Tight coupling and Dependecency Injection
In this ex. binarySearch algo is tightly coupled with bubble sort algo

```
public class BinarySearchImpl {
public void BinarySearchImpl

	private SortAlgorithm sortAlgorithm;

	public int binarySearch(int[] numbers, int numberToSearchFor) {

		int[] sortedNumbers = sortAlgorithm.sort(numbers);
		System.out.println(sortAlgorithm);
		// Search the array
		return 3;
	}

}
```

Java Annotation: Java Annotation is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Built-In Java Annotations used in Java code

- @Override(overiding parent method)
- @SuppressWarnings
- @Deprecated(his method is deprecated so compiler prints warning)

Creating Your own annotation eg:

```
//meta notation

@Retention(RetentionPolicy.RUNTIME) //at what time this annotation is
available
@Target(ElementType.TYPE)// class level

@interface SmartPhone
{
String os() default "symbian"
int version default 1

}
// using annotation
```

```
@SmartPhone(os="android" version=6)
public class NokiaPhone{
String model;
int size;
public NokiaPhone(String model ,int size)
{
this.size=size;
this.model=model;

}
public class AnnotationDemo{
public static void main(String args[])
{
NokiaPhone obj= new NokiaPhone("Fire", 5);
class c=obj.getClass();// method of obj class, return runtime class
of object
Annotation an=c.getAnnotation(SmartPhone.class);
Smartphone s=SmartPhone(an);
System.out.println(s.os());


}
}
// output : android
```

Dependency Injection:

Spring: Dependency injection framework

Spring Boot

- Inbuilt configurationIt has taken all XmlBaseConfiguration and provided annotation
  for using Spring framework.
- integrate server for development(Tomcat)
- Error handling mechanism scloud configuration
- devtools(no need to restart app)

SpringbootApplication Annotation contains SpringbootConfiguration, EnableAutoConfiguration, ComponentScan.

application.properties : configuration file- changing port, debug, configuring view resolver(to map jsp pages)

```
logging.level.org.springframework.web: DEBUG // help to debug
specific application
```

group id is very similar to package and **artifact id** is similar to class

@Controller : help us to pick up class for mapping in spring boot
@RequestMapping: help to pick method

DispatcherServlet receives all of the HTTP requests and delegates them to controller classes. and start looking from view through return statement of controller class
@ResponseBody: help to prevent dispatcher servlet to find view instead return the statement as response

Spring boot starter parent
provide default plugin, default dependencies, default java version

Spring boot starter web:
all dependencies to run web app (validation, login embedded tomcat, embedded spring webmvc etc)

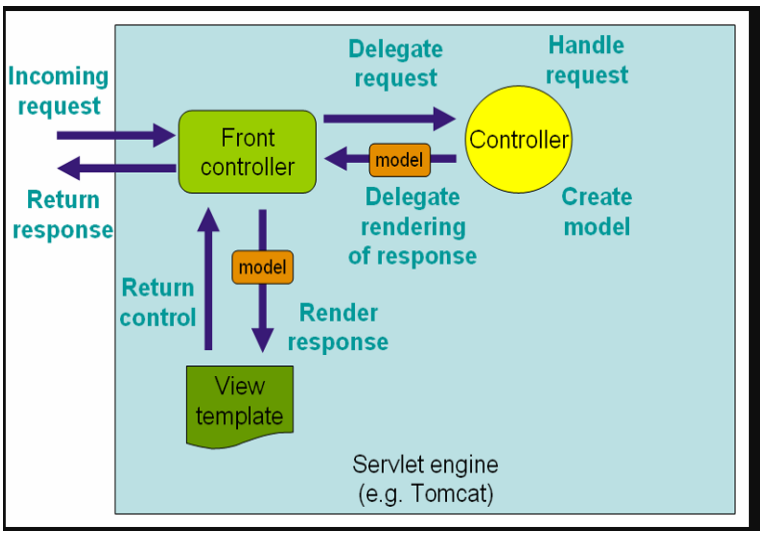
JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags

The **@RequestParam** is used to read the HTML form data provided by a user and bind it to the request parameter.

when we pass data from controller to view then it put data in **model** and the used by view

USE OF DISPATCHER SERVLET(FRONT CONTROLLER)

