

Wiring Beans

- 3 ways to wire Beans
 - + Link the beans by directly calling the `@Bean` method that creates them (wiring)
 - + Enable Spring to provide us a value using a method parameter (auto-wiring) framework controls the application at execution
 - + Dependency Injection (IoC principle - inversion of control) - `@Autowired`

1) The first 2 approaches

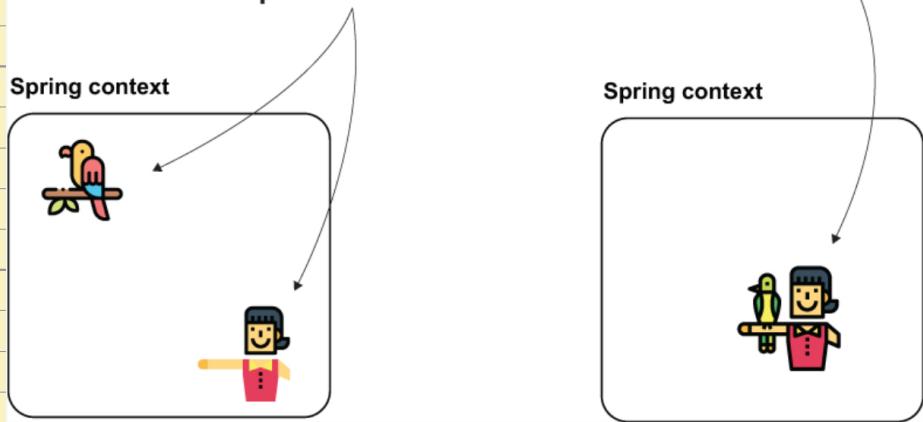
Register Beans
to Spring Context

Don't use `@Bean` in config

Establish
a relationship

STEP 1:
In the Spring context
you add a parrot and
a person as beans.

STEP 2:
You make the person
own the parrot.



```

public class Parrot {
    private String name;
    // Omitted getters and setters
    @Override
    public String toString() {
        return "Parrot : " + name;
    }
}

public class Person {
    private String name;
    private Parrot parrot;
    // Omitted getters and setters
}

public class Main {
    public static void main(String[] args) {
        var context = new AnnotationConfigApplicationContext
            (ProjectConfig.class);
        Person person =
            context.getBean(Person.class);
        Parrot parrot =
            context.getBean(Parrot.class);
        System.out.println(
            "Person's name: " + person.getName());
        System.out.println(
            "Parrot's name: " + parrot.getName());
        System.out.println(
            "Person's parrot: " + person.getParrot());
    }
}

```

add Beans to config

```

@Configuration
public class ProjectConfig {
    @Bean
    public Parrot parrot() {
        Parrot p = new Parrot();
        p.setName("Koko");
        return p;
    }

    @Bean
    public Person person() {
        Person p = new Person();
        p.setName("Ella");
        return p;
    }
}

```

❶

❷

❸

❹

❺

❻

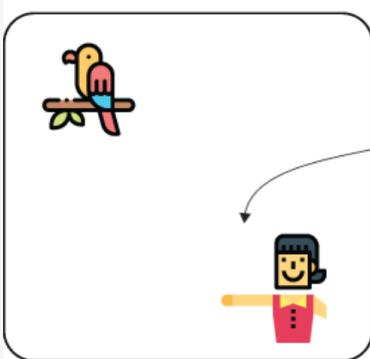
Work

Work

will not work cuz we haven't

built the relationship

Spring context



The person doesn't yet own the parrot.

The two beans are in the context, but no link has been established between them.

Diagram illustrating Spring's Bean Creation Logic:

```

graph TD
    subgraph "Spring context"
        direction TB
        S1[1. Spring calls the parrot() method to create the parrot bean and add it to the context.] --> S2[2. Spring calls the person() method to create the person bean and add it to the context.]
        S2 --> S3[3. Does this mean a second parrot instance is created here when Spring creates the person bean?]
        S3 -- YES --> S4[3A. Return directly the parrot bean from the Spring context without delegating the call anymore to the parrot() method.]
        S3 -- NO --> S4
    end

```

Code Example 1: Directly calling the method that returns the bean we want to set.

```

@Configuration
public class ProjectConfig {
    @Bean
    public Parrot parrot() {
        Parrot p = new Parrot();
        p.setName("Koko");
        return p;
    }

    @Bean
    public Person person() {
        Person p = new Person();
        p.setName("Ella");
        p.setParrot(parrot());
        return p;
    }
}

```

Annotation: make connection between Parrot and Person

Diagram: Spring context diagram showing a person holding a parrot.

We define the relationship between the person bean and the parrot bean by directly calling the method that returns the bean we want to set.

The result is the has-A relationship between the two beans. The person has-A (owns) the parrot.

