Java Programming For Beginners

***New Sections**: Java New Features (10,..,15,16,..), Spring, Spring Boot and REST API*

Learn Java Programming

- **GOAL**: Help YOU learn Programming
  - **Basics** and **Best Practices**
  - Problem Solving
    - Simple **Design** and **Debugging**
  - Help you have fun!

Installing Java

- Step 01: Installing Java on Windows
- Step 02: Installing Java on MacOS
- Step 03: Installing Java on Linux
- Step 04: Troubleshooting
- Alternative:
  - ***https://tryjshell.org/***

Programming and Problem Solving

- I **love** programming:
  - You get to solve new problems every day.
  - Learn something new everyday!
- **Steps** in **Problem Solving**:
  - **Step I**: Understand the Problem
  - **Step II**: Design
    - Break the Problem Down
  - **Step III**: Write Your Program (and Test)
    - Express Your Solution: Language Specifics (Syntax)
- Let's solve multiple problems **step by step**!
- Learning to Program = Learning to ride a bike
  - First steps are the most difficult
  - Pure Fun afterwards!

Challenge 1 : Print Multiplication Table

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Where do we start? : Print Multiplication Table

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

- Step 1: Calculate value of "5 * 5"
- Step 2: Print "5 * 5 = 25"
- Step 3: Do this 10 times

JShell

- **Do you know?**: How do Python programmers start learning Python?
  - Python shell: That's why Python is easy to learn
- **From Java 9**: Java is equally easy to learn - JShell
  - Java REPL (Read Eval Print Loop)
  - Type in a one line of code and see the output
    - Makes learning fun (Make a mistake and it immediately tells you whats wrong!)
    - All great programmers make use of JShell
- **In this course**: We use JShell to get started
  - By Section 5, you will be comfortable with Java syntax
    - We will start using Eclipse as the Java IDE!

Java Primitive Types

Type of Values	Java Primitive Type	Size (in bits)	Range of Values	Example
Integral	byte	8	-128 to 127	byte b = 5;
Integral	short	16	-32,768 to 32,767	`short s = 128;`
Integral	int	32	-2,147,483,648 to 2,147,483,647	`int i = 40000;`
Integral	long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long l = 2222222222;
Float	float	32	±3.40282347E+38F. NOT precise	`float f = 4.0f`
Float	double	64	±1.79769313486231570E+308. NOT precise	double d = 67.0
Character	char	16	`'\u0000` to `'\uffff`	`char c = 'A';`
Boolean	boolean	1	`true` or `false`	`boolean isTrue = false;`

Print Multiplication Table - Solution 1

```
jshell> int i
i ==> 0
jshell> for (i=0; i<=10; i++) {
   ...> System.out.printf("%d * %d = %d", 5, i, 5*i).println();
   ...> }
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 2
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

JVM, JRE And JDK

- **JRE** = **JVM** + **Libraries** + **Other Components**
  - **JVM** runs your program bytecode
  - *Libraries* are built-in Java utilities that can be used within any program you create. `System.out.println()` was a method in `java.lang`, one such utility.
  - *Other Components* include tools for debugging and code profiling (for memory management and performance)
- **JDK** = **JRE** + **Compilers** + **Debuggers**
  - *JDK* refers to the **Java Development Kit**. It's an acronym for the bundle needed to compile (with the compiler) and run (with the *JRE* bundle) your Java program.
- Remember:
  - **JDK** is needed to **Compile and Run** Java programs
  - **JRE** is needed to **Run** Java Programs
  - **JVM** is needed to **Run Bytecode** generated from Java programs

Installing Eclipse

- Most Popular **Open Source** Java IDE
- Download:
  - ***https://www.eclipse.org/downloads/packages/***
- Recommended:
  - "Eclipse IDE for Enterprise Java and Web Developers"
- Troubleshooting
  - Use 7Zip if you have problems with unzipping
  - Unzip to root folder "C:\Eclipse" instead of a long path
  - Guide: ***https://wiki.eclipse.org/Eclipse/Installation#Troubleshooting***

Print Multiplication Table - Solution 2

```
public class MultiplicationTable {
    public static void print() {
        for(int i=1; i<=10;i++) {
            System.out.printf("%d * %d = %d", 5, i, 5*i).println();
        }
    }

    public static void print(int number) {
        for(int i=1; i<=10;i++) {
            System.out.printf("%d * %d = %d", number, i, number*i).println();
        }
    }

    public static void print(int number, int from, int to) {
        for(int i=from; i<=to;i++) {
            System.out.printf("%d * %d = %d", number, i, number*i).println();
        }
    }
}
```

Print Multiplication Table - Refactored (No Duplication)

```java
package com.in28minutes.firstjavaproject;

public class MultiplicationTable {
    public static void print() {
        print(5, 1, 10);
    }

    public static void print(int number) {
        print(number, 1, 10);
    }

    public static void print(int number, int from, int to) {
        for(int i=from; i<=to;i++) {
            System.out.printf("%d X %d = %d", number, i, number*i).println();
        }
    }
}
```

Object Oriented Programming (OOP)

```
class Planet
    name, location, distanceFromSun // data / state / fields
    rotate(), revolve() // actions / behavior / methods

earth : new Planet
venus : new Planet
```

- A **class** is a template.
  - In above example, `Planet` is a class
- An **object** is an instance of a class.
  - `earth` and `venus` are objects.
  - `name`, `location` and `distanceFromSun` compose object state.
  - `rotate()` and `revolve()` define object's behavior.
- **Fields** are the elements that make up the object state. Object behavior is implemented through **Methods**.

Object Oriented Programming (OOP) - 2

