In28 Minutes

Java Programming For Beginners

***New Sections**: Java New Features (23,24,..), Spring, Spring Boot and REST API*

Learn Java Programming

In28 Minutes

- **GOAL**: Help YOU learn Programming
  - **Basics** and **Best Practices**
  - Problem Solving
    - Simple **Design** and **Debugging**
  - Help you have fun!
- **APPROACH**: **Hands-on** Step By Step
  - Learn Problem Solving
  - Practice **200+** Code Examples
  - Test Your Learning: **Exercises**
  - Learn to Debug Programs : Github Page
  - Build real world applications
- By the end of the course, you will be a **really good programmer**!

YOUR Success = OUR Success

- **98,000+ Learners** with 46% **5 STAR** Reviews
  - Last Year: **42,000+ active learners** & 14 million learning minutes
  - "Great mix of theory and exercises!"
  - "Interactive learning with the help of puzzles"
  - "Never thought taking an online course will be so helpful. "
  - "Builds confidence in people who fear programming"
- **RECOMMENDATION**: Bit of Patience in the first hour!

FASTEST ROADMAPS

in28minutes.com

In28 Minutes

- Google Cloud Certifications
- Azure Certifications
- AWS Certifications
- DevOps
- Java Full Stack
- Java Microservices

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

In 28 Minutes

- 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

In 28 Minutes

```
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

In 28 Minutes

```
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

In28 Minutes

- **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

In 28 Minutes

Values	Primitive Type	Size (bits)	Range	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';`

Print Multiplication Table - Solution 1

In28 Minutes

```
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

In28 Minutes

- **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

In28 Minutes

- 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

In28 Minutes

```
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)

In28 Minutes

```
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)

In 28 Minutes

```
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

In28 Minutes

```
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

In 28 Minutes

- 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!

How Java Stays Relevant

In28 Minutes

Version	Release Date	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

How Java Stays Relevant - 2

In28 Minutes

Version	Release Date	Notes
Java SE 11 (LTS)	September 2018	Long Term Support Version
Java SE 12	March 2019	
...		
Java SE 21 (LTS)	September 2023	Long Term Support Version
...		
Java SE 25 (LTS)	September 2025	Long Term Support Version
...		

Exploring Most Important Java Releases

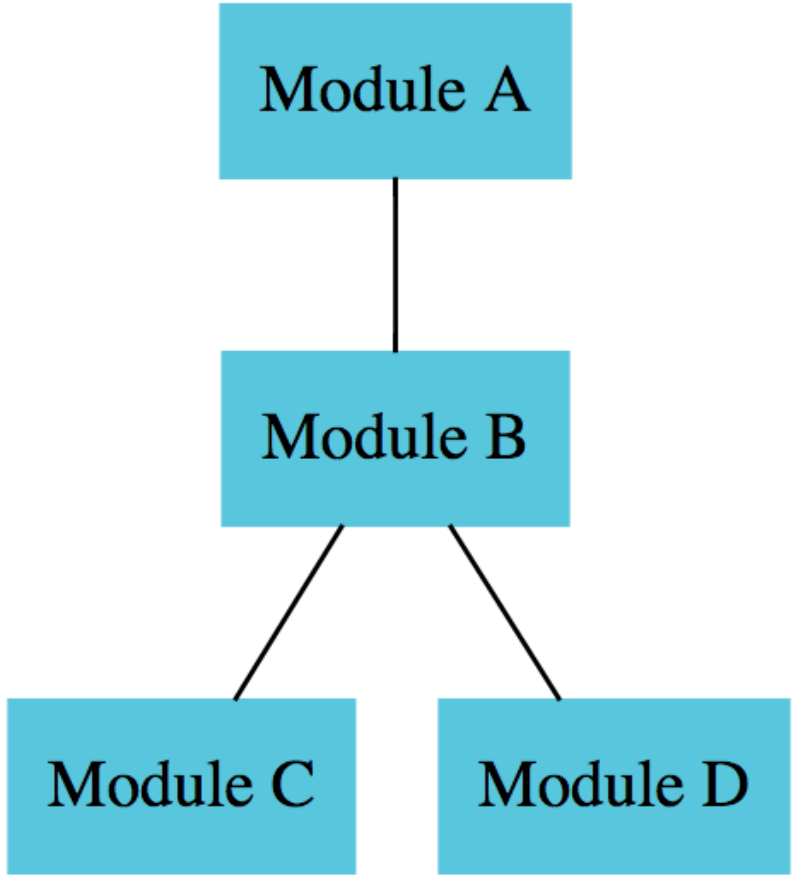
In28 Minutes

Version	Release Date	Key New Features
J2SE 5.0	Sep 2004	Enhanced For Loop, Generics, Enums, Autoboxing
Java SE 8	Mar 2014	Functional Programming - Lambdas & Streams, Static methods in interface
Java SE 9	Sep 2017	Modularization (Java Platform Module System)
Java SE 16	Mar 2021	Record Classes
Java SE 21	Sep 2023	Virtual Threads and Sequenced Collections
Java SE 24	Mar 2025	Stream Gatherers
All Java Versions	-	API Improvements, Performance and Garbage Collection Improvements

Java Modularization - Overview

In28 Minutes

- 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

In28 Minutes

- 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

In28 Minutes

```
// 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

In28 Minutes

```
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

In 28 Minutes

```
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

In 28 Minutes

```
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

What is a Sealed Class? (Java 17)

In28 Minutes

```
sealed class Vehicle permits Car, Truck, Bike {}

// No further subclassing
final class Car extends Vehicle {}

// Can be extended freely
non-sealed class Truck extends Vehicle {}

// Further restricted
sealed class Bike extends Vehicle permits ElectricBike {}

// No further subclassing
final class ElectricBike extends Bike {}
```

- **Only allowed subclasses** (`Car`, `Truck`, `Bike`) can extend `Vehicle`.
- You can choose the restrictions on sub-classes:
  - `Car` is **final**, so it **cannot be extended further**.
  - `Truck` is **non-sealed**, so **any class can extend it**.
  - `Bike` is **sealed** again, allowing only `ElectricBike` to extend it.

What is a Sealed Interface? (Java 17)

In28 Minutes

```
sealed interface Flyable permits Bird, Aeroplane, Helicopter {}

// Cannot be extended
final class Bird implements Flyable {}

// Restricted
sealed class Aeroplane implements Flyable permits Boeing {}

// Can be freely implemented
non-sealed class Helicopter implements Flyable {}

// No further subclassing
final class Boeing extends Aeroplane {}
```

- **Controls which classes can implement `Flyable`.**
- You can choose the restrictions on implementations:
  - `Bird` is **final**, so no other class can extend it.
  - `Aeroplane` is **sealed**, so only `Boeing` can extend it.
  - `Helicopter` is **non-sealed**, so any class can extend it.

Things You Should Know - Sealed Classes & Interfaces

In 28 Minutes

```
sealed class Vehicle permits Car, Truck, Bike {}

sealed interface Flyable permits Bird, Aeroplane, Helicopter {}

// Cannot be extended
final class Bird implements Flyable {}
```

- **Prevents accidental subclassing & implementations** – Only explicitly permitted classes can extend or implement
- **Improves readability and maintainability** – Clearly defines allowed subtypes
- Helps enforce **strict type hierarchies** in large applications
- Subclasses/Implementations must be **`final`**, **`non-sealed`**, or **`sealed`**

How are Threads Traditionally Implemented in Java?

In28 Minutes

