
# IOC CONTAINER (APPLICATIONCONTEXT)

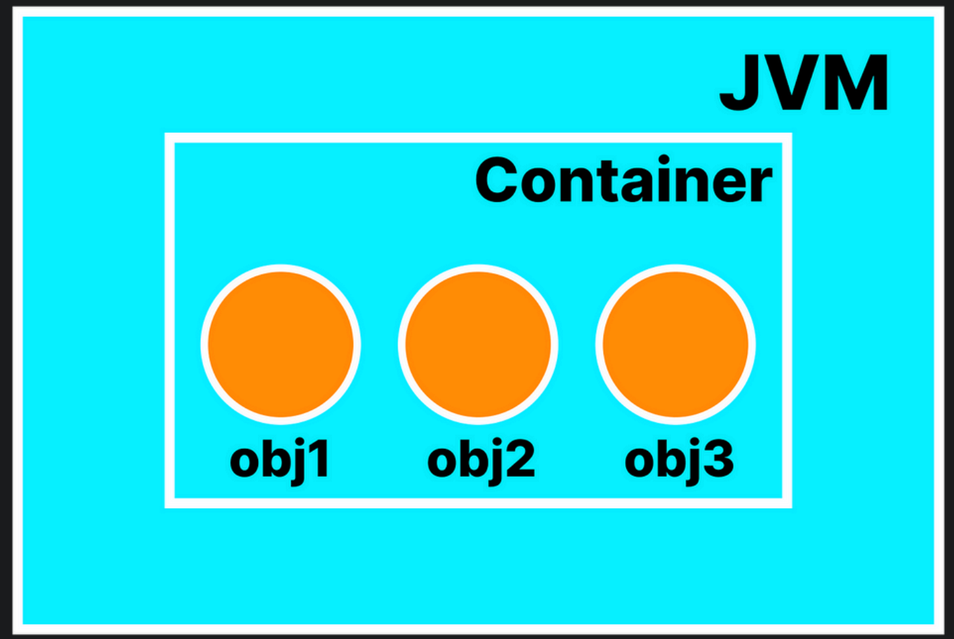
@med Mohammed Ezzaim

Imaginez que Spring Boot est comme une usine intelligente qui fabrique et gère des objets à votre place.

- **JVM** → L'usine elle-même (le moteur qui fait tourner Spring Boot).

- **Container (IoC Container)** → Le chef d'usine qui :
  - Crée les objets (appelés beans).
  - Assemble les pièces (injection de dépendances).
  - Stocke tout dans un entrepôt (contexte Spring).

- **obj1, obj2, obj3** → Les outils/services dont votre application a besoin (ex: une base de données, un service de calcul, un contrôleur web).



# IoC Container de Spring :

En Spring, le IoC Container est le cœur qui gère les objets (beans). Il existe sous deux formes principales :

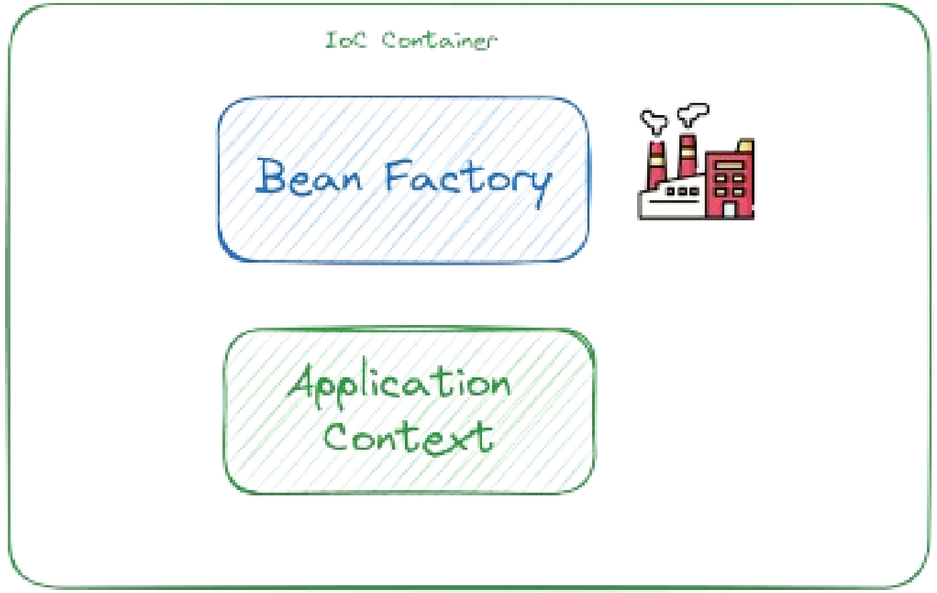
### 1. `BeanFactory` (Usine de base)

