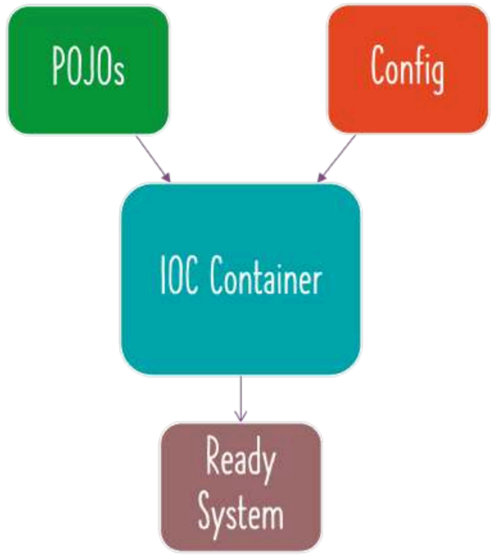
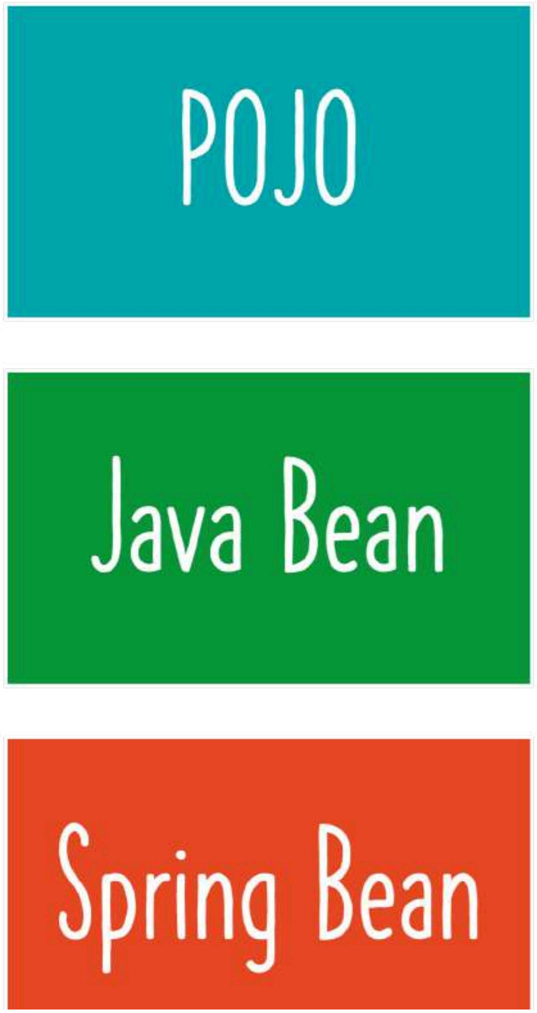
## What is Spring Container?

- **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



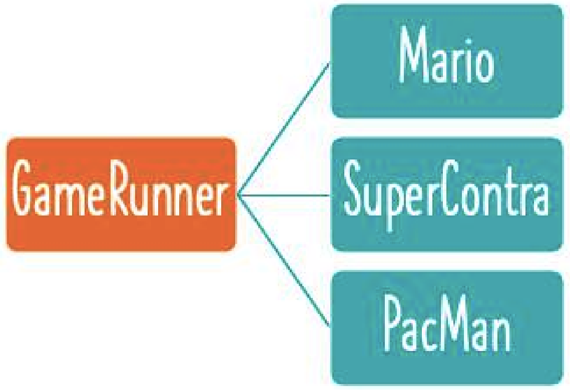
## Exploring Java Bean vs POJO vs Spring Bean

- **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



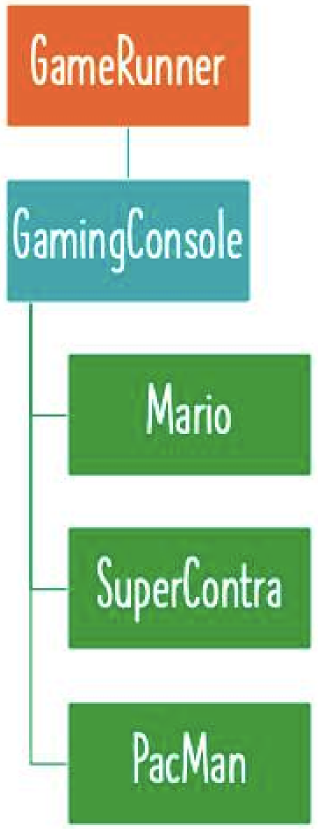
# Exploring Spring - Dependency Injection Types

- **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?

```
@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