```
class Planet
    name, location, distanceFromSun // data / state / fields
    rotate(), revolve() // actions / behavior / methods

earth : new Planet
venus : new Planet
```

- Each Planet has its own state:
  - `name`: "Earth", "Venus"
  - `location` : Each has its own orbit
  - `distanceFromSun` : They are at unique, different distances from the sun
- Each Planet has its own unique behavior:
  - `rotate()` : They rotate at different rates (and in fact, different directions!)
  - `revolve()` : They revolve round the sun in different orbits, at different speeds

Next Few Sections

- Java keeps improving:
  - Java 10, Java 11, Java 12, ..., Java 17, Java 18 ...
- Developing Java Applications is Evolving as well:
  - Spring
  - Spring Boot
  - REST API
- How about building a Real World Java Project?
  - REST API with Spring and Spring Boot
- Let's get started!

Java Versioning

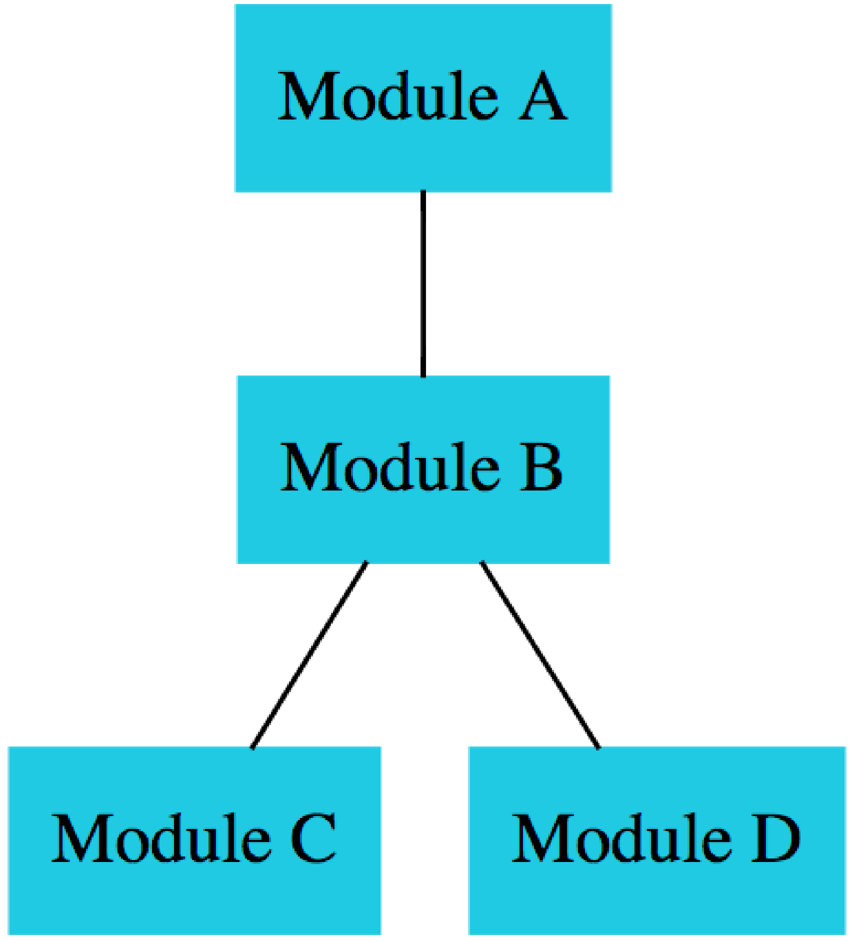
Version	Release Data	Notes
JDK 1.0	January 1996	
J2SE 5.0	September 2004	5 Releases in 8 years
Java SE 8 (LTS)	March 2014	Most important Java Release
Java SE 9	September 2017	4 Releases in 13 years
Java SE 10	March 2018	Time-Based Release Versioning
Java SE 11 (LTS)	September 2018	Long Term Support Version (Every 3 years)
Java SE 12	March 2019	
...		
Java SE 16	March 2021	
Java SE 17 (LTS)	September 2021	

Java New Features

Version	Release Data	Important New Features
J2SE 5.0	Sep 2004	Enhanced For Loop, Generics, Enums, Autoboxing
Java SE 8 (LTS)	Mar 2014	Functional Programming - Lambdas & Streams, Static methods in interface
Java SE 9	Sep 2017	Modularization (Java Platform Module System)
Java SE 10	Mar 2018	Local Variable Type Inference
Java SE 14	Mar 2020	Switch Expressions (Preview in 12 and 13)
Java SE 15	Sep 2020	Text Blocks (Preview in 13)
Java SE 16	Mar 2021	Record Classes (Preview in 14 and 15)
All Java Versions	-	API Improvements, Performance and Garbage Collection Improvements

Java Modularization - Overview

- Introduced in Java 9
- Goals:
  - Modularize JDK (IMPORTANT)
    - **rt.jar** grew to 60+ MB by **Java 8**
  - Modularize applications
- Modularizing JDK:
  - java --list-modules
    - java.base
    - java.logging
    - java.sql
    - java.xml
    - jdk.compiler
    - jdk.jartool
    - jdk.jshell
  - java -d java.sql



Java Modularization - Remember

- Module Descriptor - **module-info.java**: Defines metadata about the module:
  - **requires module.a;** - I need module.a to do my work!
  - **requires transitive module.a;** - I need module.a to do my work
    - AND my users also need access to module.a
  - **exports** - Export package for use by other modules
  - **opens package.b to module.a** - Before Java 9, reflection can be used to find details about types (private, public and protected). From Java 9, you can decide which packages to expose:
    - Above statement allows module.a access to perform reflection on public types in package.b
- **Advantages**
  - **Compile Time** Checks
    - For availability of modules
  - Better **Encapsulation**
    - Make only a subset of classes from a module available to other modules
  - **Smaller** Java Runtime
    - Use only the modules of Java that you need!

Local Variable Type Inference

```
// List<String> numbers = new ArrayList<>(list);
var numbers = new ArrayList<>(list);
```

- Java compiler infers the type of the variable at compile time
- Introduced in Java 10
- You can add final if you want
- `var` can also be used in loops
- Remember:
  - You cannot assign `null`
  - `var` is NOT a keyword
- Best Practices:
  - Good variable names
  - Minimize Scope
  - Improve readability for chained expressions

Switch Expression

```java
String monthName = switch (monthNumber) {
case 1 -> {
    System.out.println("January");
    // yield statement is used in a Switch Expression
    // break,continue statements are used in a Switch Statement
    yield "January"; // yield mandatory!
}
case 2 -> "February";
case 3 -> "March";
case 4 -> "April";
default -> "Invalid Month";
};
```

- Create expressions using switch statement
- Released in JDK 14
  - Preview - JDK 12 and 13
- Remember:
  - No fallthrough
  - Use `yield` or -> to return value

Text Blocks

