spring Boot | Baeldung](https://www.baeldung.com/spring-boot-internationalization)

[java - Language translation in springboot web application - Stack Overflow](https://stackoverflow.com/questions/61749473/language-translation-in-springboot-web-application)

[Spring Boot internationalization i18n: Step-by-step with examples (lokalise.com)](https://lokalise.com/blog/spring-boot-internationalization/)

[Spring Boot With Spring Batch | Baeldung](https://www.baeldung.com/spring-boot-spring-batch)

[How to Internationalize and Localize Your Java and Spring Boot Apps - DZone](https://dzone.com/articles/learn-how-to-internationalize-and-localize-your-ja)

[Master Localization in Spring Boot: A Step-by-Step Tutorial - YouTube](https://www.youtube.com/watch?v=R68OJVIXKv0)

[Java Spring Boot i18n | Translating your app - YouTube](https://www.youtube.com/watch?v=ESzs1q-fpHw)

[springboot localization | springboot i18n | springboot internalization | live demo | okay java - YouTube](https://www.youtube.com/watch?v=SBQQscX8FpI)

[Localization and Internationalization in Java Tutorial Part 1 - YouTube](https://www.youtube.com/watch?v=h0educ2nosg)

[Localization and Internationalization in Java Tutorial Part 2 - YouTube](https://www.youtube.com/watch?v=Xv5UXGhfgMU)[Localization and Internationalization in Java Tutorial Part 3 - YouTube](https://www.youtube.com/watch?v=2_Bvi5mz1-k)

[28. Dynamic language support (spring.io)](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/dynamic-language.html)