Spring Core

Beans

Q: What is a Spring Bean?

A: When it comes to Spring, pretty much any and every custom object is a Bean – be it your domain object POJO, the REST Controllers, the heart and soul of your Service and DAO layers, every custom object is primarily a bean so long as its lifecycle is managed by Spring.

Typically, before Spring, it was the responsibility of the developer to instantiate, initialize and maintain the lifecycle of objects which are required for the application to function. One of the primary responsibilities in all this was to ensure you don't create so many objects in the Heap so as to cause OutOfMemoryError.

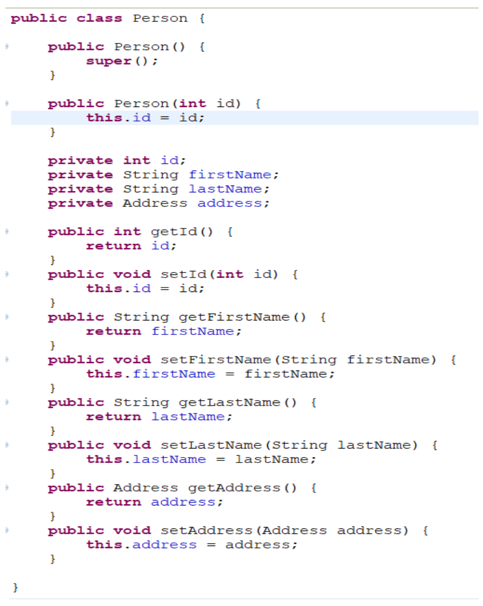
However, with Spring, all of this lifecycle control has inversed. The Spring container is now responsible for instantiating, initializing and maintaining the lifecycle of the objects. The developer's job is now only to let Spring know (by means of configurations) the details about the required object, like –

- Type of the object
- Initial values of the state variables
- Constructor based mandatory dependency or Setter based optional dependency of attributes

Below is a classic example of Spring bean definitions –

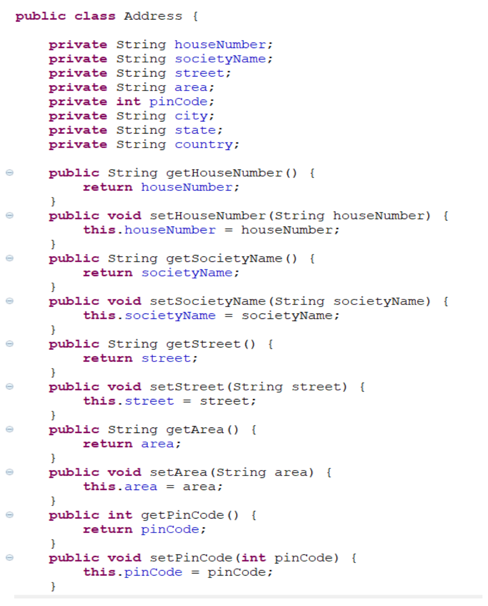
Classes – Every Person has-an Address

Person.java



```java
public class Person {

    public Person() {
        super();
    }

    public Person(int id) {
        this.id = id;
    }

    private int id;
    private String firstName;
    private String lastName;
    private Address address;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public Address getAddress() {
        return address;
    }
    public void setAddress(Address address) {
        this.address = address;
    }

}
```

Address.java



```java
public class Address {

    private String houseNumber;
    private String societyName;
    private String street;
    private String area;
    private int pinCode;
    private String city;
    private String state;
    private String country;

    public String getHouseNumber() {
        return houseNumber;
    }
    public void setHouseNumber(String houseNumber) {
        this.houseNumber = houseNumber;
    }
    public String getSocietyName() {
        return societyName;
    }
    public void setSocietyName(String societyName) {
        this.societyName = societyName;
    }
    public String getStreet() {
        return street;
    }
    public void setStreet(String street) {
        this.street = street;
    }
    public String getArea() {
        return area;
    }
    public void setArea(String area) {
        this.area = area;
    }
    public int getPinCode() {
        return pinCode;
    }
    public void setPinCode(int pinCode) {
        this.pinCode = pinCode;
    }
```

XML-based Bean Declaration –

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="person" class="io.budbak.spring.core.beans.beans.Person">
    <constructor-arg name="id" value="1" />
    <property name="firstName" value="John" />
    <property name="lastName" value="Doe" />
    <property name="address" ref="address" />
  </bean>

  <bean id="address" class="io.budbak.spring.core.beans.beans.Address">
    <property name="houseNumber" value="123" />
    <property name="societyName" value="ABC" />
    <property name="street" ref="Main Street" />
    <property name="area">VIP Area</property>
    <property name="pinCode">100100</property>
    <property name="city" value="Buffalo" />
    <property name="state">New Jersey</property>
    <property name="country">USA</property>
  </bean>