```java
System.out.println("\"First Line\"\nSecond Line\nThird Line");
System.out.println("""
    "First Line"
        Second Line
        Third Line"""
        );
```

- Simplify Complex Text Strings
- Released in JDK 15
  - Preview - JDK 13 and 14
- Remember:
  - First Line : """ Followed by line terminator
    - """abc or """abc""" in First Line are NOT valid
  - Automatic Alignment is done
  - Trailing white space is stripped
  - You can use text blocks where ever you can use a String

Records

```
record Person(String name, String email, String phoneNumber) { }
```

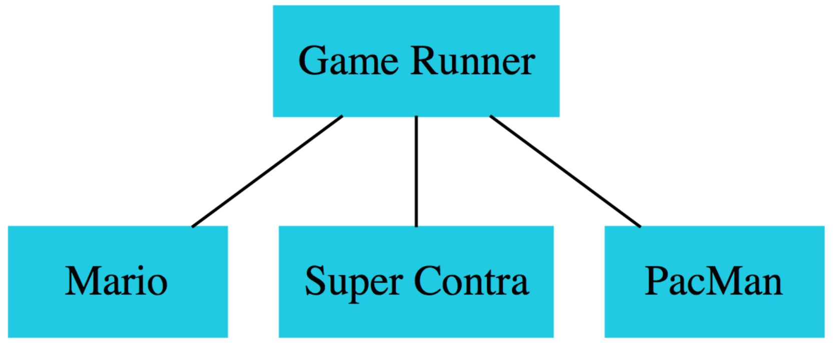
- Eliminate verbosity in creating Java Beans
  - Public accessor methods, constructor, equals, hashcode and toString are automatically created
  - You can create custom implementations if you would want
- Released in JDK 16
  - Preview - JDK 14 and 15
- Remember:
  - Compact Constructors are only allowed in Records
  - You can add static fields, static initializers, and static methods
    - BUT you CANNOT add instance variables or instance initializers
    - HOWEVER you CAN add instance methods

Getting Started with Spring Framework - Goals

- Build a Loose Coupled Hello World Gaming App with **Modern Spring** Approach
- Get **Hands-on** with Spring and understand:
  - Why Spring?
  - **Terminology**
    - Tight Coupling and Loose Coupling
    - IOC Container
    - Application Context
    - Component Scan
    - Dependency Injection
    - Spring Beans
    - Auto Wiring

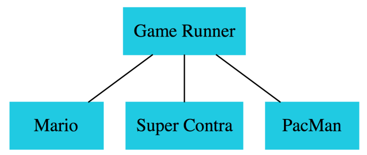
Loose Coupling with Spring Framework

- Design Game Runner to run games:
  - Mario, Super Contra, PacMan etc
- **Iteration 1**: Tightly Coupled
  - GameRunner class
  - Game classes: Mario, Super Contra, PacMan etc
- **Iteration 2**: Loose Coupling - Interfaces
  - GameRunner class
  - GamingConsole interface
    - Game classes: Mario, Super Contra, PacMan etc
- **Iteration 3**: Loose Coupling - Spring
  - Spring framework will manage all our objects!
    - GameRunner class
    - GamingConsole interface
      - Game classes: Mario, Super Contra, PacMan etc



Spring Framework - Questions

- **Question 1**: What's happening in the background?
  - Let's debug!
- **Question 2**: What about the terminology? How does it relate to what we are doing?
  - Dependency, Dependency Injection, IOC Container, Application Context, Component Scan, Spring Beans, Auto Wiring etc!
- **Question 3**: Does the Spring Framework really add value?
  - We are replacing 3 simple lines with 3 complex lines!
- **Question 4**: What if I want to run Super Contra game?
- **Question 5**: How is Spring JAR downloaded?
  - Magic of Maven!

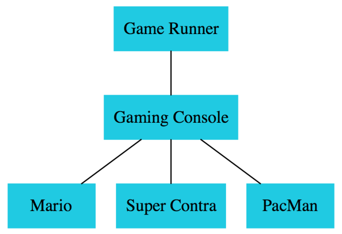


Question 1: What's happening in the background?

- Let's Debug:
  - Identified candidate component class: file [GameRunner.class]
  - Identified candidate component class: file [MarioGame.class]
  - Creating shared instance of singleton bean 'gameRunner'
  - Creating shared instance of singleton bean 'marioGame'
  - Autowiring by type from bean name 'gameRunner' via constructor to bean named 'marioGame'
  - org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'gameRunner' defined in file [GameRunner.class]
    - Unsatisfied dependency expressed through constructor parameter 0;
    - nested exception is:org.springframework.beans.factory.NoUniqueBeanDefinitionException
    - No qualifying bean of type 'com.in28minutes.learnspringframework.game.GamingConsole' available
    - expected single matching bean but found 3: marioGame,pacManGame,superContraGame

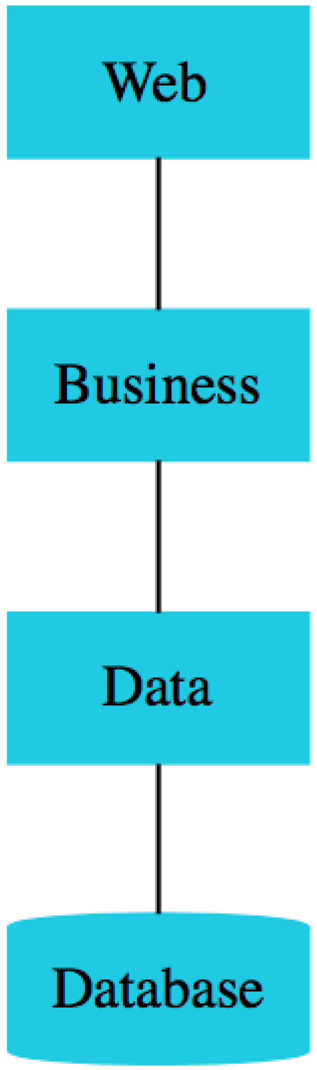
Question 2: Spring Framework - Important Terminology

- **@Component** (..): Class managed by Spring framework
- **Dependency**: GameRunner needs GamingConsole impl!
  - GamingConsole Impl (Ex: MarioGame) is a dependency of GameRunner
- **Component Scan**: How does Spring Framework find component classes?
  - It scans packages! (`@ComponentScan("com.in28minutes")`)
- **Dependency Injection**: Identify beans, their dependencies and wire them together (provides **IOC** - Inversion of Control)
  - **Spring Beans**: An object managed by Spring Framework
  - **IoC container**: Manages the lifecycle of beans and dependencies