```
public class SleepingThread implements Runnable {
    public void run() {
        try { TimeUnit.SECONDS.sleep(1);}
        catch (Exception ex) {}
    }
}
public class PlaformThreadLimitsTester {
    public static void main(String[] args) {
        for (int i = 0; i < 1000000; i++) {
            System.out.println(i);
            new Thread(new SleepingThread()).start();
        }
    }
}
```

- **How?** Each Java thread maps to an OS thread.
- **Issue?** Limited scalability
  - Limited number of OS threads.
  - Too many threads => high memory
  - Above program would fail due to Out Of Memory Exception

Problem: 1 Java Thread maps to 1 OS Thread

In28 Minutes

Problem	Why It's Bad
High Memory Usage	Each OS thread needs ~1MB memory. 10,000 threads → ~10GB RAM wasted!
Limited Scalability	OS cannot handle hundreds of thousands of threads.
IO-Bound Inefficiency	If a thread waits for API/DB, the corresponding OS thread is blocked doing nothing.

What are Virtual Threads? (Java 21)

In28 Minutes

```
public class VirtualThreadLimitsTester {
    public static void main(String[] args) {
        for (int i = 0; i < 1000000; i++) {
            System.out.println(i);
            Thread.startVirtualThread(new SleepingThread());
        }
    }
}
```

- **What?** 1000s of Virtual threads can run on top of a small pool of OS threads.
- **Who Manages?** Managed by the Java Virtual Machine (JVM)
- **Benefit?** Create millions of lightweight threads without high memory usage.

Advantages of Virtual Threads (Java 21)

In28 Minutes

Feature	Virtual Threads (New)
Memory Usage	Low (Thousands to millions of threads)
Context Switching	Cheap
Scalability	Very high

Platform Threads vs Virtual Threads (Java 21)

In28 Minutes

Feature	Platform Threads (Old)	Virtual Threads (New)
Implementation	Managed by the Operating System (OS)	Managed by the JVM
Thread Count	Limited by the number of OS threads	Can have a lot more, even millions
Thread Creation	Expensive	Very cheap
Context Switching	Expensive	Cheap
OS thread usage efficiency	Low	High

Using Executor Service To Launch Threads (Java 21)

In 28 Minutes

```
public class VirtualThreadsWithExecutorService {
    public static void main(String[] args) {
        ExecutorService executor
            = Executors.newVirtualThreadPerTaskExecutor();
        for (int i = 0; i < 1000000; i++) {
            System.out.println(i);
            executor.execute(new SleepingThread());
        }
        executor.shutdown();
    }
}
```

- **More resource-friendly**: Built-in pooling and scheduling
- **Preferred Approach**: Preferred way for launching many short-lived tasks
- **Avoid Manual Management**: You don't need to manage threads manually

Platform Threads vs Virtual Threads - When? (Java 21)

In28 Minutes

```
ExecutorService executor = Executors.newVirtualThreadPerTaskExecutor();
executor.submit(() -> {
            System.out.println("Hello from " + Thread.currentThread());
        });
executor.shutdown();

Thread vt = Thread.ofVirtual().start(
                    () -> System.out.println("Running in a Virtual Thread"));
```

- **When to Use Traditional Java Threads?**
  - **CPU-Bound Tasks Needing Low Number of Threads** – Heavy computations (e.g., image processing, data crunching) needing only a few threads
- **When to Use Virtual Threads?**
  - **IO-Bound Tasks** – Database queries, API calls, file reads.
  - **Massive Concurrency** – Millions of tasks (e.g., handling web requests).
  - **Efficient Resource Usage** – No wasted OS threads waiting for IO.

Need For Sequenced Collections (Java 21)

In28 Minutes

```
// Get the first element of the list
String firstElement = list.get(0);
// Get the last element of the list
int lastIndex = list.size() - 1;
String lastElement = list.get(lastIndex);
```

- Imagine **getting the first element** of collection
  - It's pretty simple
- Imagine **accessing last element** of collection
  - It's a little bit more complex
- What if there is a Java API that makes it simple?
  - **Solution**: `SequencedCollection` Interface (along with `SequencedSet` and `SequencedMap`)
  - **Widely Implemented**: Almost all important collection implementations support some form of sequenced collections operations (`ArrayList`, `LinkedList`,`HashSet`)
  - **Tip**: Also allows to access the collection in a **reversed** order

Exploring API of Sequenced Collections (Java 21)

In28 Minutes

```
interface SequencedCollection<E> extends Collection<E> {

    void addFirst(E); // Add as first element in the collection
    void addLast(E); // Add as last element in the collection

    E getFirst(); // Get first element from the collection
    E getLast(); // Get last element from the collection

    E removeFirst(); // remove first element from the collection
    E removeLast(); // remove last element from the collection

    SequencedCollection<E> reversed();

}
```

- **SequencedCollection**: Extends Collection
  - Adds a few new methods

Exploring SequencedCollections with List Example

In28 Minutes

```
var courseDetails = new ArrayList<>();

courseDetails.add("Spring Security");

courseDetails.addFirst("Spring Boot");
courseDetails.addLast("Spring Boot in AI");

courseDetails.add("Cloud Computing with AWS");
courseDetails.add("Cloud Computing with Azure");

System.out.println("First Element:: " + courseDetails.getFirst());
System.out.println("Last Element:: " + courseDetails.getLast());

System.out.println("Remove First Element:: " + courseDetails.removeFirst());
System.out.println("Remove Last Element:: " + courseDetails.removeLast());

var reverseCourseDetails = courseDetails.reversed();
System.out.println("Reversed List:: "+reverseCourseDetails);
```

Exploring API of SequencedSet

In28 Minutes

```
interface SequencedSet<E> extends Set<E>, SequencedCollection<E> {
    SequencedSet<E> reversed();    // covariant override
}

var courses = List.of("Spring", "Spring Boot", "AWS", "Azure");
var courseDetailsSet = new LinkedHashSet<>(courses);

System.out.println("First Element:: "+courseDetailsSet.getFirst());
System.out.println("Last Element:: "+courseDetailsSet.getLast());
System.out.println("Order:: "+courseDetailsSet);

System.out.println("Adding new elements");
courseDetailsSet.addFirst("Spring Security");
courseDetailsSet.addLast("Spring AI");
System.out.println("Order:: "+courseDetailsSet);
System.out.println("Reversed:: "+courseDetailsSet.reversed());
```

Exploring SequencedMap

In28 Minutes

```
interface SequencedMap<K,V> extends Map<K,V> {
    // new methods
    SequencedMap<K,V> reversed(); // Reverse the map
    SequencedSet<K> sequencedKeySet(); // Get the sequenced key set
    SequencedCollection<V> sequencedValues(); // Get the sequenced values
    SequencedSet<Entry<K,V>> sequencedEntrySet(); // Get the entrySet
    V putFirst(K, V); // Add as first element in the Map
    V putLast(K, V); // Add as last element in the Map

    // methods promoted from NavigableMap
    Entry<K, V> firstEntry();
    Entry<K, V> lastEntry();
    Entry<K, V> pollFirstEntry();
    Entry<K, V> pollLastEntry();
}
```

- SequencedMap extends Map
  - Several new methods

Exploring SequencedMap - LinkedHashMap

In28 Minutes

```
var courseDetails = new LinkedHashMap<>();
courseDetails.put(1, "Spring");
courseDetails.put(2, "Spring Boot");
courseDetails.put(3, "Spring AI");
System.out.println("Map::" + courseDetails);

System.out.println("Adding Elements");
courseDetails.putFirst(10, "Spring Security");
courseDetails.putLast(20, "Spring Cloud");
System.out.println("Map::" + courseDetails);

System.out.println("sequencedKeySet:: " + courseDetails.sequencedKeySet());
System.out.println("sequencedValues:: " + courseDetails.sequencedValues());
System.out.println("sequencedEntrySet:: " + courseDetails.sequencedEntrySet());
System.out.println("First Entry:: " + courseDetails.firstEntry());
System.out.println("Last Entry:: " + courseDetails.lastEntry());