- **Fonctionnalités minimales** :
  - Crée et fournit des beans.
  - Gère les dépendances simples.
- **Exemple** :

java Copy Download

```java
BeanFactory factory = new XmlBeanFactory(new ClassPathResource("beans.xml"));
MonService service = factory.getBean(MonService.class);
```

- **Quand l'utiliser ?**
  - Pour des applications légères (rarement utilisé directement).



# IoC Container de Spring :

En Spring, le IoC Container est le cœur qui gère les objets (beans). Il existe sous deux formes principales :

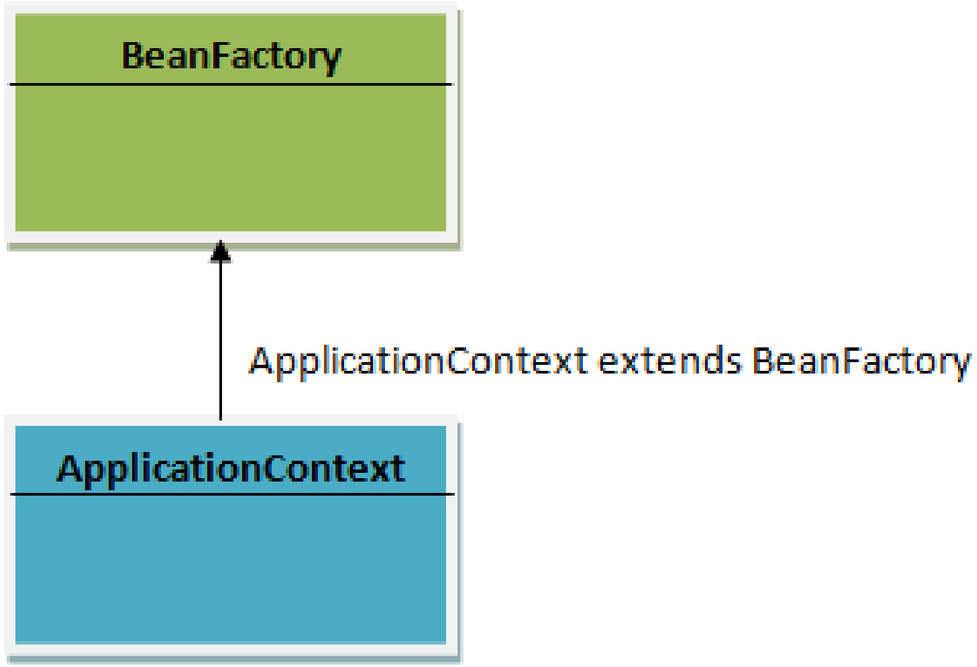
1. `BeanFactory` (Usine de base)