    - **Types**: ApplicationContext (complex), BeanFactory (simpler features - rarely used)
  - **Autowiring**: Process of wiring in dependencies for a Spring Bean



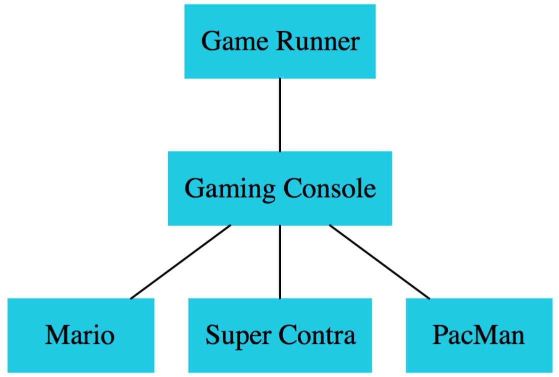
Question 3: Does the Spring Framework really add value

- In **Game Runner** Hello World App, we have very few classes
- BUT Real World applications **are much more complex**:
  - Multiple Layers (Web, Business, Data etc)
  - Each layer is **dependent** on the layer below it!
    - Example: Business Layer class talks to a Data Layer class
      - Data Layer class is a **dependency** of Business Layer class
    - There are thousands of such dependencies in every application!
- With Spring Framework:
  - INSTEAD of FOCUSING on objects, their dependencies and wiring
    - You can focus on the business logic of your application!
  - **Spring Framework manages the lifecycle** of objects:
    - Mark components using annotations: `@Component` (and others..)
    - Mark dependencies using `@Autowired`
    - Allow Spring Framework to do its magic!
- Ex: Controller > BusinessService (sum) > DataService (data)!



Question 4: What if I want to run Super Contra game?

- Try it as an exercise
  - @Primary
- Playing with Spring:
  - Exercise:
    - Dummy implementation for PacMan and make it Primary!
  - Debugging Problems:
    - Remove @Component and Play with it!

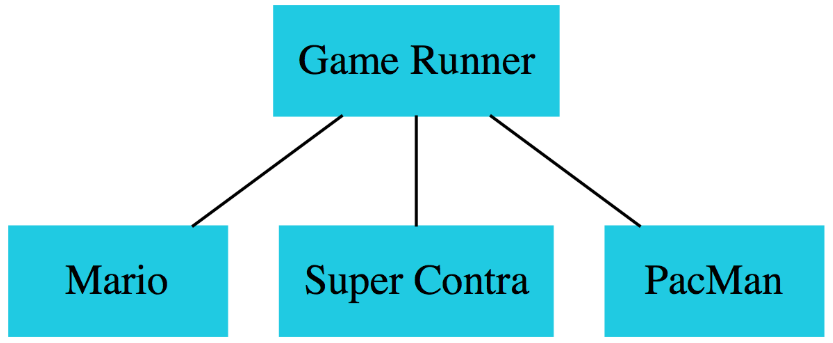


Question 5: How is Spring JAR downloaded? (Maven)

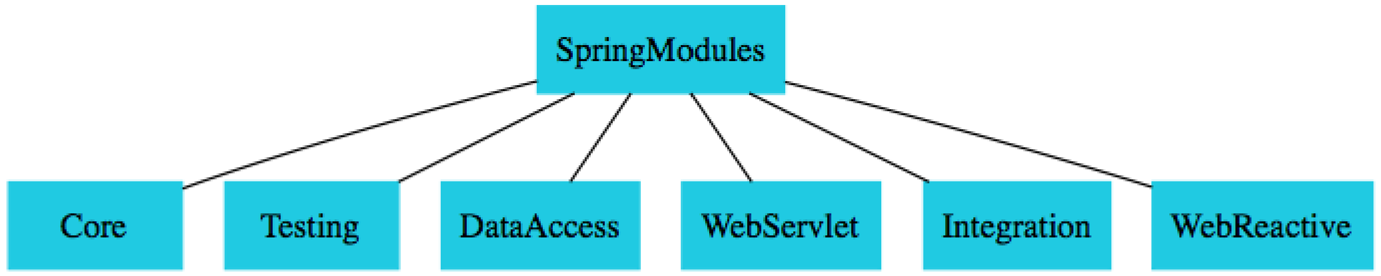
- What happens if you manually download Spring JAR?
  - Remember: Spring JAR needs other JARs
  - What if you need to upgrade to a new version?
- **Maven**: Manage JARs needed by apps (application dependencies)
  - Once you add a dependency on Spring framework, Maven would download:
    - Spring Framework and its dependencies
- All configuration in **pom.xml**
  - Maven artifacts: Identified by a Group Id, an Artifact Id!
- **Important Features**:
  - Defines a **simple project setup** that follows best practices
  - Enables **consistent usage** across all projects
  - Manages **dependency updates** and transitive dependencies
- **Terminology Warning**: Spring Dependency vs Maven Dependency

Exploring Spring - Dependency Injection Types

- **Constructor-based** : Dependencies are set by creating the Bean using its Constructor
- **Setter-based** : Dependencies are set by calling setter methods on your beans
- **Field**: No setter or constructor. Dependency is injected using reflection.
- Which one should you use?
  - Spring team recommends Constructor-based injection as dependencies are automatically set when an object is created!

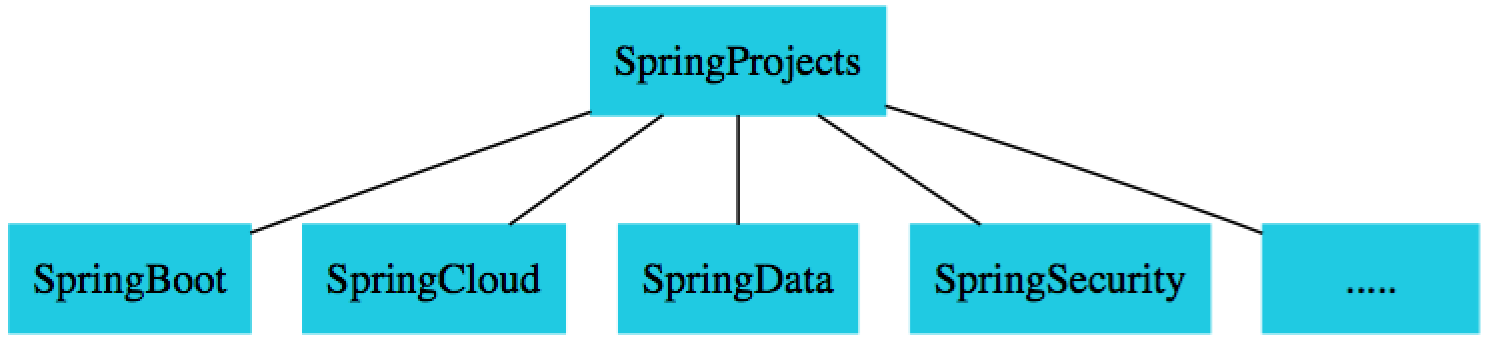


Spring Modules



- Spring Framework is divided into **modules**:
  - **Core**: IoC Container etc
  - **Testing**: Mock Objects, Spring MVC Test etc
  - **Data Access**: Transactions, JDBC, JPA etc
  - **Web Servlet**: Spring MVC etc
  - **Web Reactive**: Spring WebFlux etc
  - **Integration**: JMS etc
- Each application can choose the modules they want to make use of
  - They do not need to make use of all things everything in Spring framework!

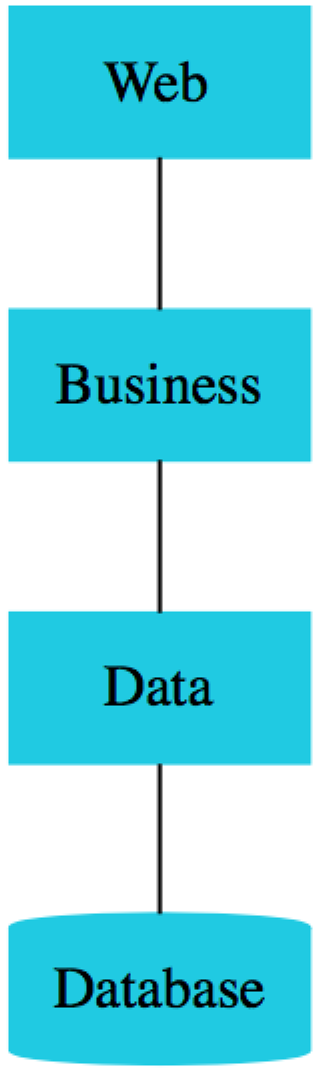
Spring Projects



- Spring Projects: Spring keeps evolving (REST API > Microservices > Cloud)
  - **Spring Boot:** Most popular framework to build microservices
  - **Spring Cloud**: Build cloud native applications
  - **Spring Data**: Integrate the same way with different types of databases : NoSQL and Relational
  - **Spring Integration:** Address challenges with integration with other applications
  - **Spring Security**: Secure your web application or REST API or microservice

Why is Spring Popular?

- **Loose Coupling**: Spring manages beans and dependencies
  - Make writing unit tests easy!
  - Provides its own unit testing project - Spring Unit Testing
- **Reduced Boilerplate Code**: Focus on Business Logic
  - Example: No need for exception handling in each method!
    - All Checked Exceptions are converted to Runtime or Unchecked Exceptions
- **Architectural Flexibility**: Spring Modules and Projects
  - You can pick and choose which ones to use (You DON'T need to use all of them!)
- **Evolution with Time**: Microservices and Cloud
  - Spring Boot, Spring Cloud etc!



Spring JDBC - Example

JDBC example