System.out.println("First Poll Entry:: " + courseDetails.pollFirstEntry());
System.out.println("Map::" + courseDetails);
System.out.println("Last Poll Entry:: " + courseDetails.pollLastEntry());
System.out.println("Map::" + courseDetails);

System.out.println("Reversed:: " + courseDetails.reversed());
```

Pattern Matching with Variable Declaration (Java 14)

In28 Minutes

```
public void process(Object obj) {

    //if (obj instanceof String) {
        //String s = (String) obj;  // Explicit casting needed
        //System.out.println("Message: " + s);
    //}


    //instanceof + variable declaration!
    if (obj instanceof String s) {
        System.out.println("Message: " + s);  // No explicit cast needed
    }

}
```

- Simplified **`instanceof` Check**
  - Eliminates explicit casting.
  - Declares variable inline.
  - Cleaner and safer.

Pattern Matching with Records (Java 21)

In 28 Minutes

```
record Transaction(String sender, String receiver, int amount) {}

public void processTransaction(Object obj) {

    if (obj instanceof Transaction(
            String sender, String receiver, int amount)) {

        System.out.println("Processing transaction: " + sender
                           + " -> " + receiver + " : $" + amount);

    }

}
```

- Supports **automatic deconstruction** of records.
- Automatically **extracts fields** (`sender`, `receiver`, `amount`).

Pattern Matching with Nested Records (Java 21)

In28 Minutes

```
package com.in28minutes;

record Customer(String name, String email) {}

record Product(String name, double price) {}

record Order(Customer customer, Product product) {}


public class CustomerSupport {
    public static void processOrder(Object obj) {
        if (obj instanceof Order(
                Customer(String name, String email),
                Product(String productName, double price))) {
            System.out.println("Customer " + name +
                    " ordered " + productName + " for $" + price);
        }
    }

    public static void main(String[] args) {
        processOrder(new Order(new Customer("Ranga","Email"),
                new Product("Cricket Bat",100)));
    }
}
```

Switch with Enums (Java 17)

In28 Minutes

```
DayOfWeek dayOfWeek = DayOfWeek.FRIDAY;

String message = switch (dayOfWeek) {
    case MONDAY -> "Getting Started With My Week";
    case TUESDAY, WEDNESDAY, THURSDAY -> "Mid of the Week";
    case FRIDAY -> "Getting Ready For Weekend";
    default -> "Enjoying the Weekend";
};
```

- Enum + Switch = Magic!
- Eliminates **repetitive if-else** logic.
- Ensures all Enum values are handled!

Switch with Record Patterns (Java 21)

In28 Minutes

```
sealed interface CustomerMessage permits Message, Feedback{}
final record Message(String text) implements CustomerMessage {}
final record Feedback(double rating, String description) implements CustomerMessage {}

CustomerMessage message = new Message("I need help");

String response = switch (message) {
    case Message(String text)
        -> "Processing message: " + text;
    case Feedback(double rating, String description)
        -> "Processing feedback rating: " + rating + " stars" +
                        " with description" + description;
};

System.out.println(response);
```

- Allows direct **record deconstruction**.
- More concise and readable code.

Nested Record Patterns in Switch (Java 21)

In28 Minutes

```
record SupportRequest(String user, CustomerMessage message) {}

SupportRequest request = new SupportRequest("Rahul", new Message("I need help"));

String response = switch (request) {
    case SupportRequest(String user, Message(String text))
        -> "User " + user + " sent a message : #" + text;
    case SupportRequest(String user, Feedback(double rating, String description))
        -> "User " + user + " left a rating: " + rating + " stars" +
                    " with description " + description;
};

System.out.println(response);
```

- **Automatic deconstruction** of records
- Work seamlessly with nested records

Getting Started with Stream Gatherers (Java 24)

In 28 Minutes

```
numbers.gather(Gatherers.windowSliding(5))
       .forEach(System.out::println); // Output: [1,2,3,4,5], ..., [16,17,18,19,20]
```

- **Enhanced stream processing**: Enhances stream processing with more control and flexibility
- **Intermediate Transformations and Grouping**: Lets you transform and group stream elements **while streaming**, not just at the end like `collect()`
- **Lots of Flexibility**: Support for **windows**, **sliding views**, **parallel mapping**, and **custom accumulation**
  - Example functions:
    - **`fold()`** – Combines all stream values into one using custom logic (like a running total).
    - **`mapConcurrent()`** – Processes elements in parallel using a function, ideal for heavy/slow tasks.
    - **`windowFixed()`** – Splits the stream into equal-sized, non-overlapping groups.
    - **`windowSliding()`** – Creates overlapping groups for sliding window computations.

Gatherers.fold method

In 28 Minutes

```
var numbers = IntStream.rangeClosed(1, 20).boxed();

numbers.gather(Gatherers.fold(
    () -> 0,
    (sum, i) -> sum + i
)).forEach(System.out::println);  // Output: 210
```

- **What?**: Combines all elements in the stream into a single result (like adding numbers together).
- **How?**: You provide a starting value (e.g., `0`) and a rule to combine values (e.g., `sum + i`).
- **Usecase**: Great for tasks like summing, multiplying, or custom reductions.

Gatherers.mapConcurrent method


```
var numbers = IntStream.rangeClosed(1, 20).boxed();

numbers.gather(Gatherers.mapConcurrent(
    4,
    i -> i * i
)).forEach(System.out::println);  // Output: squares of 1–20 (unordered)
```

- **What?**: Applies a function (like squaring) to each element in parallel.
- **Control Maximum Concurrency**: You control how many operations can run at the same time (with `maxConcurrency`).
- **Usecase**: Useful for speeding up slow operations using concurrency (e.g., API calls or heavy calculations).

Gatherers.windowFixed method

In28 Minutes

```
var numbers = IntStream.rangeClosed(1, 20).boxed();

numbers.gather(Gatherers.windowFixed(5))
       .forEach(System.out::println);

// Output: [1,2,3,4,5], [6,7,8,9,10], ..., [16,17,18,19,20]
```

- **What?**: Groups the stream elements into **non-overlapping fixed-size lists**.
- **Example**:For example, if the size is 5, you'll get `[1,2,3,4,5]`, `[6,7,8,9,10]`, etc.
- **Usecase**: Helpful when processing data in equal-sized chunks or batches.

Gatherers.windowSliding method

In28 Minutes

```
var numbers = IntStream.rangeClosed(1, 20).boxed();

numbers.gather(Gatherers.windowSliding(5))
       .forEach(System.out::println);