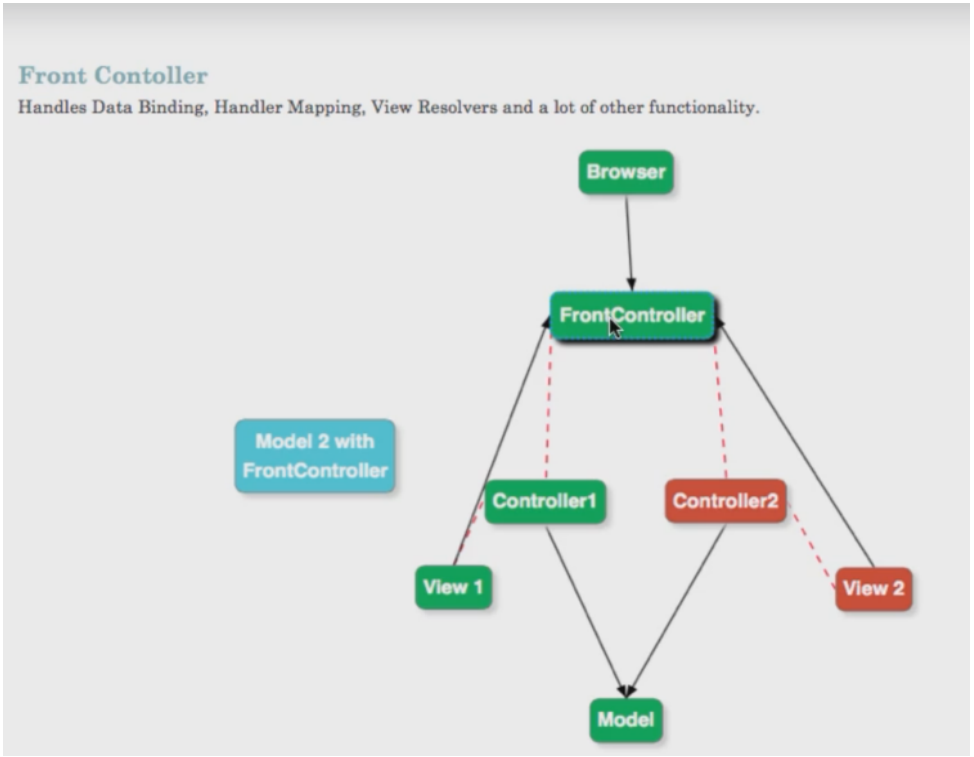


Spring boot **AutoConfiguration** auto configure dispatcher servlet

The **@Component** annotation is used to load a java class as a bean. All classes with annotation such as @Component, @bean etc are auto-wired in the spring boot application.

The **@autowired annotation** is used to inject dependency. Dependency injection is done in two ways, by name and by type. @Autowired by type uses the class type to auto wire the spring boot bean class. Help in loose coupling

Web Architecture



Session : way to store value across multiple requests(views).
@SessionAttribute Annotation help in this.

Command Object - a JavaBean which will be populated with the data from your forms Once the form is submitted, all the individual attributes are mapped/bound to this object.

Command Bean: help in double binding i.e mapping form->bean ,bean->form

@InitBinder:Annotation that identifies methods which initialize the `WebDataBinder` which will be used for populating command and form object arguments of annotated handler methods.
webdatabinder help to bind data in particular format across the controller

JSP Fragment: Top put common code fragment(like header footer)(jspf)

spring-boot-starter-security starter which aggregates Spring Security related dependencies together. after adding it Spring Boot application automatically requires the Basic Authentication for all HTTP Endpoints

Spring boot provides its own login functionality with default user name and token but we an overwrite it with our

JUNIT: UNIT TESTING FRAMEWORK

. To mark a method as a test method, annotate it with the `@Test` annotation to check an expected result versus the actual result. we use assert method or assert statement

assertEquals(expected, actual)

test method should be public and void return type

@BeforeEach or @Before runs before each test

@BeforeClass and @AfterClass : annotation , runs before and after every class, method should me static

@DisplayName can be used to define the name of the test which is displayed to the user.

assert statement which validates that expected and actual value is the same, if not the message at the end of the method is shown

@RepeatedTest defines that this test method will be executed multiple times

```
class CalculatorTest {

    Calculator calculator;

    @BeforeEach
    void setUp() {
        calculator = new Calculator();
    }

    @Test
    @DisplayName("Simple multiplication should work")
    void testMultiply() {
        assertEquals(20, calculator.multiply(4, 5),
                "Regular multiplication should work");
    }

    @RepeatedTest(5)
    @DisplayName("Ensure correct handling of zero")
    void testMultiplyWithZero() {
        assertEquals(0, calculator.multiply(0, 5), "Multiple with
zero should be zero");
        assertEquals(0, calculator.multiply(5, 0), "Multiple with
zero should be zero");
```

```
    }
}
```

assertNotNull - assertNotNull(yourObject, "optional failure message");
assertFalse - assertFalse('a' > 'b', () → "optional failure message");// check if return result is false or not
assertTrue - assertTrue('a' < 'b', () → "optional failure message");

`assertAll`.In this grouped assertion all assertions are executed, even after a failure. The error messages get also grouped together.
If you want to ensure that a test fails, if it isn't done in a certain amount of time you can use the `assertTimeout()` method.
If you want your tests to cancel after the timeout period is passed you can use the `assertTimeoutPreemptively()` method.
The `@Disabled` or `@Disabled("Why disabled")` annotation marks a test to be disabled.

@AssertArrayEqual: to check two array if they are equal or not
@**Test(timeout=100)** wheather test is run in 100 ms sec or not

Parameterized tests allow a developer to run the same test over and over again using different values. There are five steps that you need to follow to create a parameterized test.

- Annotate test class with @RunWith(Parameterized.class).
- Create a public static method annotated with @Parameters that returns a Collection of Objects (as Array) as test data set.
- Create a public constructor that takes in what is equivalent to one "row" of test data.
- Create an instance variable for each "column" of test data.
- Create your test case(s) using the instance variables as the source of the test data.

Mockito

A *mock object* is a dummy implementation for an interface or a class. It allows to define the output of certain method calls. They typically record the interaction with the system and tests can validate that.

stub : class that return dummy data(only for unit test)

SUT: System under test