```
public void deleteTodo(int id) {
    PreparedStatement st = null;
    try {
        st = db.conn.prepareStatement(DELETE_TODO_QUERY);
        st.setInt(1, id);
        st.execute();
    } catch (SQLException e) {
        logger.fatal("Query Failed : " + DELETE_TODO_QUERY, e);
    } finally {
        if (st != null) {
            try {st.close();}
            catch (SQLException e) {}
        }
    }
}
```

Spring JDBC example

```
public void deleteTodo(int id) {
    jdbcTemplate.update(DELETE_TODO_QUERY, id);
}
```

Spring Framework - Review

- **Goal**: 10,000 Feet overview of Spring Framework
  - Help you understand the terminology!
    - Dependency
    - Dependency Injection (and types)
      - Autowiring
      - Spring Beans
      - Component Scan
    - IOC Container (Application Context)
  - We will play with other Spring Modules and Projects later in the course
- **Advantages**: Loosely Coupled Code (Focus on Business Logic), Architectural Flexibility and Evolution with time!

Getting Started with Spring Boot - Goals

- Build a Hello World App in **Modern Spring Boot** Approach
- Get **Hands-on** with Spring Boot
  - Why Spring Boot?
  - **Terminology**
    - Spring Initializr
    - Auto Configuration
    - Starter Projects
    - Actuator
    - Developer Tools

Hands-on: Understand Power of Spring Boot