// Output: [1,2,3,4,5], [2,3,4,5,6], ..., [16,17,18,19,20]
```

- **Create Sliding Windows**: Creates **overlapping lists** (sliding windows) of a given size.
- **Example**: With a window size of 5, you'll get `[1,2,3,4,5]`, `[2,3,4,5,6]`, `[3,4,5,6,7]`, etc.
- **Ideal Usecase**: Ideal for running calculations over moving windows, like averages or trends.

Getting Started with Spring Framework - Goals

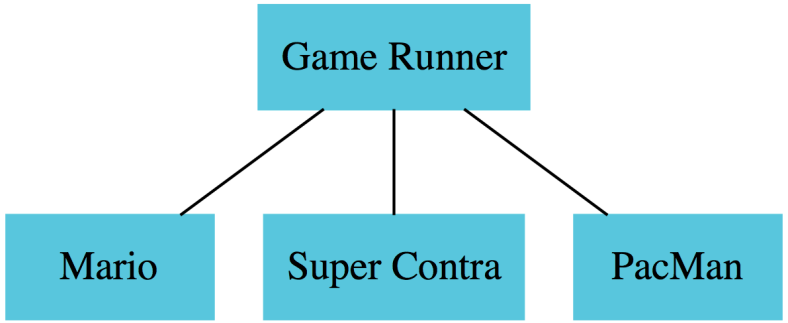
In 28 Minutes

- 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

In 28 Minutes

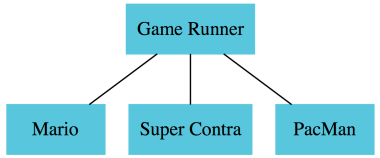
- Design Game Runner to run games:
  - Mario, Super Contra, PacMan etc
- **Iteration 1**: Tightly Coupled
  - GameRunner class
  - Game classes: Mario, Super Contra, PacMan etc
- **Iteration 2**: Loose Coupling - Interfaces
  - GameRunner class
  - GamingConsole interface
    - Game classes: Mario, Super 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

In28 Minutes

- **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?

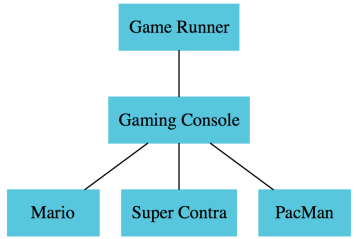
In28 Minutes

- 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

In28 Minutes

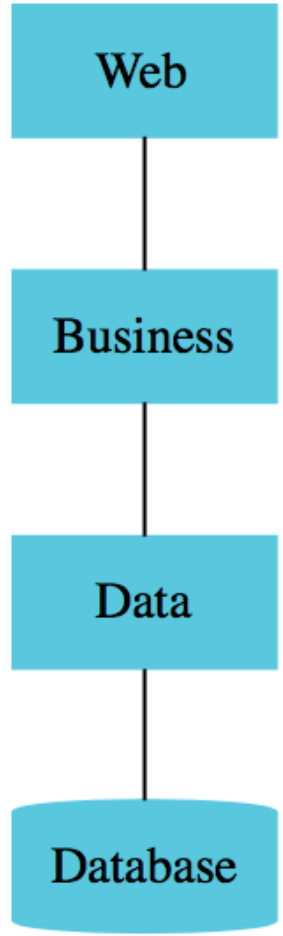
- **@Component** (..): Class managed by Spring framework
- **Dependency**: GameRunner needs GamingConsole impl!
  - GamingConsole Impl (Ex: MarioGame) is a dependency of GameRunner
- **Component Scan**: How does Spring Framework find component classes?
  - It scans packages! (`@ComponentScan("com.in28minutes")`)
- **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 28 Minutes

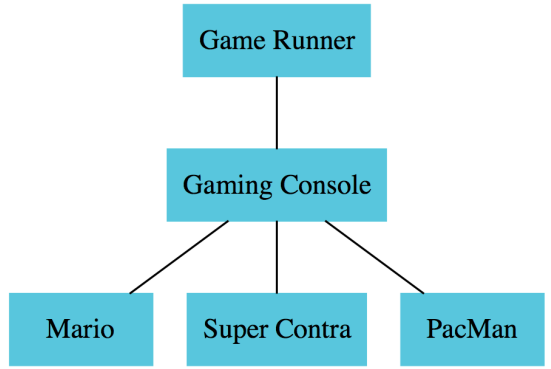
- In **Game Runner** Hello World App, we have very few classes
- BUT Real World applications **are much more complex**:
  - Multiple Layers (Web, Business, Data etc)
  - Each layer is **dependent** on the layer below it!
    - Example: Business Layer class talks to a Data Layer class
      - Data Layer 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?

In 28 Minutes

- 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)

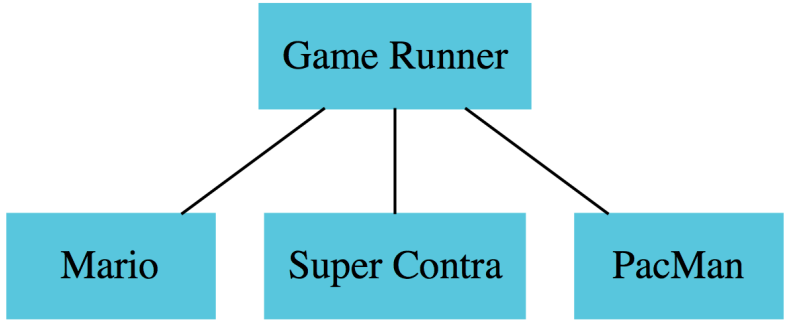
In 28 Minutes

- 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

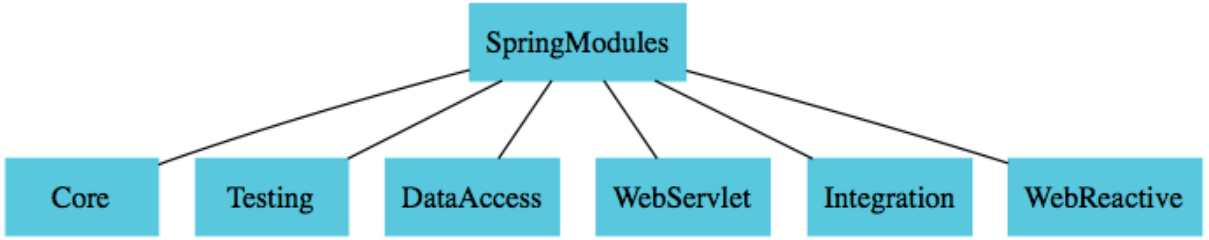
Exploring Spring - Dependency Injection Types

In 28 Minutes

- **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!

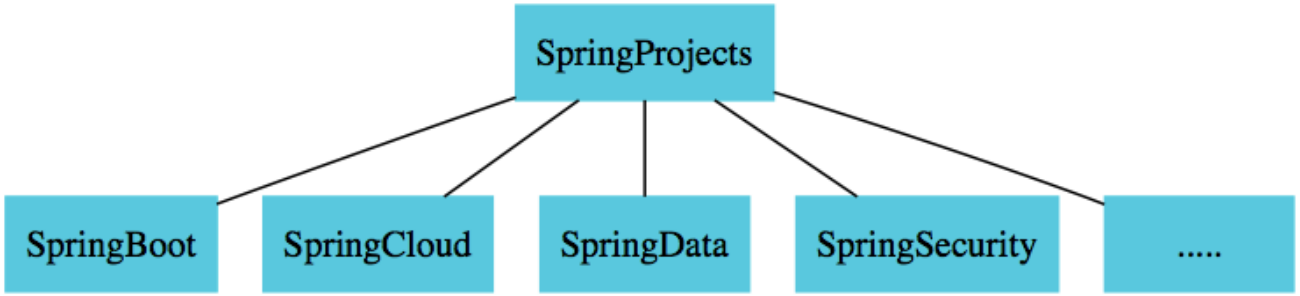


Spring Modules



- 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

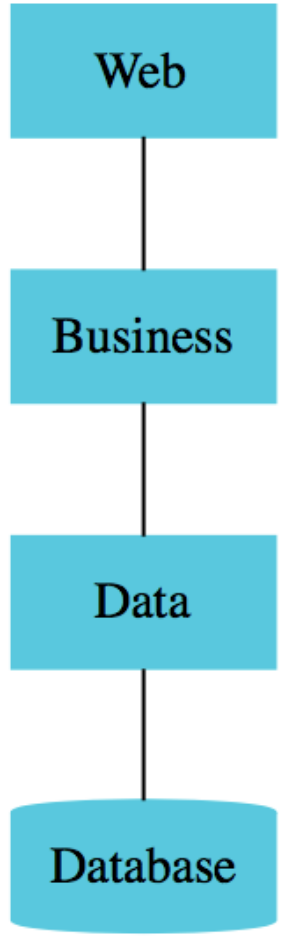


- Spring Projects: Spring keeps evolving (REST API > Microservices > Cloud)
  - **Spring Boot**: Most popular framework to build microservices
  - **Spring Cloud**: Build cloud native applications
  - **Spring Data**: Integrate the same way with different types of databases : NoSQL and Relational
  -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

Why is Spring Popular?

In28 Minutes

- **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

In28 Minutes

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

In28 Minutes

- **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

In28 Minutes

- 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

In28 Minutes

```
// 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!

