In28 Minutes

Spring Framework in 10 Steps

Getting Started with Spring Framework - Goals

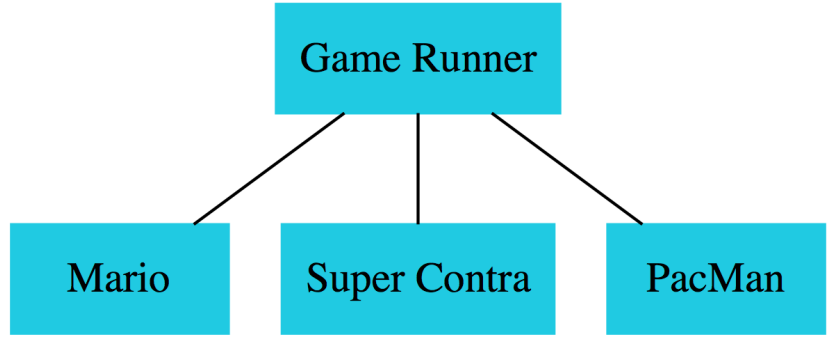
In28 Minutes

- 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

In28 Minutes

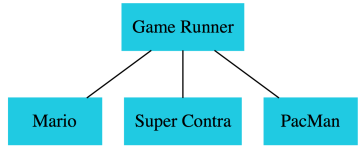
- Design Game Runner to run games:
  - Mario, Super Contra, PacMan etc
- **Iteration 1**: Tightly Coupled
  - GameRunner class
  - Game classes: Mario, Super Contra, PacMan etc
- **Iteration 2**: Loose Coupling - Interfaces
  - GameRunner class
  - GamingConsole interface
    - Game classes: Mario, Super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

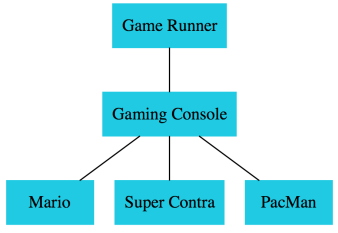
In28 Minutes

- 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

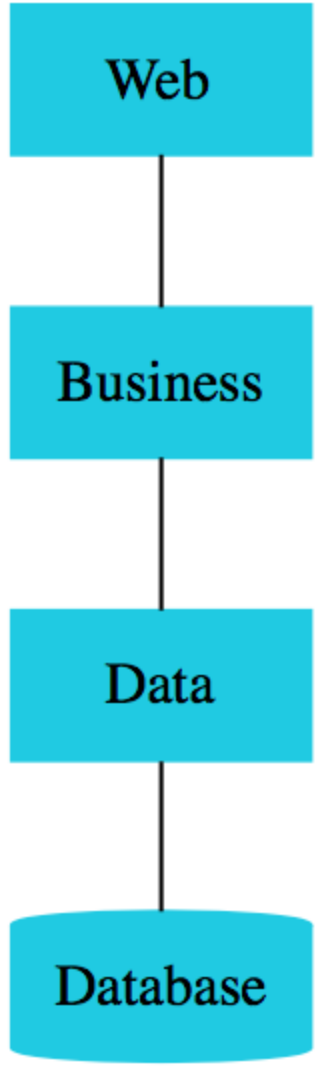
In28 Minutes

- **@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?

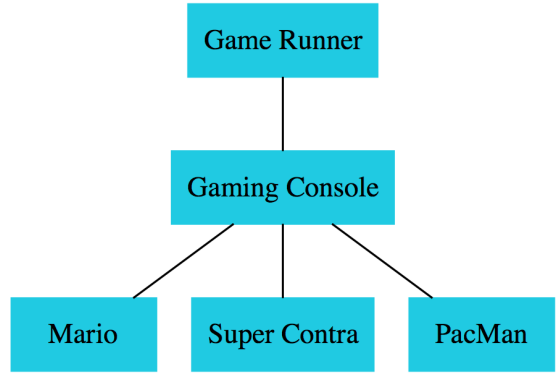
- In **Game Runner** Hello World App, we have very few classes
- BUT Real World applications **are much more complex**:
  - Multiple Layers (Web, Business, Data etc)
  - Each layer is **dependent** on the layer below it!
    - Example: Business Layer class talks to a Data Layer class
      - Data Layer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