```
// http://localhost:8080/courses
[
  {
    "id": 1,
    "name": "Learn Microservices",
    "author": "in28minutes"
  }
]
```

- Let's Build a **Hello World App** using **Spring Initializr**
- Setup **BooksController**

World Before Spring Boot!

https://github.com/in28minutes/SpringMvcStepByStep/blob/master/Step15.md#pomxml

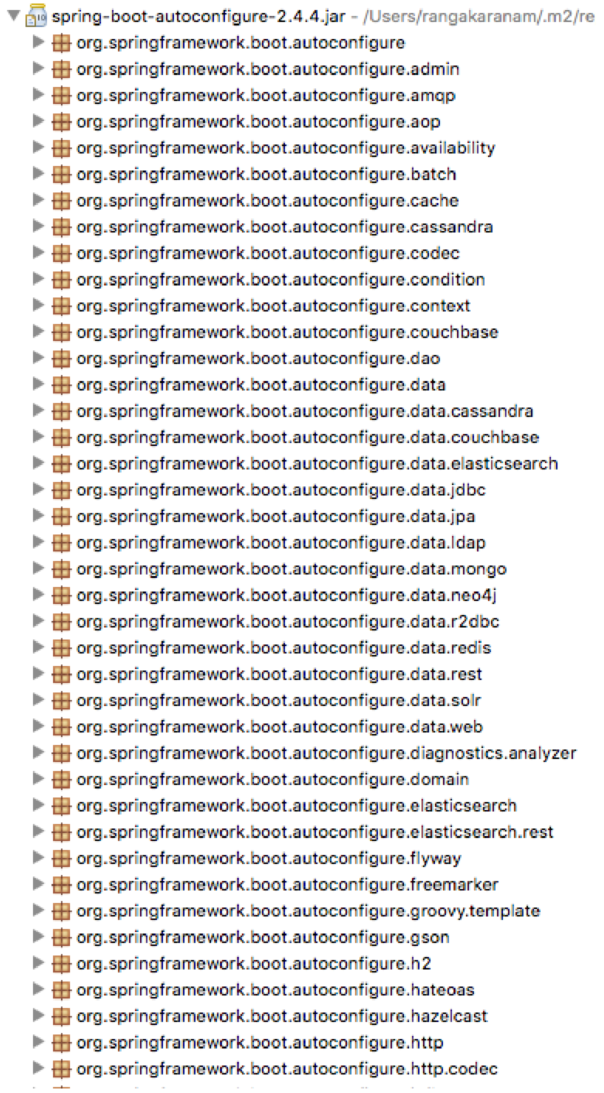
- Setting up Spring Web Projects **before Spring Boot was NOT easy**!
  - **Define maven dependencies** and manage versions for frameworks
    - spring-webmvc, jackson-databind, log4j etc
  - Define **web.xml** (`/src/main/webapp/WEB-INF/web.xml`)
    - Define Front Controller for Spring Framework (`DispatcherServlet`)
  - Define a **Spring context XML** file (`/src/main/webapp/WEB-INF/todo-servlet.xml`)
    - Define a Component Scan (`<context:component-scan base-package="com.in28minutes" />`)
  - **Install Tomcat** or use tomcat7-maven-plugin plugin (or any other web server)
  - Deploy and Run the application in Tomcat
- How does Spring Boot do its **Magic**?
  - Spring Boot Starter Projects
  - Spring Boot Auto Configuration

Spring Boot Starter Projects

- **Goal of Starter Projects**: Help you get a project up and running quickly!
  - **Web Application** - Spring Boot Starter Web
  - **REST API** - Spring Boot Starter Web
  - **Talk to database using JPA** - Spring Boot Starter Data JPA
  - **Talk to database using JDBC** - Spring Boot Starter JDBC
  - **Secure your web application or REST API** - Spring Boot Starter Security
- Manage list of **maven dependencies** and versions for different kinds of apps:
  - **Spring Boot Starter Web**: Frameworks needed by typical web applications
    - spring-webmvc, spring-web, spring-boot-starter-tomcat, spring-boot-starter-json

Spring Boot Auto Configuration

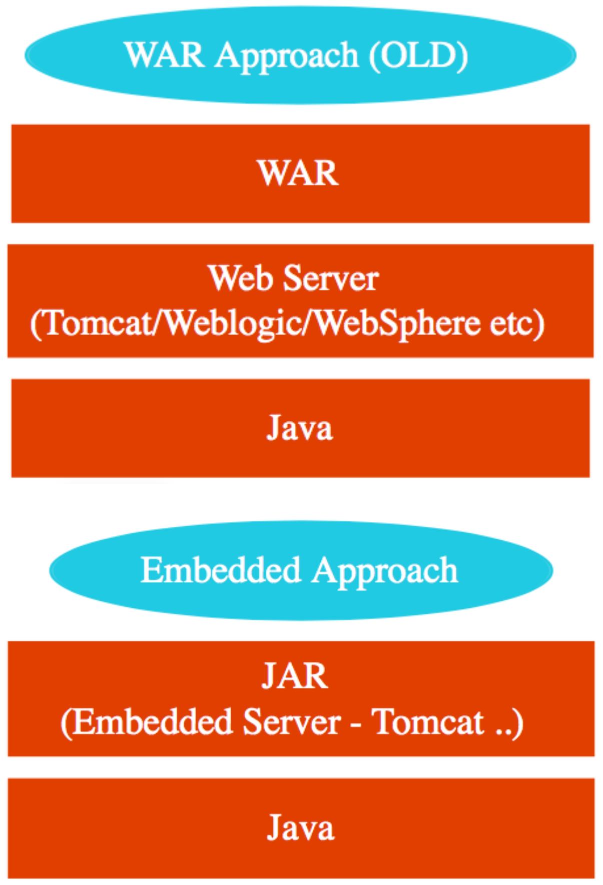
- Spring Boot provides **Auto Configuration**
  - **Basic configuration** to run your application using the frameworks defined in your maven dependencies
  - Auto Configuration is **decided based on:**
    - Which frameworks are in the Class Path?
    - What is the existing configuration (Annotations etc)?
  - **An Example**: (Enable debug logging for more details):
    - If you use Spring Boot Starter Web, following are auto configured:
      - Dispatcher Servlet (`DispatcherServletAutoConfiguration`)
      - Embedded Servlet Container - Tomcat is the default (`EmbeddedWebServerFactoryCustomizerAutoConfiguration`)
      - Default Error Pages (`ErrorMvcAutoConfiguration`)
      - Bean to/from JSON conversion (`JacksonHttpMessageConvertersConfiguration`)



