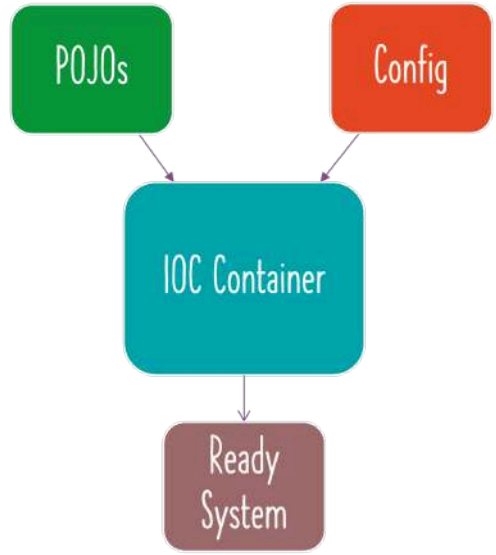
# What is Spring Container?

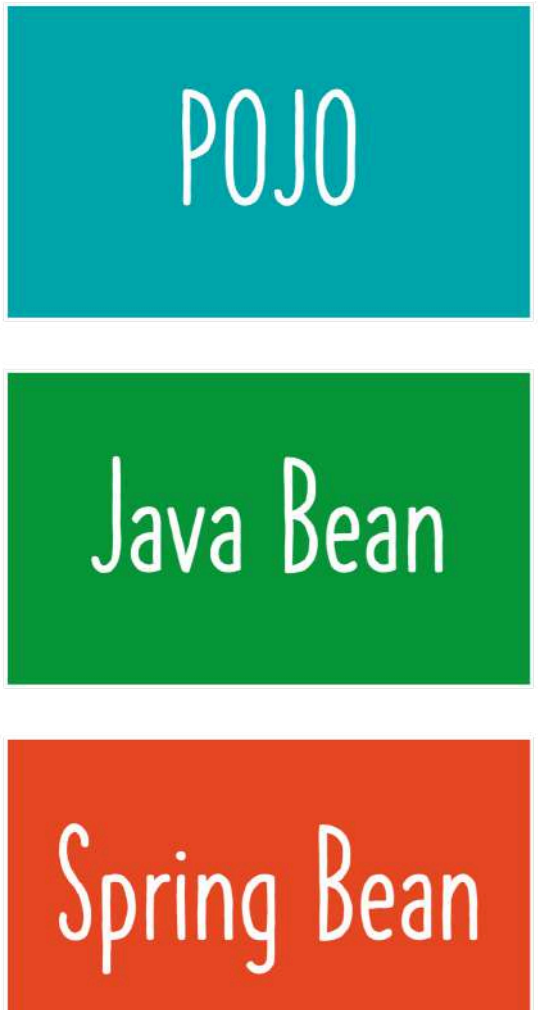
- **Spring Container**: Manages Spring beans & their lifecycle
- **1: Bean Factory**: Basic Spring Container
- **2: Application Context**: Advanced Spring Container with enterprise-specific features
  - Easy to use in web applications
  - Easy internationalization
  - Easy integration with Spring AOP
- **Which one to use?**: Most enterprise applications use Application Context
  - Recommended for web applications, web services - REST API and microservices



# Exploring Java Bean vs POJO vs Spring Bean

In28 Minutes

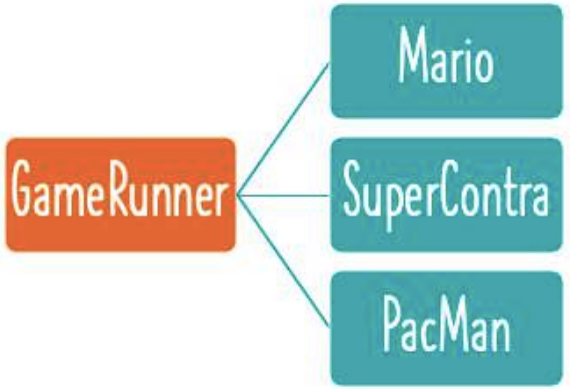
- **Java Bean**: Classes adhering to 3 constraints:
  - 1: Have public default (no argument) constructors
  - 2: Allow access to their properties using getter and setter methods
  - 3: Implement java.io.Serializable
- **POJO**: Plain Old Java Object
  - No constraints
  - Any Java Object is a POJO!
- **Spring Bean**: Any Java object that is managed by Spring
  - Spring uses IOC Container (Bean Factory or Application Context) to manage these objects