In28 Minutes

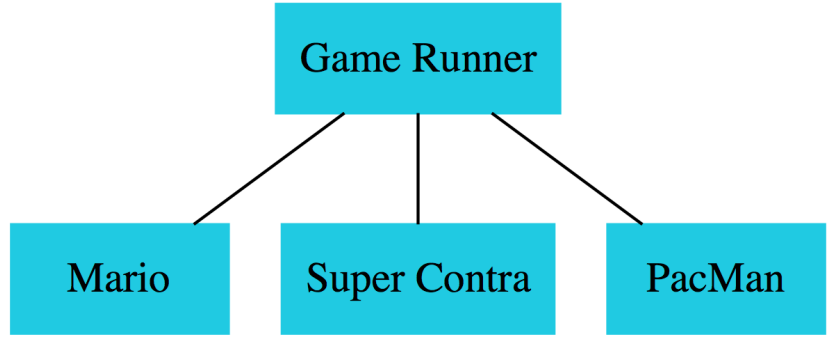
Maven™

- 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

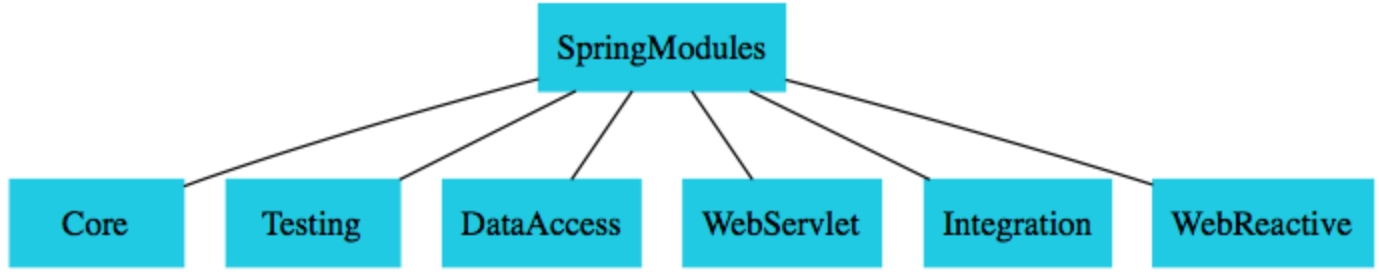
In 28 Minutes

- **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!



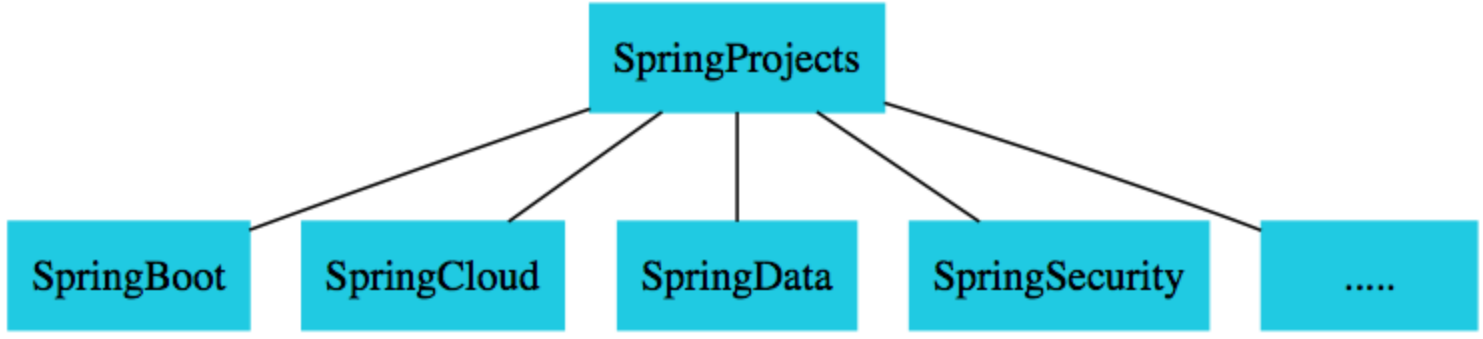
Spring Modules

In28 Minutes



- 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

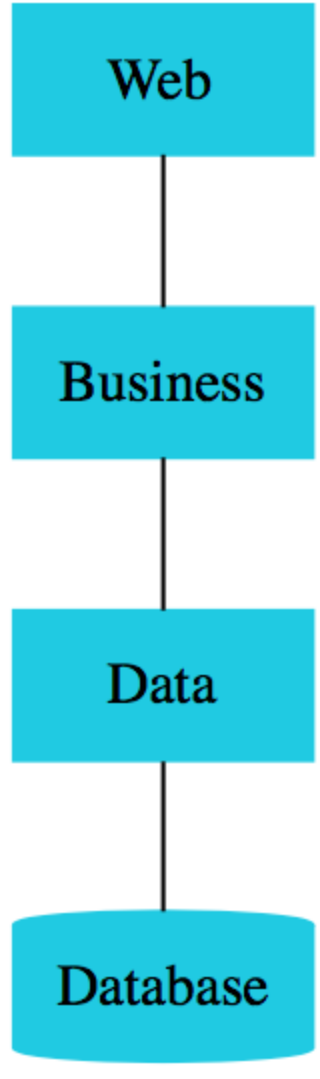


- Spring Projects: Spring keeps evolving (REST API > Microservices > Cloud)
  - **Spring Boot**: Most popular framework to build microservices
  - **Spring Cloud**: Build cloud native applications
  - **Spring Data**: Integrate the same way with different types of databases : NoSQL and Relational
  - **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 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!