```
▼ spring-boot-autoconfigure-2.4.4.jar - /Users/rangakaranam/.m2/re
  ▶ org.springframework.boot.autoconfigure
  ▶ org.springframework.boot.autoconfigure.admin
  ▶ org.springframework.boot.autoconfigure.amqp
  ▶ org.springframework.boot.autoconfigure.aop
  ▶ org.springframework.boot.autoconfigure.availability
  ▶ org.springframework.boot.autoconfigure.batch
  ▶ org.springframework.boot.autoconfigure.cache
  ▶ org.springframework.boot.autoconfigure.cassandra
  ▶ org.springframework.boot.autoconfigure.codec
  ▶ org.springframework.boot.autoconfigure.condition
  ▶ org.springframework.boot.autoconfigure.context
  ▶ org.springframework.boot.autoconfigure.couchbase
  ▶ org.springframework.boot.autoconfigure.dao
  ▶ org.springframework.boot.autoconfigure.data
  ▶ org.springframework.boot.autoconfigure.data.cassandra
  ▶ org.springframework.boot.autoconfigure.data.couchbase
  ▶ org.springframework.boot.autoconfigure.data.elasticsearch
  ▶ org.springframework.boot.autoconfigure.data.jdbc
  ▶ org.springframework.boot.autoconfigure.data.jpa
  ▶ org.springframework.boot.autoconfigure.data.ldap
  ▶ org.springframework.boot.autoconfigure.data.mongo
  ▶ org.springframework.boot.autoconfigure.data.neo4j
  ▶ org.springframework.boot.autoconfigure.data.r2dbc
  ▶ org.springframework.boot.autoconfigure.data.redis
  ▶ org.springframework.boot.autoconfigure.data.rest
  ▶ org.springframework.boot.autoconfigure.data.solr
  ▶ org.springframework.boot.autoconfigure.data.web
  ▶ org.springframework.boot.autoconfigure.diagnostics.analyzer
  ▶ org.springframework.boot.autoconfigure.domain
  ▶ org.springframework.boot.autoconfigure.elasticsearch
  ▶ org.springframework.boot.autoconfigure.elasticsearch.rest
  ▶ org.springframework.boot.autoconfigure.flyway
  ▶ org.springframework.boot.autoconfigure.freemarker
  ▶ org.springframework.boot.autoconfigure.groovy.template
  ▶ org.springframework.boot.autoconfigure.gson
  ▶ org.springframework.boot.autoconfigure.h2
  ▶ org.springframework.boot.autoconfigure.hateoas
  ▶ org.springframework.boot.autoconfigure.hazelcast
  ▶ org.springframework.boot.autoconfigure.http
  ▶ org.springframework.boot.autoconfigure.http.codec
```

Spring Boot Embedded Servers

- How do you deploy your application?
  - Step 1 : Install Java
  - Step 2 : Install Web/Application Server
    - Tomcat/WebSphere/WebLogic etc
  - Step 3 : Deploy the application WAR (Web ARchive)
    - This is the OLD **WAR** Approach
    - Complex to setup!
- **Embedded Server** - Simpler alternative
  - Step 1 : Install Java
  - Step 2 : Run **JAR** file
  - **Make JAR not WAR** (Credit: Josh Long!)
  - Embedded Server **Examples**:
    - spring-boot-starter-tomcat
    - spring-boot-starter-jetty
    - spring-boot-starter-undertow


WAR Approach (OLD)
WAR
Web Server (Tomcat/Weblogic/WebSphere etc)
Java
Embedded Approach
JAR (Embedded Server - Tomcat ..)
Java

More Spring Boot Features

- **Spring Boot Actuator**: Monitor and manage your application in your production
  - Provides a number of endpoints:
    - **beans** - Complete list of Spring beans in your app
    - **health** - Application health information
    - **metrics** - Application metrics
    - **mappings** - Details around Request Mappings
- **Spring Boot DevTools**: Increase developer productivity
  - Why do you need to restart the server for every code change?

Spring Boot vs Spring MVC vs Spring

- **Spring Framework** Core Feature: Dependency Injection
  - @Component, @Autowired, IOC Container, ApplicationContext, Component Scan etc..
  - **Spring Modules and Spring Projects:** Good Integration with Other Frameworks (Hibernate/JPA, JUnit & Mockito for Unit Testing)
- **Spring MVC** (Spring Module): Build web applications in a decoupled approach
  - Dispatcher Servlet, ModelAndView and View Resolver etc
- **Spring Boot** (Spring Project): Build production ready applications quickly
  - **Starter Projects** - Make it easy to build variety of applications
  - **Auto configuration** - Eliminate configuration to setup Spring, Spring MVC and other projects!
  - Enable production ready non functional features:
    - Actuator : Enables Advanced Monitoring and Tracing of applications.
    - Embedded Servers - No need for separate application servers!
    - Default Error Handling

Spring Boot - Review

- **Goal:** 10,000 Feet overview of Spring Boot
  - Help you understand the terminology!
    - Starter Projects
    - Auto Configuration
    - Actuator
    - DevTools
- **Advantages:** Get started quickly with production ready features!

REST API

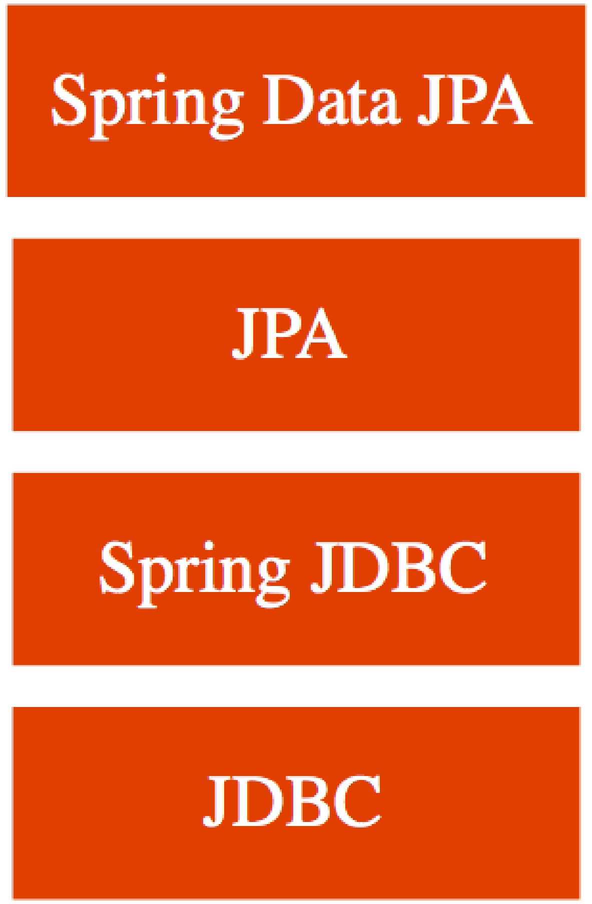
- **REST API**: Architectural Style for the Web
  - **Resource**: Any information (Example: Courses)
  - **URI**: How do you identify a resource? (/courses, /courses/1)
  - You can perform actions on a resource (Create/Get/Delete/Update). Different HTTP Request Methods are used for different operations:
    - **GET** - Retrieve information (/courses, /courses/1)
    - **POST** - Create a new resource (/courses)
    - **PUT** - Update/Replace a resource (/courses/1)
    - **PATCH** - Update a part of the resource (/courses/1)
    - **DELETE** - Delete a resource (/courses/1)
  - **Representation**: How is the resource represented? (XML/JSON/Text/Video etc..)
  - **Server**: Provides the service (or API)
  - **Consumer**: Uses the service (Browser or a Front End Application)

Spring and Spring Boot Release Cycles

- What is the **difference** between these?
  - 2.5.0 (SNAPSHOT)
  - 2.4.5 (M3)
  - 2.4.4
- Release Number: MAJOR.MINOR.FIX
- Spring and Spring Boot **Release Cycle**:
  - SNAPSHOT (versions under development) > Mile Stones > Released Version
- **Recommendation** - Do NOT use SNAPSHOTs or M1 or M2 or M3
  - Prefer released versions!

JDBC to Spring JDBC to JPA to Spring Data JPA

- **JDBC**
  - Write a lot of SQL queries!
  - And write a lot of Java code
- **Spring JDBC**
  - Write a lot of SQL queries
  - BUT lesser Java code
- **JPA**
  - Do NOT worry about queries
  - Just Map Entities to Tables!
- **Spring Data JPA**
  - Let's make JPA even more simple!
  - I will take care of everything!



JDBC to Spring JDBC

JDBC example

