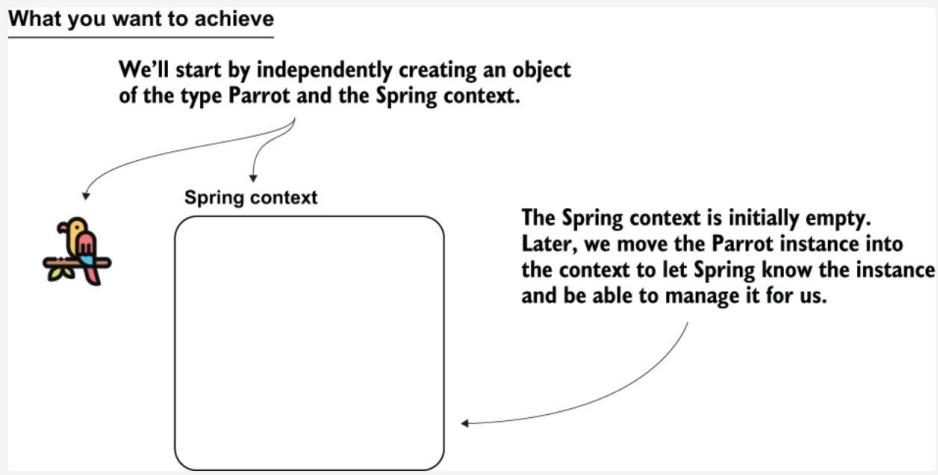
Defining Bean

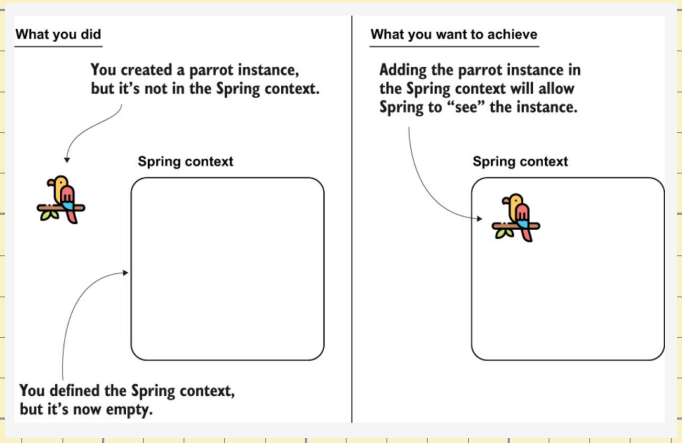


- instead of allocating the Parrot object in the memory manually (new Parrot()), we doing sth else

Java 11 datatype

create Spring Context instance

```
public class Main {

  public static void main(String[] args) {
    var context =
        new AnnotationConfigApplicationContext();

    Parrot p = new Parrot();
  }
}
```



all of the Beans created will be called by default upon executing the main method (in code ordering)

1) adding Beans to the Spring context

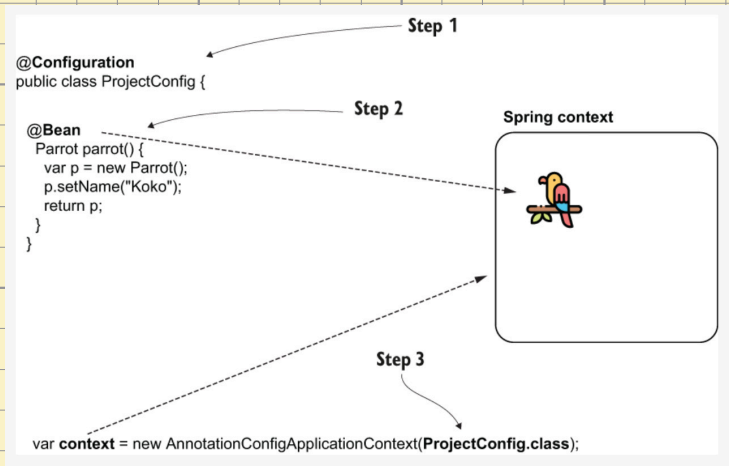
- follow these steps:

a) define a config class (@Configuration) - configure the context of Spring

b) add method to the config class that returns the object instance we want to add to the context and annotate with @Bean

c) make Spring use the created config in (a)