# Exploring Spring - Dependency Injection Types

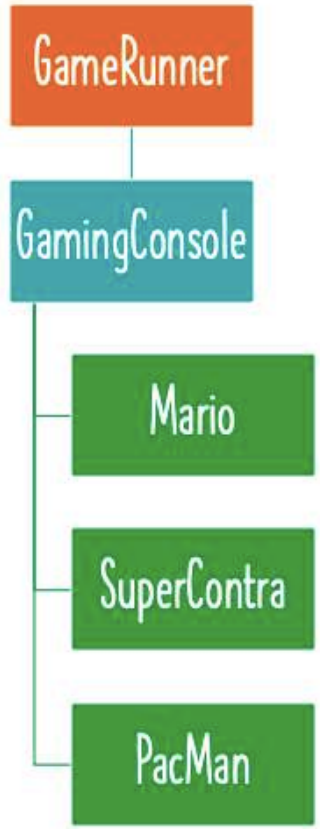
In 28 Minutes

- **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.
- Question: **Which one should you use?**
  - Spring team recommends Constructor-based injection as dependencies are automatically set when an object is created!



# Exploring auto-wiring in depth

- When a dependency needs to be @Autowired, IOC container looks for matches/candidates (by name and/or type)
  - **1:** If no match is found
    - **Result:** Exception is thrown
    - You need to help Spring Framework find a match
      - Typical problems:
        - @Component (or ..) missing
        - Class not in component scan
  - **2:** One match is found
    - **Result:** Autowiring is successful
  - **3:** Multiple candidates
    - **Result:** Exception is thrown
    - You need to help Spring Framework choose between the candidates
      - 1: Mark one of them as @Primary
        - If only one of the candidates is marked @Primary, it becomes the auto-wired value
      - 2: Use @Qualifier - Example: @Qualifier("myQualifierName")
        - Provides more specific control
        - Can be used on a class, member variables and method parameters



# @Primary vs @Qualifier - Which one to use?

In 28 Minutes

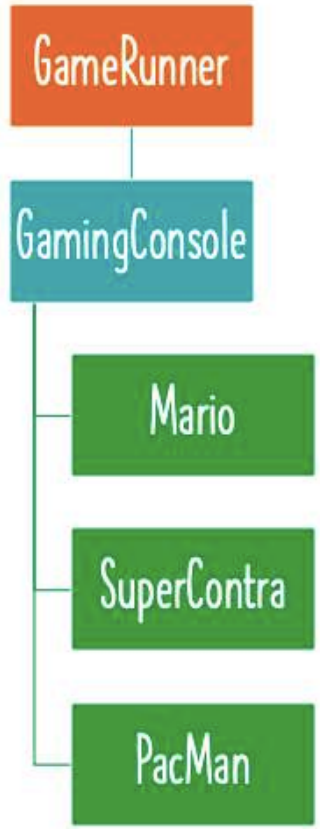
```
@Component @Primary
class QuickSort implement SortingAlgorithm {}

@Component
class BubbleSort implement SortingAlgorithm {}

@Component @Qualifier("RadixSortQualifier")
class RadixSort implement SortingAlgorithm {}

@Component
class ComplexAlgorithm
    @Autowired
    private SortingAlgorithm algorithm;

@Component
class AnotherComplexAlgorithm
    @Autowired @Qualifier("RadixSortQualifier")
    private SortingAlgorithm iWantToUseRadixSortOnly;
```