In28 Minutes

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

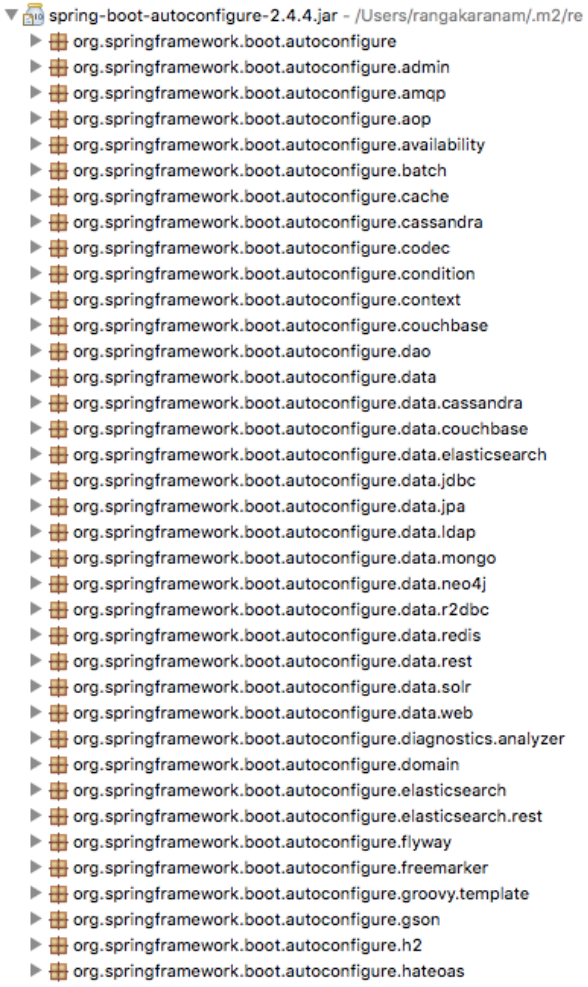
Spring Boot Starter Projects

- **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

In 28 Minutes

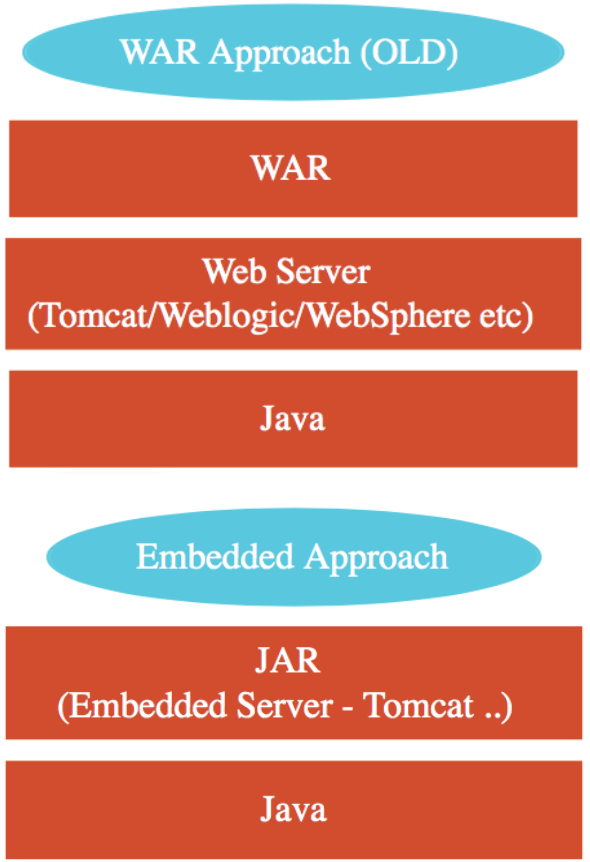
- Spring Boot provides **Auto Configuration**
  - **Basic configuration** to run your application using the frameworks defined in your maven dependencies
  - Auto Configuration is **decided based on**:
    - Which frameworks are in the Class Path?
    - What is the existing configuration (Annotations 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



Spring Boot Embedded Servers

In28 Minutes

- 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



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

In28 Minutes

- **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

In28 Minutes

- **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!

In28 Minutes

JUnit

In 5 Steps

Introduction to Unit Testing with JUnit

In 28 Minutes

- Large applications can have **1000s of code files** and **millions of lines of code**
- **Testing**: Check app behavior against expected behavior
  - **Option 1**: Deploy the complete application and test
    - This is called **System Testing** or **Integration Testing**
  - **Option 2**: Test specific units of application code independently
    - Examples: A specific method or group of methods
    - This is called **Unit Testing**
    - **Advantages of Unit Testing**
      - Finds bug early (run under CI)
      - Easy to fix bugs
      - Reduces costs in the long run
    - **Most Popular Java Frameworks**: JUnit and Mockito
  - **Recommended**: Option 1 + Option 2

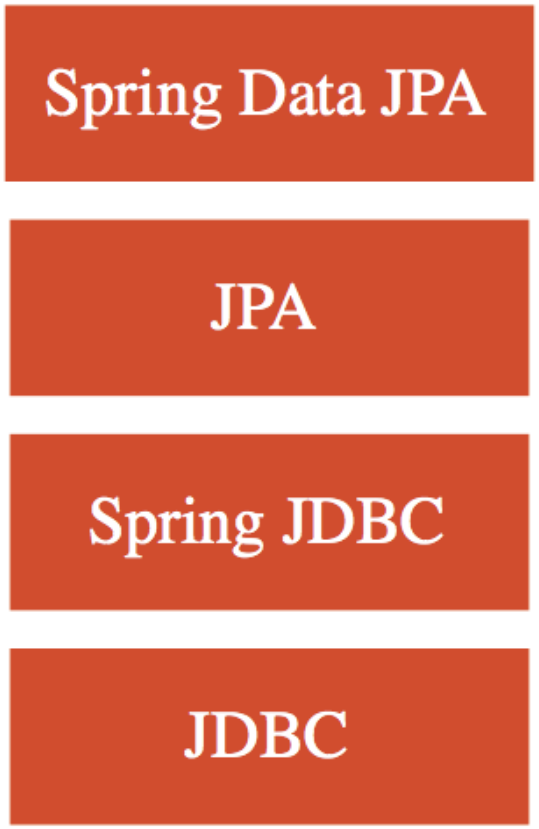
In28 Minutes

JPA and Hibernate in 10 Steps

Getting Started with JPA and Hibernate

In 28 Minutes

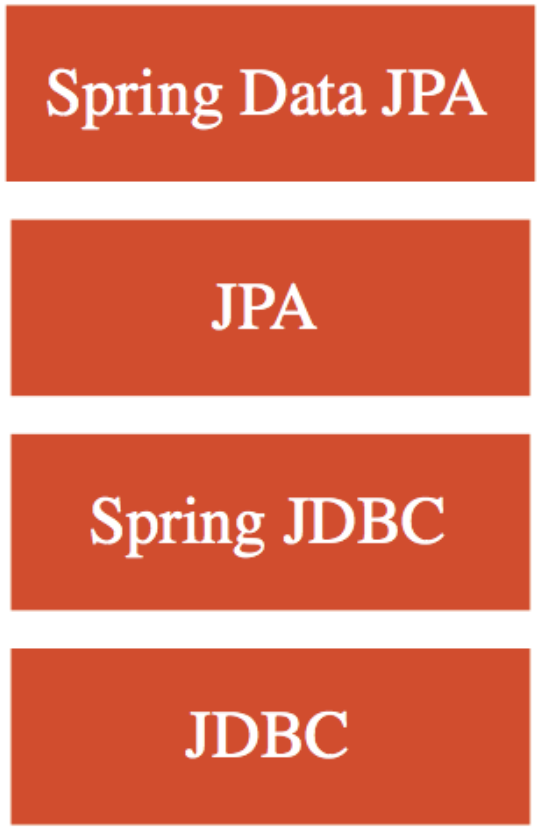
- Build a Simple JPA App using **Modern Spring Boot** Approach
- Get **Hands-on** with JPA, Hibernate and Spring Boot
  - World before JPA - JDBC, Spring JDBC
  - Why JPA? Why Hibernate? (JPA vs Hibernate)
  - Why Spring Boot and Spring Boot Data JPA?
  - **JPA** Terminology: Entity and Mapping



