

Inversion of Control (IoC) and Dependency Injection (DI)

Inversion of Control (IoC)

Definition: IoC is a design principle in which the control of object creation and management is transferred from the application code to a framework or container. This allows developers to focus on the core business logic while the framework takes care of the lifecycle and interactions of the objects.

Example with **Laptop** and **Alien** Classes:

- In the given example, the **Alien** class has a dependency on the **Laptop** class. Instead of **Alien** being responsible for creating an instance of **Laptop**, the Spring framework manages this for us.
- When the application starts, the Spring container creates the **Laptop** instance and injects it into the **Alien** class. This is a prime example of IoC, where the control of object creation is inverted from the user code to the framework.

```
@Component
public class Alien {
    @Autowired
    Laptop laptop; // Dependency on Laptop

    public void code() {
        laptop.compile(); // Calling the compile method on the injected Laptop
instance
    }
}
```

Dependency Injection (DI)

Definition: DI is a specific implementation of IoC that involves providing an object's dependencies from an external source rather than having the object create

them internally. This fosters loose coupling between classes and enhances testability and maintainability.

Example with **Laptop** and **Alien** Classes:

- In the provided example, the **Alien** class is dependent on the **Laptop** class. By using the **@Autowired** annotation, Spring automatically injects an instance of **Laptop** into **Alien**.
- This eliminates the need for the **Alien** class to instantiate a **Laptop** object directly, promoting loose coupling.

```
@Component
public class Laptop {
    public void compile() {
        System.out.println("Compiling...");
    }
}
```

```
@Component
public class Alien {
    @Autowired
    Laptop laptop; // Dependency Injection occurs here

    public void code() {
        laptop.compile(); // Utilizing the Laptop instance
    }
}
```

Summary

- **IoC** allows the Spring framework to manage the lifecycle and relationships of objects, freeing developers from having to write boilerplate code for object management.
- **DI** focuses specifically on how dependencies are provided to objects, enabling better separation of concerns. In this example, the **Alien** class does not know how to create a **Laptop**, it only knows that it has one (thanks to Spring's DI), leading to more modular and testable code.

Commented [1]: should be clarified more, seems confusing or needs more clarity