- **@Primary** - A bean should be given preference when multiple candidates are qualified
- **@Qualifier** - A specific bean should be auto-wired (name of the bean can be used as qualifier)
- **ALWAYS** think from the perspective of the class using the SortingAlgorithm:
  - **1: Just @Autowired**: Give me (preferred) SortingAlgorithm
  - **2: @Autowired + @Qualifier**: I only want to use specific SortingAlgorithm - RadixSort
  - (REMEMBER) @Qualifier has higher priority then @Primary

# Spring Framework - Important Terminology

In28 Minutes

- **@Component** (..): An instance of class will be 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



# @Component vs @Bean

In 28 Minutes

| Heading | @Component | @Bean |
|---|---|---|
| **Where?** | Can be used on any Java class | Typically used on methods in Spring Configuration classes |
| **Ease of use** | Very easy. Just add an annotation. | You write all the code. |
| **Autowiring** | Yes - Field, Setter or Constructor Injection | Yes - method call or method parameters |
| **Who creates beans?** | Spring Framework | You write bean creation code |
| **Recommended For** | Instantiating Beans for Your Own Application Code: @Component | 1:Custom Business Logic<br>2: Instantiating Beans for 3rd-party libraries: @Bean |
| **Beans per class?** | One (Singleton) or Many (Prototype) | One or Many - You can create as many as you want |

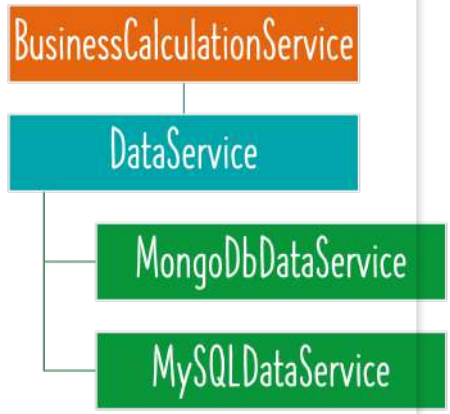
# Why do we have a lot of Dependencies?

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: BusinessCalculationService

# Exercise - BusinessCalculationService

```
public interface DataService
    int[] retrieveData();

public class MongoDbDataService implements DataService
    public int[] retrieveData()
        return new int[] { 11, 22, 33, 44, 55 };

public class MySQLDataService implements DataService
    public int[] retrieveData()
        return new int[] { 1, 2, 3, 4, 5 };

public class BusinessCalculationService
    public int findMax()
        return Arrays.stream(dataService.retrieveData())
                                        .max().orElse(0);
```




- Create **classes and interfaces as needed**
  - Use constructor injection to inject dependencies
  - Make `MongoDbDataService` as primary
  - Create a **Spring Context**
    - Prefer annotations
    - Retrieve `BusinessCalculationService` bean and run `findMax` method

# Exploring Lazy Initialization of Spring Beans




- Default initialization for Spring Beans: **Eager**
- Eager initialization is recommended:
  - Errors in the configuration are discovered immediately at application startup
- However, you can configure beans to be lazily initialized using **Lazy** annotation:
  - NOT recommended (AND) Not frequently used
- **Lazy** annotation:
  - Can be used almost everywhere @Component and @Bean are used
  - Lazy-resolution proxy will be injected instead of actual dependency
  - Can be used on Configuration (@Configuration) class:
    - All `@Bean` methods within the `@Configuration` will be lazily initialized

# Comparing Lazy Initialization vs Eager Initialization

In28 Minutes

| Heading | Lazy Initialization | Eager Initialization |
|---|---|---|
| **Initialization time** | Bean initialized when it is first made use of in the application | Bean initialized at startup of the application |
| **Default** | NOT Default | Default |
| **Code Snippet** | @Lazy OR @Lazy(value=true) | @Lazy(value=false) OR (Absence of @Lazy) |
| **What happens if there are errors in initializing?** | Errors will result in runtime exceptions | Errors will prevent application from starting up |
| **Usage** | Rarely used | Very frequently used |
| **Memory Consumption** | Less (until bean is initialized) | All beans are initialized at startup |
| **Recommended Scenario** | Beans very rarely used in your app | Most of your beans |