Learning JPA and Hibernate - Approach

In 28 Minutes

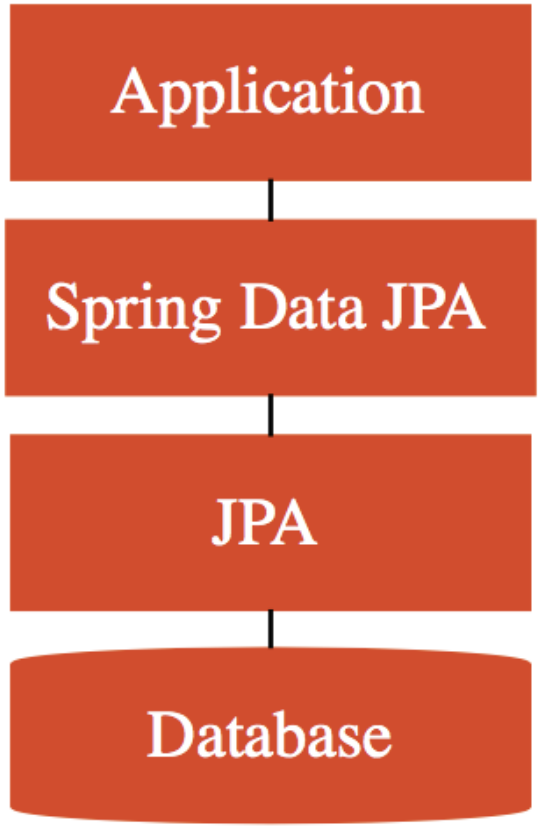
- 01: Create a **Spring Boot Project** with H2
- 02: Create **COURSE table**
- 03: Use **Spring JDBC** to play with COURSE table
- 04: Use **JPA and Hibernate** to play with COURSE table
- 05: Use **Spring Data JPA** to play with COURSE table



Spring Boot Auto Configuration Magic

In28 Minutes

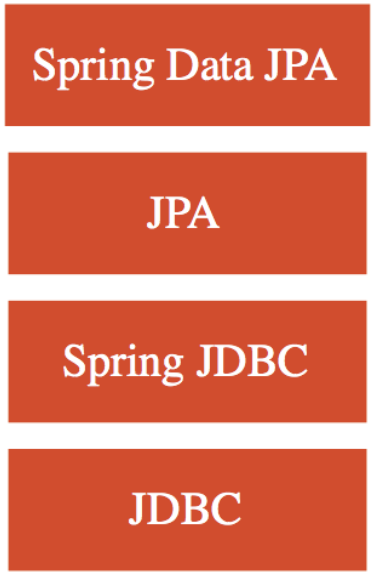
- We added Data JPA and H2 dependencies:
  - Spring Boot Auto Configuration does some magic:
    - Initialize JPA and Spring Data JPA frameworks
    - Launch an in memory database (H2)
    - Setup connection from App to in-memory database
    - Launch a few scripts at startup (example: `data.sql`, `schema.sql`)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



JDBC to Spring JDBC to JPA to Spring Data JPA

In 28 Minutes

- **JDBC**
  - Write a lot of SQL queries! (*delete from todo where id=?*)
  - And write a lot of Java code
- **Spring JDBC**
  - Write a lot of SQL queries (*delete from todo where id=?*)
  - 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

In28 Minutes

JDBC example

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

Spring JDBC example

```
public void deleteTodo(int id) {
    jdbcTemplate.update("delete from todo where id=?", id);
}
```

JPA Example

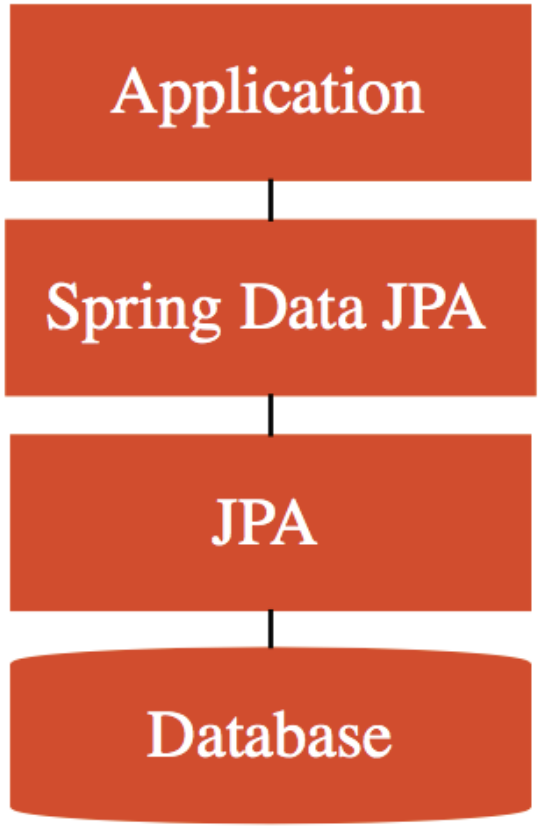
```
@Repository
public class PersonJpaRepository {

  @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>{
```

Hibernate vs JPA

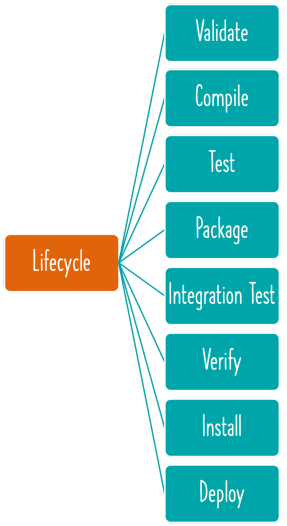
- **JPA** defines the specification. It is an API.
  - How do you define entities?
  - How do you map attributes?
  - Who manages the entities?
- Hibernate is one of the popular implementations of JPA
- Using Hibernate directly would result in a lock in to Hibernate
  - There are other JPA implementations (Toplink, for example)



Maven

What is Maven?

- **Things you do** when writing code each day:
  - Create new projects
  - Manages **dependencies** and their versions
    - Spring, Spring MVC, Hibernate,...
    - Add/modify dependencies
  - **Build** a JAR file
  - Run your application locally in Tomcat or Jetty or ..
  - Run **unit tests**
  - Deploy to a test environment
  - and a lot more..
- Maven helps you do all these and more...