Code Example 2: Instructing Spring to provide a bean from its context by defining a parameter for the method.

```

@Configuration
public class ProjectConfig {
    @Bean
    public Parrot parrot() {
        Parrot p = new Parrot();
        p.setName("Koko");
        return p;
    }

    @Bean
    public Person person(@Parrot parrot) {
        Person p = new Person();
        p.setName("Ella");
        p.setParrot(parrot);
        return p;
    }
}

```

We set the value of the person's attribute with the reference Spring provided.

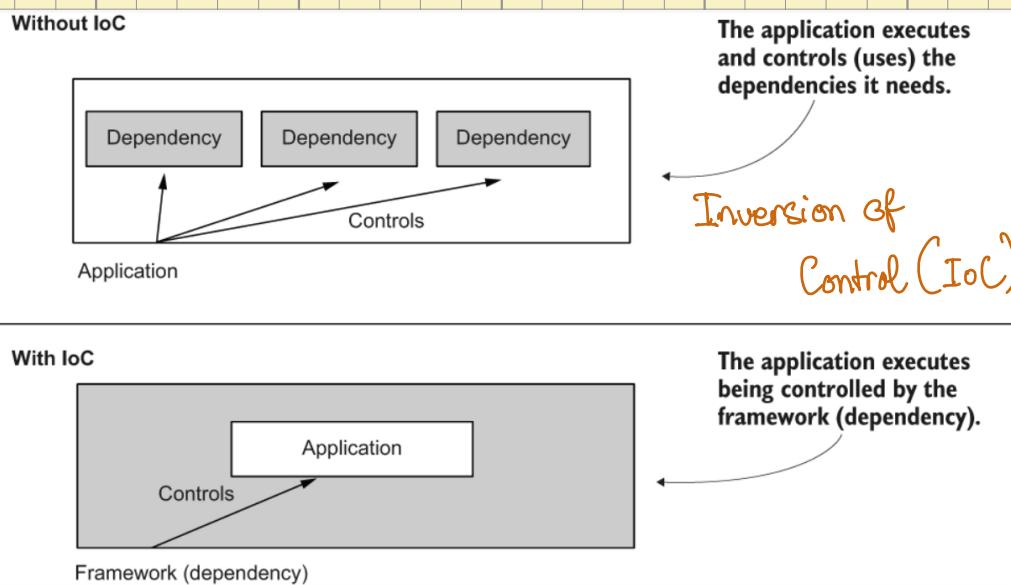
Diagram: Spring context diagram showing a person holding a parrot.

We instruct Spring to provide a bean from its context by defining a parameter for the method.

The result is the has-A relationship between the two beans. The person has-A (owns) the parrot.

Note: this is another way to wire Beans thru Bean methods' parameters

2) Dependency Injection



3 ways to
use @Autowired
annotation

- inject vals at class fields
- inject value at constructor
(* most used)
- inject value at setters
(not recommended in production code)

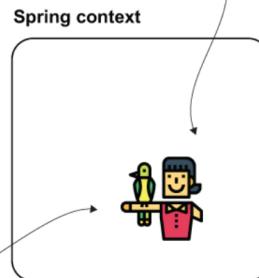
a) inject at class field

```
@Component  
public class Person {  
  
    private String name = "Ella";  
  
    @Autowired  
    private Parrot parrot;  
  
    // ...  
}
```

We instruct Spring to provide a bean from its context and set it directly as the value of the field, annotated with `@Autowired`. This way we establish a relationship between the two beans.

The stereotype annotation `@Component` instructs Spring to create and add a bean to the context of the type of this class: `Person`.

injection
at class's
field



```
@Component  
public class Person {  
  
    private String name = "Ella";  
  
    @Autowired  
    private Parrot parrot;  
  
    // Omitted getters and setters  
}
```

```
@Configuration  
@ComponentScan(basePackages = "beans")  
public class ProjectConfig {  
  
    // ...
}
```

→ Why is this approach not recommended?

look at this example

```
@Component  
public class Person {  
  
    private String name = "Ella";  
  
    @Autowired  
    private final Parrot parrot;  
  
}
```

this code will not compile
this will not work b/c
final variable has to
have a default value
(initial)

↳ difficult to manage
at initialization

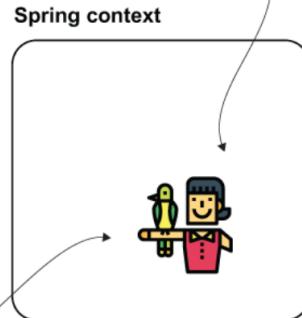
b) inject at class construction (most suggested)

```
@Component ←  
public class Person {  
  
    private String name = "Ella";  
  
    private final Parrot parrot;  
  
    @Autowired  
    public Person(Parrot parrot) {  
        this.parrot = parrot;  
    }  
  
    // ...  
}
```

When Spring creates the bean of type Person, it calls the constructor annotated with `@Autowired`. Spring provides a bean of type Parrot from its context as value of the parameter.