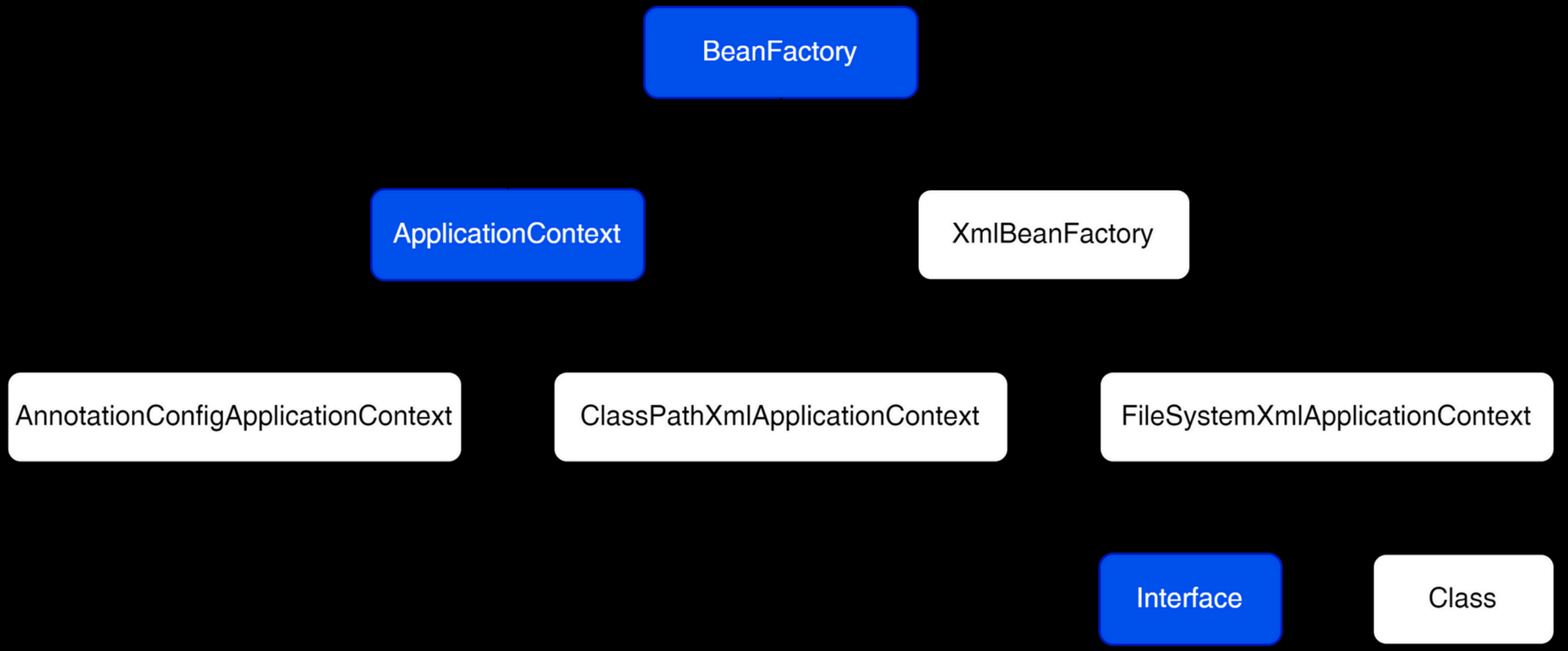
- **Fonctionnalités minimales** :
  - Crée et fournit des beans.
  - Gère les dépendances simples.
- **Exemple** :

java Copy Download

```java
BeanFactory factory = new XmlBeanFactory(new ClassPathResource("beans.xml"));
MonService service = factory.getBean(MonService.class);
```

- **Quand l'utiliser ?**
  - Pour des applications légères (rarement utilisé directement).




BeanFactory
ApplicationContext
XmlBeanFactory
AnnotationConfigApplicationContext
ClassPathXmlApplicationContext
FileSystemXmlApplicationContext
Interface
Class

# IoC Container de Spring :

### 2. `ApplicationContext` (Usine avancée – la plus utilisée)

**Extensions de `BeanFactory` avec des super-pouvoirs :**

- ✅ **Gestion des événements** (ex: `@EventListener`).
- ✅ **Internationalisation** (traduction des messages).
- ✅ **Accès aux ressources** (fichiers, configurations).
- ✅ **Intégration AOP** (programmation orientée aspect).
- ✅ **Chargement automatique dans Spring Boot.**