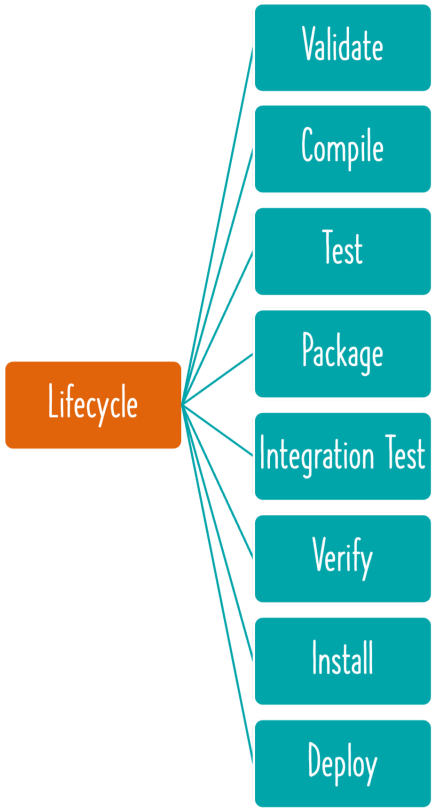
Exploring Project Object Model - pom.xml

In 28 Minutes

- Let's explore Project Object Model - pom.xml
  - **1: Maven dependencies**: Frameworks & libraries used in a project
    - Ex: `spring-boot-starter-web` and `spring-boot-starter-test`
    - Why are there so many dependencies in the classpath?
      - Answer: Transitive Dependencies
      - (REMEMBER) Spring dependencies are DIFFERENT
  - **2: Parent Pom**: spring-boot-starter-parent
    - Dependency Management: `spring-boot-dependencies`
    - Properties: `java.version`, plugins and configurations
  - **3: Name of our project**: groupId + artifactId
    - **1: groupId**: Similar to package name
    - **2: artifactId**: Similar to class name
    - **Why is it important?**
      - Think about this: How can other projects use our new project?
- **Activity**: help:effective-pom, dependency:tree & Eclipse UI
  - Let's add a new dependency: spring-boot-starter-web

Exploring Maven Build Life Cycle

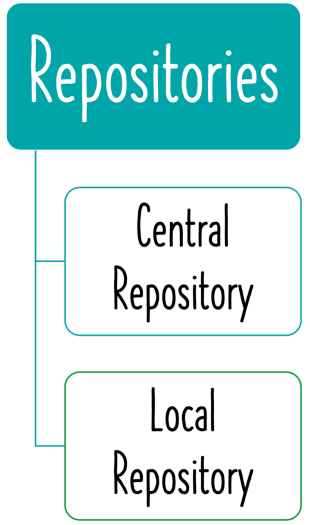
- When we run a maven command, maven build life cycle is used
- Build LifeCycle is a sequence of steps
  - Validate
  - Compile
  - Test
  - Package
  - Integration Test
  - Verify
  - Install
  - Deploy



How does Maven Work?

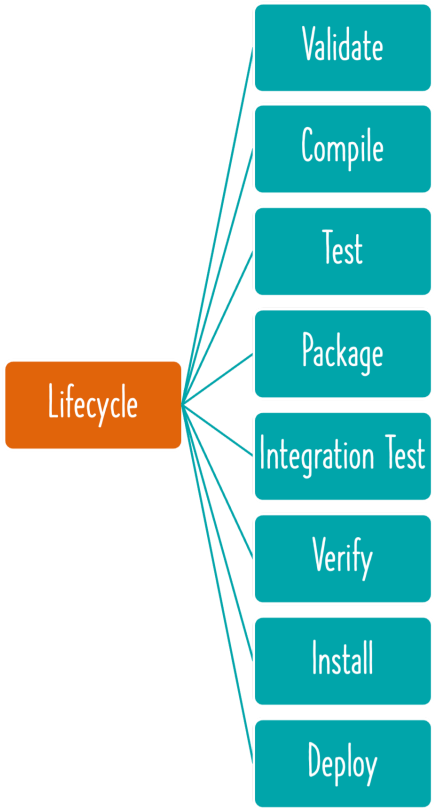
In28 Minutes

- Maven follows **Convention over Configuration**
  - Pre defined folder structure
  - Almost all Java projects follow **Maven structure** (Consistency)
- **Maven central repository** contains jars (and others) indexed by artifact id and group id
  - Stores all the versions of dependencies
  - repositories > repository
  - pluginRepositories > pluginRepository
- When a dependency is added to pom.xml, Maven tries to download the dependency
  - Downloaded dependencies are stored inside your maven local repository
  - **Local Repository** : a temp folder on your machine where maven stores the jar and dependency files that are downloaded from Maven Repository.



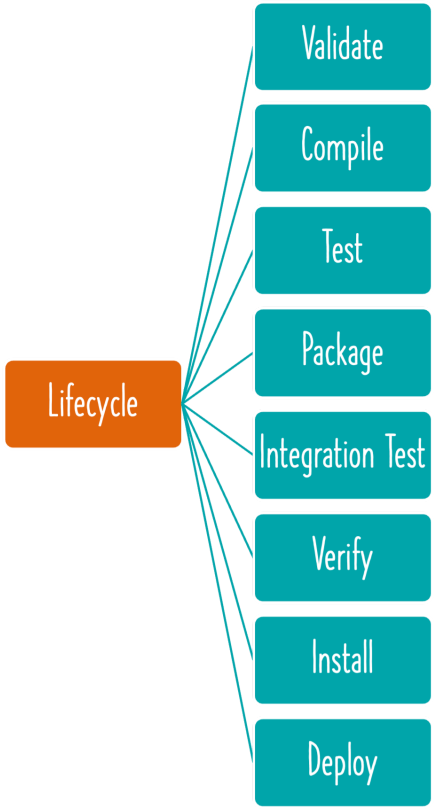
Important Maven Commands

- mvn --version
- mvn compile: Compile source files
- mvn test-compile: Compile test files
  - OBSERVCE CAREFULLY: This will also compile source files
- mvn clean: Delete target directory
- mvn test: Run unit tests
- mvn package: Create a jar
- mvn help:effective-pom
- mvn dependency:tree



Spring Boot Maven Plugin

- **Spring Boot Maven Plugin**: Provides Spring Boot support in Apache Maven
  - Example: Create executable jar package
  - Example: Run Spring Boot application
  - Example: Create a Container Image
  - **Commands**:
    - mvn spring-boot:**repackage** (create jar or war)
      - Run package using `java -jar`
    - mvn spring-boot:**run** (Run application)
    - mvn spring-boot:**start** (Non-blocking. Use it to run integration tests.)
    - mvn spring-boot:**stop** (Stop application started with start command)
    - mvn spring-boot:**build-image** (Build a container image)



How are Spring Releases Versioned?

In28 Minutes

- **Version scheme** - MAJOR.MINOR.PATCH[-MODIFIER]
  - **MAJOR**: Significant amount of work to upgrade (10.0.0 to 11.0.0)
  - **MINOR**: Little to no work to upgrade (10.1.0 to 10.2.0)
  - **PATCH**: No work to upgrade (10.5.4 to 10.5.5)
  - **MODIFIER**: Optional modifier
    - **Milestones** - M1, M2, .. (10.3.0-M1,10.3.0-M2)
    - **Release candidates** - RC1, RC2, .. (10.3.0-RC1, 10.3.0-RC2)
    - **Snapshots** - SNAPSHOT
    - **Release** - Modifier will be ABSENT (10.0.0, 10.1.0)
- **Example versions in order:**
  - 10.0.0-SNAPSHOT, 10.0.0-M1, 10.0.0-M2, 10.0.0-RC1, 10.0.0-RC2, 10.0.0, ...
- **MY RECOMMENDATIONS:**
  - Avoid SNAPSHOTs
  - Use ONLY Released versions in PRODUCTION

