Circular Dependencies

Circular dependencies are the scenarios when two or more beans try to inject each other via constructor.

Let's consider following two classes (outside of Spring framework):

```
class Driver {
  public Driver(Car car) {
  }
}

class Car {
  public Car(Driver driver) {
  }
}
```

How we are going to initialize above two classes via constructor? Let's try.

```
public class Test {
  public static void main(String[] args) {
      Car car = new Car( );// how we going to supply driver's instance here?
      Driver driver = new Driver(car);
  }
```

```
}
```

It's not possible to write a compilable code which can initialize exactly one instance of each Car and Driver, and pass to each other's constructor. We can, however, refactor our above code and can pass circular references via setters but that would kill the semantics of 'initializing mandatory final fields via constructors.

The same problem Spring faces when it has to inject the circular dependencies via constructor. Spring throws BeanCurrentlyInCreationException in that situation. Let's see some examples.

Example of circular dependencies in Spring

Beans

```
@Component
public class Car {
  private final Driver driver;

  public Car(Driver driver) {
      this.driver = driver;
  }
}

@Component
public class Driver {
  private final Car car;
```

```
  public Driver(Car car) {
      this.car = car;
  }
}
```

```
@ComponentScan
@Configuration
public class ExampleMain {

    public static void main(String[] args) {
        new AnnotationConfigApplicationContext(ExampleMain.class);
    }
}
```

Output

Caused by: org.springframework.beans.factory.BeanCurrentlyInCreationException: Error creating bean with name 'car': Requested bean is currently in creation: Is there an unresolvable circular reference?

In next examples, we will see how to avoid BeanCurrentlyInCreationException.

1 - Successfully injecting circular dependencies using setter injection

This example shows how use circular dependencies without having BeanCurrentlyInCreationException in Spring by using setter injection.

Example

Beans having circular dependencies

```
package com.piseth.java.school;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Car {
  private Driver driver;

  @Autowired
  public void setDriver(Driver driver) {
      this.driver = driver;
  }

  public void drive() {
      System.out.println("driven by: " + driver);
  }
```

```
}

@Component
public class Driver {
  private Car car;

  @Autowired
  public void setCar(Car car) {
      this.car = car;
  }

  public void showCar() {
      System.out.println("my car: " + car);
  }
}
```

Main class

```
@ComponentScan
@Configuration
public class ExampleMain {
```

```
  @Autowired
  private Car car;
  @Autowired
  private Driver driver;

  @PostConstruct
  public void postConstruct() {
      car.drive();
      driver.showCar();
  }

  public static void main(String[] args) {
      new AnnotationConfigApplicationContext(
              ExampleMain.class);
  }
}
```

Output

```
driven by: com.piseth.java.school.Driver@62262219
my car: com.piseth.java.school.Car@4f808dcb
```

2 - Successfully injecting circular dependencies using @Lazy Annotation

This example shows how to successfully use circular dependencies in Spring by using @Lazy annotation at injection point. Spring uses a proxy instead of the real object at the injection point. This proxy delays the initialization of the underlying object until it is first used. This is the preferable way to handle circular dependencies in Spring.

Examples

Bean with circular dependencies

We need to use @Lazy only with one bean.

```
@Component
public class Car {
  private final Driver driver;

  public Car(@Lazy Driver driver) {
      this.driver = driver;
  }

  public void drive() {
      System.out.println("driven by: " + driver);
  }
}
```

```
@Component
public class Driver {
  private final Car car;

  public Driver(Car car) {
      this.car = car;
  }

  public void showCar() {
      System.out.println("my car: " + car);
  }
}
```

Main class

```
@ComponentScan
@Configuration
public class ExampleMain {

  @Autowired
  private Car car;
  @Autowired
```

```
  private Driver driver;


  @PostConstruct
  public void postConstruct() {
      car.drive();
      driver.showCar();
  }


  public static void main(String[] args) {
      new AnnotationConfigApplicationContext(ExampleMain.class);
  }
}
```

Output

```
driven by: com.piseth.java.school.Driver@5b65f8e1
my car: com.piseth.java.school.Car@69a9f050
```