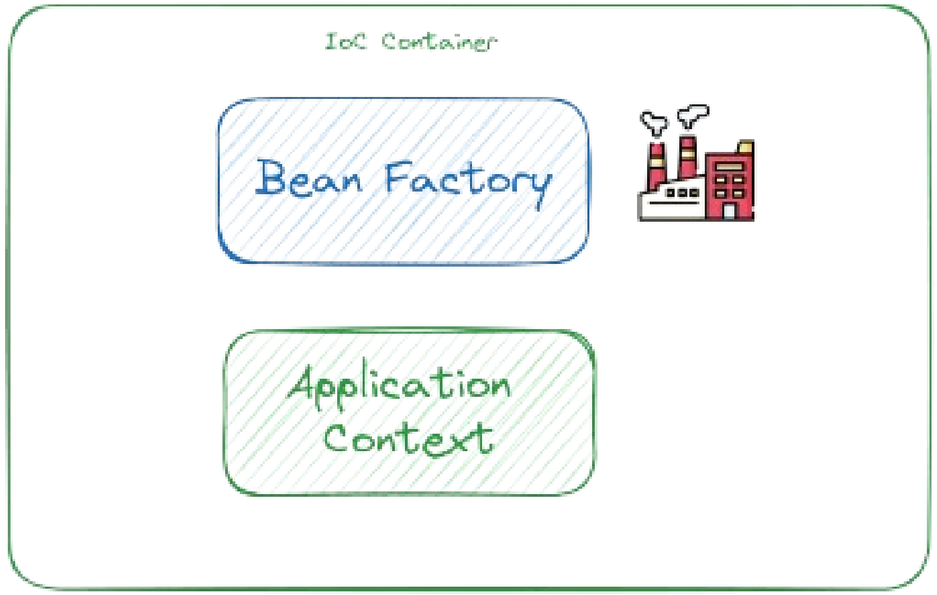
**Exemples d'implémentations :**

- `AnnotationConfigApplicationContext` (configuration par annotations).
- `ClassPathXmlApplicationContext` (configuration par XML).
- **Spring Boot** utilise `SpringApplication.run()`, qui démarre un `ApplicationContext`.

**Exemple :**

java Copy Download

```java
ApplicationContext context = SpringApplication.run(MaApplication.class, args);
MonService service = context.getBean(MonService.class); // Récupère un bean
```



# IoC Container de Spring :

### Le IoC Container est-il un objet de type `ApplicationContext` ?

**Réponse courte :**

✅ **Oui**, dans les applications Spring modernes (notamment avec **Spring Boot**), le **IoC Container** est *concrètement* représenté par un objet de type `ApplicationContext`.

### Différence Clé

| Fonctionnalité | `BeanFactory` | `ApplicationContext` |
|---|---|---|
| Chargement des beans | **Paresseux** (crée quand on demande) | **Eager** (crée au démarrage) |
| Fonctions avancées | ❌ Non | ✅ Oui (events, AOP, i18n...) |
| Utilisation typique | Rare | **Toujours** (Spring Boot) |