REST API

In28 Minutes

- **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

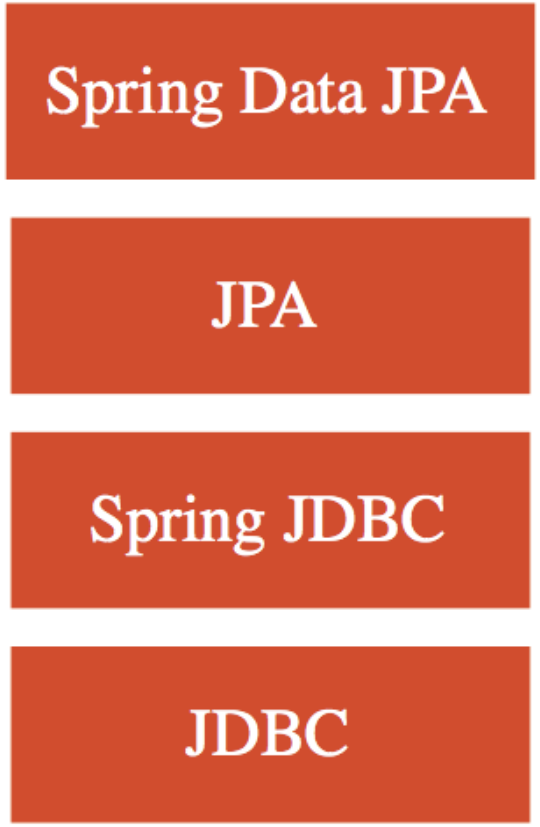
In28 Minutes

- 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

In28 Minutes

- **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

In28 Minutes

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) {........
```

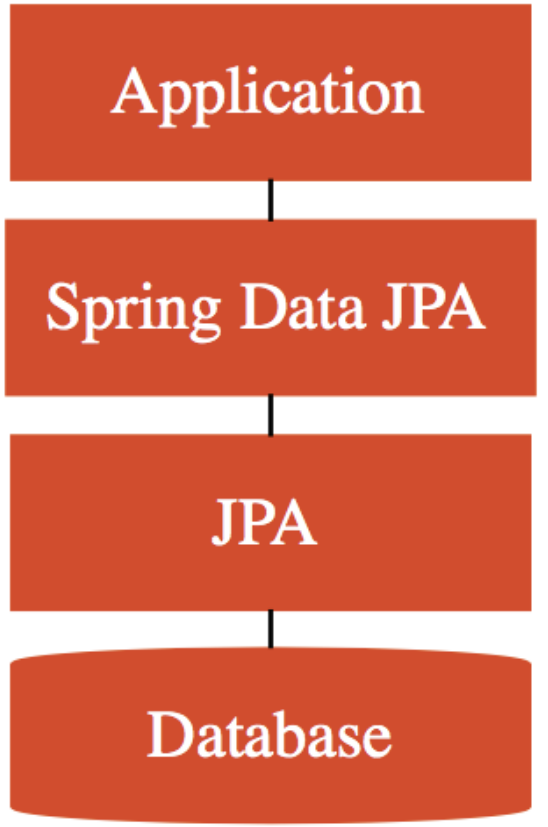
Spring Data JPA Example

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

Spring Boot Auto Configuration Magic - Data JPA

In28 Minutes

- 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

In 28 Minutes

- Java keeps improving:
  - Java 10, Java 11, Java 12, ...
- Java Project - REST API in Modern Approach:
  - Spring
  - Spring Boot
- Do NOT forget to leave a Review!

What's Next? - Don't Stop Learning!

In28 Minutes

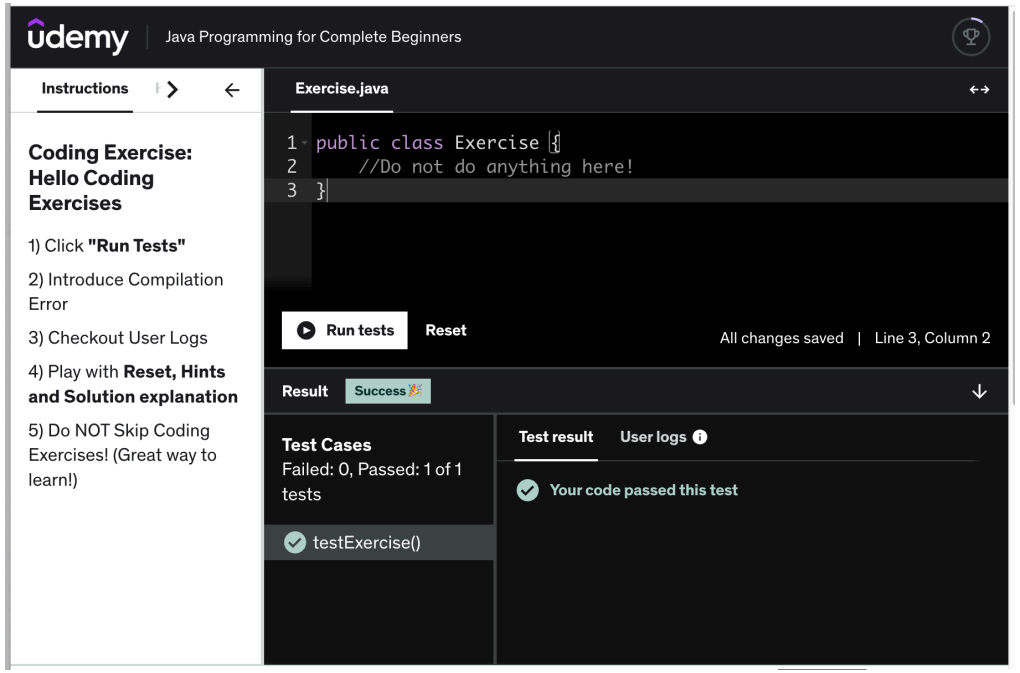
https://github.com/in28minutes/learn

- **Step I**: Build more applications:
  - REST API and Microservices
  - Full Stack Applications (Angular and React)
  - Mobile Applications
  - Learn Unit Testing (JUnit and Mockito) and Clean Code
- **Step II**: Learn Java Frameworks in Depth:
  - Spring & Spring Boot
  - Hibernate and JPA
- **Step III**: Go Cloud (AWS, Azure and Google Cloud)
- **Step IV**: Learn DevOps

NEW FEATURE: Coding exercises without IDE installation

In 28 Minutes

- Hurrah! You can do coding exercises directly on Udemy
  - Without needing an IDE!
- We are adding in a lot of exercises!
- Each coding exercise has:
  - Instructions (or problem statement)
  - Hints
  - Solution Explanation
    - Solution video (watch me solve the exercise!)



NEW FEATURE: Coding exercises without IDE installation - 2

In 28 Minutes

- Next exercise will help you get familiar with Udemy Coding Exercises IDE
  - Open it in a new window and practice while you are watching this video!
- Advantages:
  - Your solution is automatically checked
  - You get additional practice
  - Additional skills you'll improve:
    - Reading
    - Documentation

78. Introduction To Java Coding Exercises

1min

Coding Exercise 1: Coding Exercise: Hello Coding Exercises

Coding Exercise 2: Coding Exercise: Print Hello World

79. Solution Video For Coding Exercise: Print Hello World

1min

Coding Exercise 3: Coding Exercise: Time Converter

80. Solution Video For Coding Exercise: Time Converter

1min

Coding Exercise 4: Coding Exercise: Exam Result Checker

81. Solution Video For Coding Exercise: Exam Result Checker

1min

Coding Exercise 5: Coding Exercise: Is Valid Triangle

82. Solution Video For Coding Exercise: Is Valid Triangle

My 10 Rules for Happy Programmers

In 28 Minutes

- **Embrace the challenge**: Each problem is an opportunity to learn
- **It's okay to fail**: Failure is a part of the learning process
- **Practice makes perfect**: The more you code, the better you'll get
- **Be patient**: Learning to code takes time and effort
- **Have fun**: Coding can be a lot of fun, enjoy the process
- **Don't give up.**: If you're struggling, keep at it
- **Break it down**: Break a complex problem into smaller parts
- **Be persistent.**: Don't give up on a problem just because it's difficult
- **Celebrate progress**: Acknowledge your achievements, no matter how small
- **Stay curious**: Keep exploring new technologies, programming languages, and concepts

What Next?

FASTEST ROADMAPS

in28minutes.com

In28 Minutes

- Google Cloud Certifications
- Azure Certifications
- AWS Certifications
- DevOps
- Java Full Stack
- Java Microservices