The stereotype annotation `@Component` instructs Spring to create and add a bean to the context of the type of this class: Person.

this way is recommended
bc we now
can set the
initial value
for final
field



make
testing
easier
too

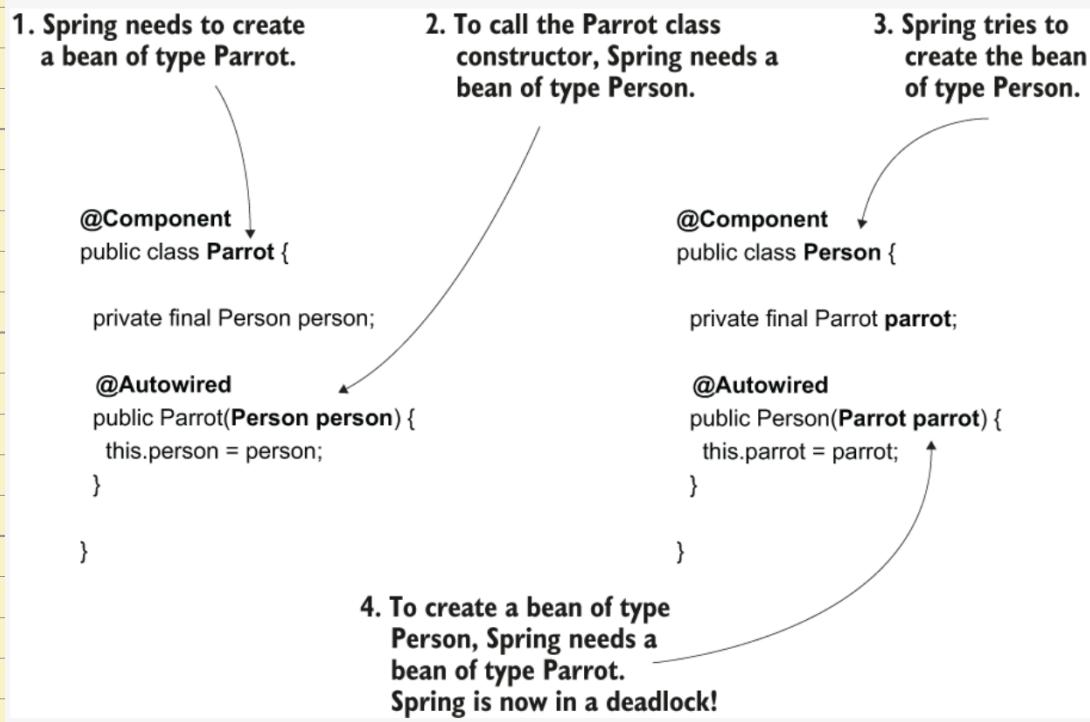
c) inject at setters

- More disadvantages than advantages
- used in old apps

```
@Component  
public class Person {  
  
    private String name = "Ella";  
  
    private Parrot parrot;  
  
    // Omitted getters and setters  
  
    @Autowired  
    public void setParrot(Parrot parrot) {  
        this.parrot = parrot;  
    }  
}
```

3) Circular Dependencies

- Bean A needs Bean B to construct, but Bean B also requires Bean A during construction
 - circular dependencies
 - deadlock



* Solution : rewrite the code

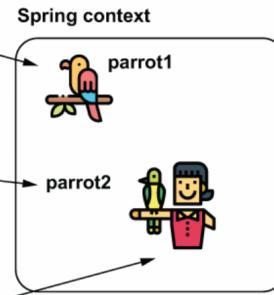
4) Choosing from multiple beans in Spring Context

1. identifier of the parameter is the same

as the name of the bean in Spring context

```
@Configuration  
public class ProjectConfig {  
  
    @Bean  
    public Parrot parrot1() {  
        Parrot p = new Parrot();  
        p.setName("Koko");  
        return p;  
    }  
  
    @Bean  
    public Parrot parrot2() {  
        Parrot p = new Parrot();  
        p.setName("Miki");  
        return p;  
    }  
  
    @Bean  
    public Person person(Parrot parrot2) {  
        Person p = new Person();  
        p.setName("Ella");  
        p.setParrot(parrot2);  
        return p;  
    }  
}
```

In the context, we define two beans of type Parrot. These two beans will take the name of the methods creating them: parrot1 and parrot2. We also define one bean of type Person.



Spring will provide the value of the bean whose name is the same as the name of the parameter we defined.

2. otherwise,

- + use `@Primary`
- + use `@Qualifier` e.g `@Qualifier("parrot2") Parrot parrot`
- + throw exception

```
@Configuration
public class ProjectConfig {

    @Bean
    public Parrot parrot1() {
        Parrot p = new Parrot();
        p.setName("Koko");
        return p;
    }

    @Bean
    public Parrot parrot2() {
        Parrot p = new Parrot();
        p.setName("Miki");
        return p;
    }

    @Bean
    public Person person(
        @Qualifier("parrot2") Parrot parrot) {
        Person p = new Person();
        p.setName("Ella");
        p.setParrot(parrot);
        return p;
    }
}
```