# IoC Container de Spring :

### Analogie

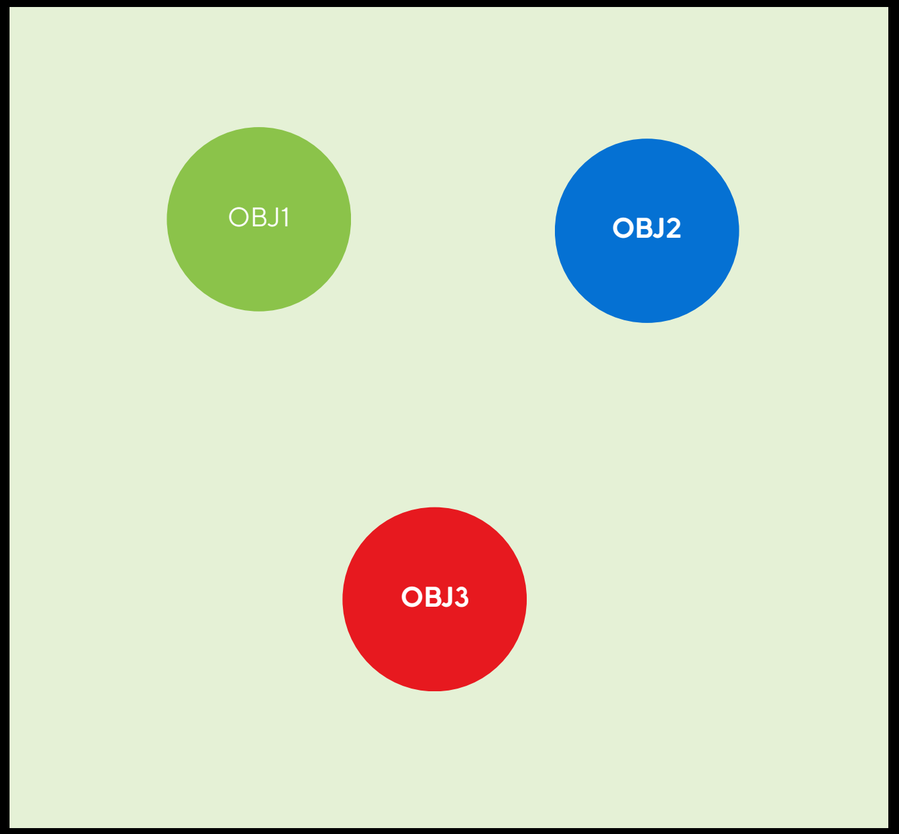
- `BeanFactory` → **Une cafetière basique** (fait juste du café).
- `ApplicationContext` → **Une machine à café premium** (café + mousse de lait, timer, nettoyage auto...).