```
public void deleteTodo(int id) {
    PreparedStatement st = null;
    try {
        st = db.conn.prepareStatement(DELETE_TODO_QUERY);
        st.setInt(1, id);
        st.execute();
    } catch (SQLException e) {
        logger.fatal("Query Failed : " + DELETE_TODO_QUERY, e);
    } finally {
        if (st != null) {
            try {st.close();}
            catch (SQLException e) {}
        }
    }
}
```

Spring JDBC example

```
public void deleteTodo(int id) {
    jdbcTemplate.update(DELETE_TODO_QUERY, id);
}
```

JPA Example

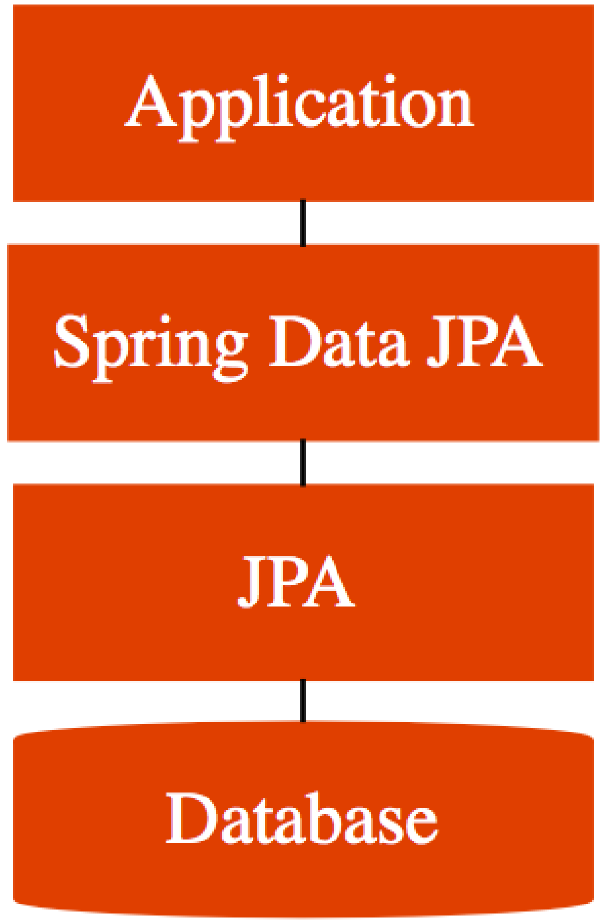
```
@Repository
@Transactional
public class PersonJpaRepository {

  @PersistenceContext
  EntityManager entityManager;

  public Person findById(int id) {
    return entityManager.find(Person.class, id);
  }

  public Person update(Person person) {
    return entityManager.merge(person);
  }

  public Person insert(Person person) {
    return entityManager.merge(person);
  }

  public void deleteById(int id) {........
```

Spring Data JPA Example

```
public interface TodoRepository extends JpaRepository<Todo, Integer>{
```

Spring Boot Auto Configuration Magic - Data JPA

- We added Data JPA and H2 dependencies:
  - Spring Boot Auto Configuration does some magic:
    - Initialize JPA and Spring Data JPA frameworks
    - Launch an in memory database (H2)
    - Setup connection from App to in-memory database
    - Launch a few scripts at startup (example: `data.sql`)
- **Remember** - H2 is in memory database
  - Does NOT persist data
  - Great for learning
  - BUT NOT so great for production
  - Let's see how to use MySQL next!



Congratulations

- Java keeps improving:
  - Java 10, Java 11, Java 12, ...
- Java Project - REST API in Modern Approach:
  - Spring
  - Spring Boot