</beans>
```

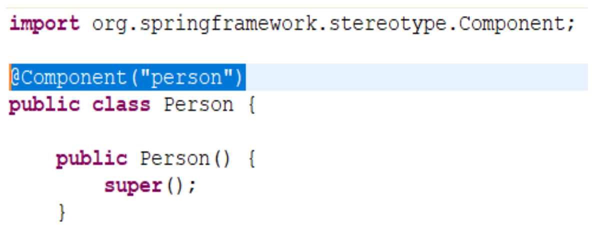
The above code snippets highlight both the ways to declare beans and their dependencies –

- Constructor-based –
  - Specify the constructor arguments and their values. E.g. Person.id
- Setter-based –
  - Specify the class attributes/properties and their values. E.g. Person.firstName
  - In case the property refers to another object, define using the “ref” attribute. E.g. Person.address -> references to Address bean.

Annotation-based Bean declaration –

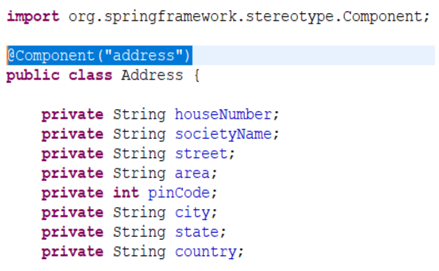
Annotate both Person and Address classes with @Component annotation.

Person.java



```java
import org.springframework.stereotype.Component;

@Component("person")
public class Person {

    public Person() {
        super();
    }
```

Address.java



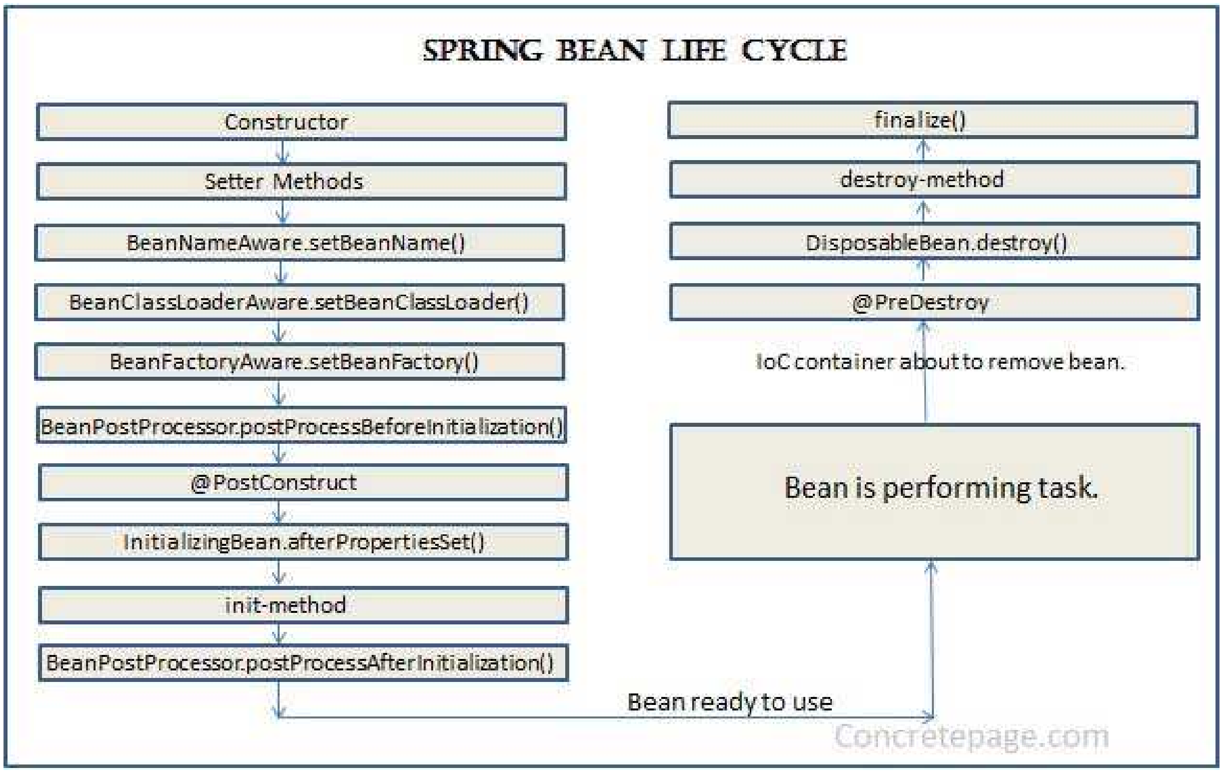
```java
import org.springframework.stereotype.Component;

@Component("address")
public class Address {

    private String houseNumber;
    private String societyName;
    private String street;
    private String area;
    private int pinCode;
    private String city;
    private String state;
    private String country;
```

In the Main Class where the beans are to be retrieved –

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;

@ComponentScan(basePackages = "io.budbak.spring.core.beans.beans")
public class Main {

    @Bean("person")
    public Person getPerson() {
        return new Person(1);
    }

    @Bean("address")
    public Address getAddress() {
        return new Address();
    }

}
```

Q: What is the Lifecycle of a Spring Bean?

A: Although Spring Bean Life Cycle and IoC and DI warrant a detailed article, the basic Spring Bean Life Cycle is best explained in the diagram below (Source: concretepage.com)


SPRING BEAN LIFE CYCLE
Constructor
Setter Methods
BeanNameAware.setBeanName()
BeanClassLoaderAware.setBeanClassLoader()
BeanFactoryAware.setBeanFactory()
BeanPostProcessor.postProcessBeforeInitialization()
@PostConstruct
InitializingBean.afterPropertiesSet()
init-method
BeanPostProcessor.postProcessAfterInitialization()
Bean ready to use
Bean is performing task.
IoC container about to remove bean.
@PreDestroy
DisposableBean.destroy()
destroy-method
finalize()
Concretepage.com

Hope this article on Spring Beans was useful. A detailed explanation is available on the official Spring Documentation.