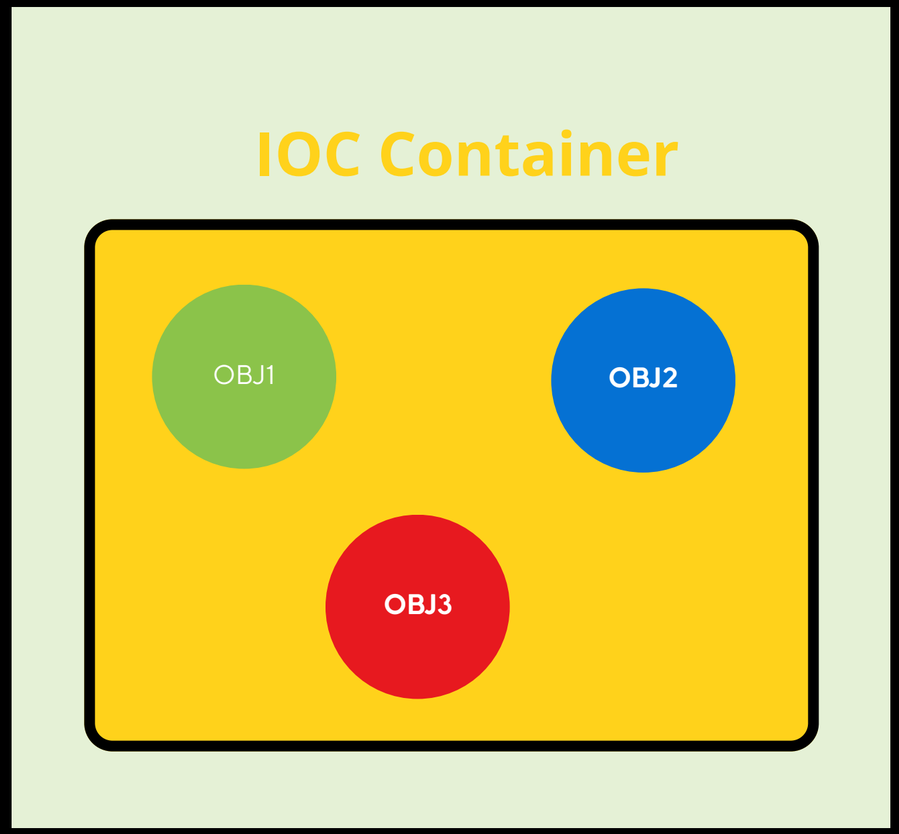
---

### En Résumé

- **Spring Boot utilise toujours** `ApplicationContext` (car plus complet).
- `BeanFactory` est un socle technique, mais peu utilisé seul.
- Les **beans** ( `obj1` , `obj2` , `obj3` ) sont gérés par le container, que ce soit en `BeanFactory` ou `ApplicationContext` .

👉 **Pour 99% des cas, vous utilisez** `ApplicationContext` **sans même y penser (merci Spring Boot !).**

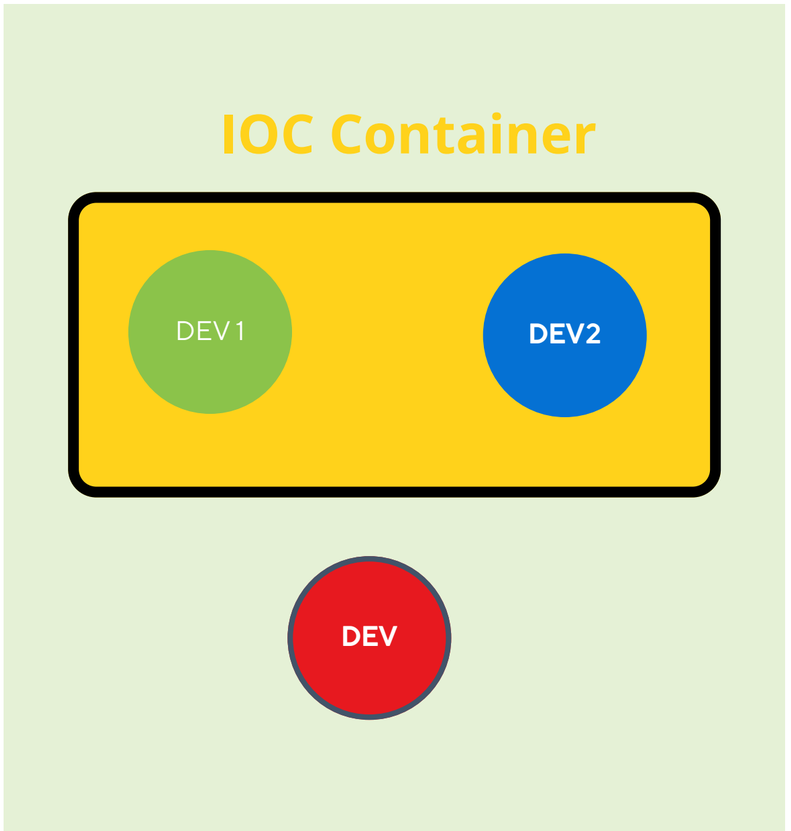

JVM
OBJ1
OBJ2
OBJ3


JVM
IOC Container
OBJ1
OBJ2
OBJ3

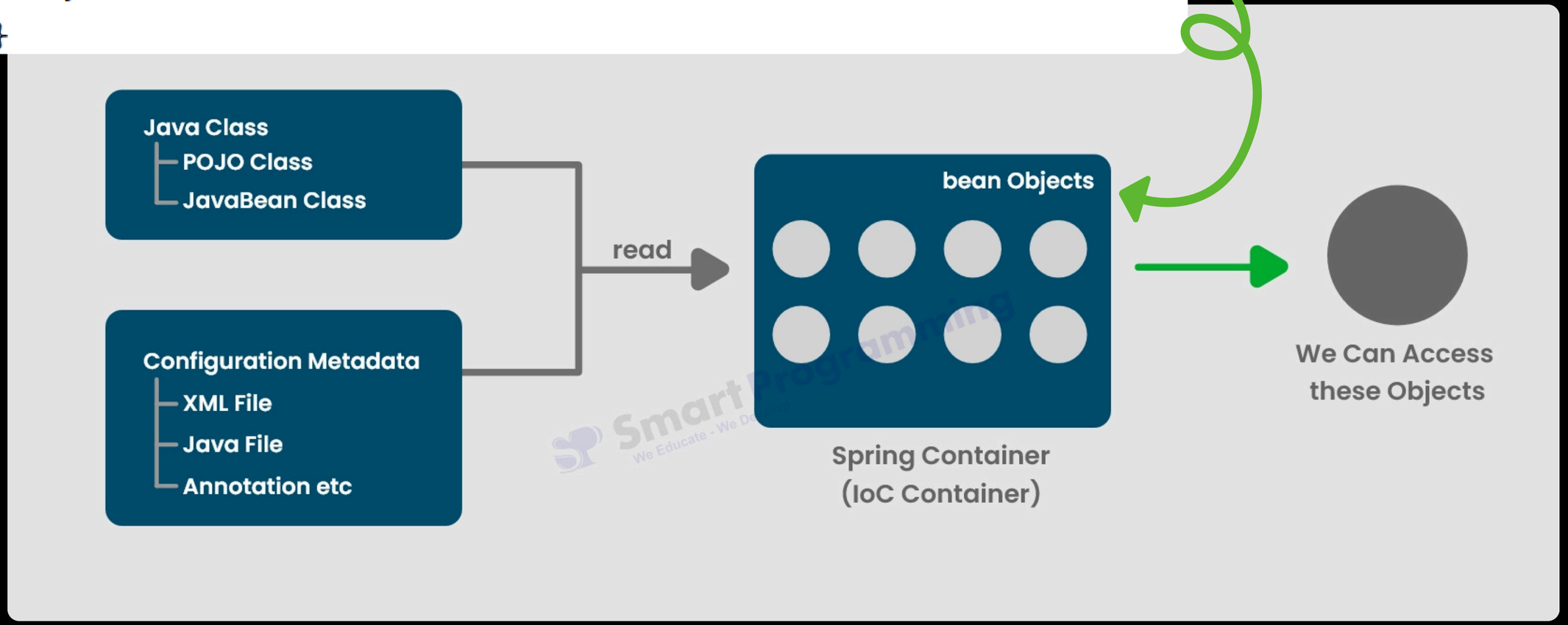
```java
/**
 * @author Ezzaim Mohammed
 **/
public class Dev {
    public void build() {
        System.out.println("Hello!, 3IA Community");
    }
}
```

```java
@SpringBootApplication
public class FirstProjectApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstProjectApplication.class, args);

        Dev dev = new Dev();
        dev.build();
    }
}
```

# JVM



```java
@SpringBootApplication
public class FirstProjectApplication {
    public static void main(String[] args) {
        SpringApplication.run(FirstProjectApplication.class, args);
    }
}
```



```java
@SpringBootApplication
public class FirstProjectApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstProjectApplication.class, args);

        Dev dev = new Dev();
        dev.build();
    }
}
```

```java
public static ConfigurableApplicationContext run(Class<?> primarySource, String... args) {
    return run(new Class[]{primarySource}, args);
}
```

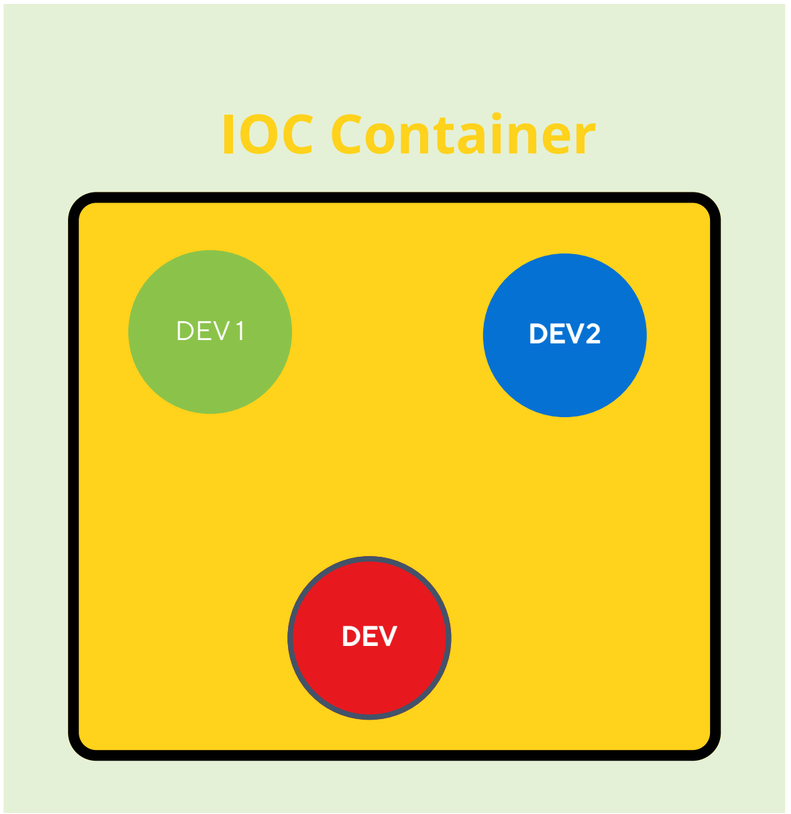
```java
public interface ConfigurableApplicationContext extends ApplicationContext, Lifecycle, Closeable {
    String CONFIG_LOCATION_DELIMITERS = ",; \t\n";
    String BOOTSTRAP_EXECUTOR_BEAN_NAME = "bootstrapExecutor";
    String CONVERSION_SERVICE_BEAN_NAME = "conversionService";
    String LOAD_TIME_WEAVER_BEAN_NAME = "loadTimeWeaver";
    String ENVIRONMENT_BEAN_NAME = "environment";
    String SYSTEM_PROPERTIES_BEAN_NAME = "systemProperties";
    String SYSTEM_ENVIRONMENT_BEAN_NAME = "systemEnvironment";
    String APPLICATION_STARTUP_BEAN_NAME = "applicationStartup";
    String SHUTDOWN_HOOK_THREAD_NAME = "SpringContextShutdownHook";
```

```java
public interface ApplicationContext extends EnvironmentCapable, ListableBeanFactory, HierarchicalBeanFactory
    @Nullable
    String getId();

    String getApplicationName();

    String getDisplayName();

    long getStartupDate();

    @Nullable
    ApplicationContext getParent();

    AutowireCapableBeanFactory getAutowireCapableBeanFactory() throws IllegalStateException;
}
```

```java
@Component
public class Dev {
    public void build() {
        System.out.println("Hello!, 3IA Community");
    }
}
```

```java
@SpringBootApplication
public class FirstProjectApplication {

    public static void main(String[] args) {
        ApplicationContext context = SpringApplication.run(FirstProjectApplication.class, args);

        Dev dev = context.getBean(Dev.class);
        dev.build();
    }
}
```

# JVM



```java
@SpringBootApplication
public class FirstProjectApplication {

    public static void main(String[] args) {
        ApplicationContext context = SpringApplication.run(FirstProjectApplication.class, args);
        String[] beanNames = context.getBeanDefinitionNames();
        for (String beanName : beanNames) {
            System.out.println(beanName);
        }

    }
}
```

```
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
firstProjectApplication
org.springframework.boot.autoconfigure.internalCachingMetadataReaderFactory
dev
```

| Méthode | Retourne |
| --- | --- |
| `context.getBeanDefinitionNames()` | **Nom des beans** (String) |
| `context.getBean(beanName)` | **Instance de l'objet** |

# MERCI !

@med Mohammed Ezzaim