configuration class is a class to instruct Spring to do specific actions



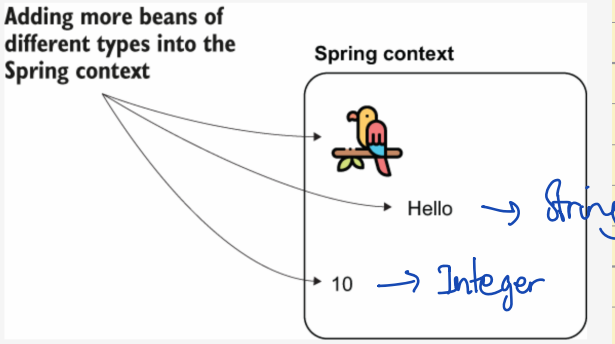
```
public class Main {

  public static void main(String[] args) {
    var context =
      new AnnotationConfigApplicationContext(
        ProjectConfig.class);

    Parrot p = context.getBean(Parrot.class);

    System.out.println(p.getName());
  }
}
```

register config to Spring Context

❶ → tell Spring to get you the Parrot bean

↳ getName of Parrot

* What if we want to add more Beans with different types?



→ String

→ Integer

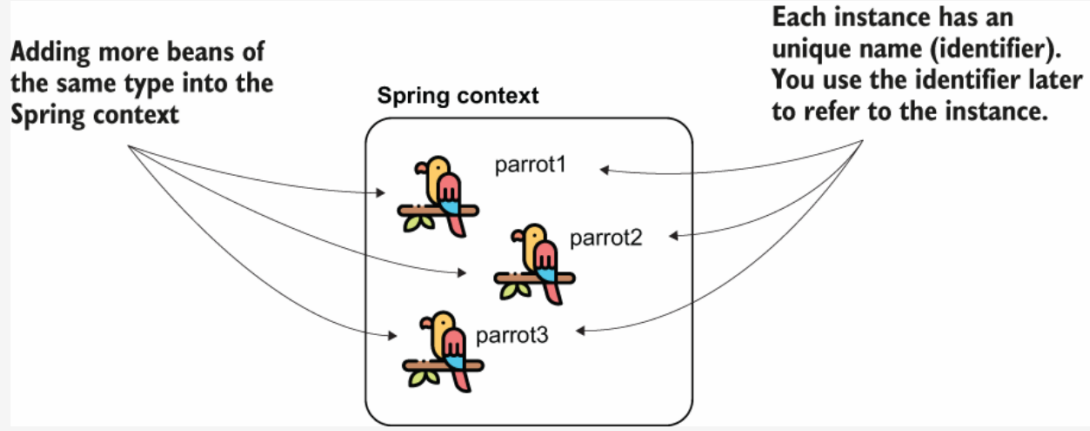
Solution:

add more Beans in the Config and get Bean with the corresponding data type

```
@Configuration
public class ProjectConfig {

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

  @Bean                    ❶
  String hello() {
    return "Hello";
  }

  @Bean                    ❷
  Integer ten() {
    return 10;
  }
}
```

```
public class Main {

  public static void main(String[] args) {
    var context = new AnnotationConfigApplicationContext(
                    ProjectConfig.class);

    Parrot p = context.getBean(Parrot.class);     ❶
    System.out.println(p.getName());

    String s = context.getBean(String.class);
    System.out.println(s);

    Integer n = context.getBean(Integer.class);
    System.out.println(n);
  }
}
```

* What if we want to add more Beans with similar types?



.getBean(Parrot.class) only throws exception in this case cuz of ambiguity

```
@Configuration
public class ProjectConfig {

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

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

  @Bean
  Parrot parrot3() {
    var p = new Parrot();
    p.setName("Riki");
    return p;
  }
}
```

```
public class Main {

  public static void main(String[] args) {
    var context = new
      AnnotationConfigApplicationContext(ProjectConfig.class);

    Parrot p = context.getBean("parrot2", Parrot.class);    ❶
    System.out.println(p.getName());

  }
}
```

← specify the method

✓

or we can do it like this

(*) @Bean(name = "Miki")

(*) @Bean(value: "Miki")

(*) @Bean("Miki")

```
@Bean(name = "miki")    ❶
Parrot parrot2() {
  var p = new Parrot();
  p.setName("Miki");    ❷
  return p;
}
```

if we don't setName for parrot, will return null

or

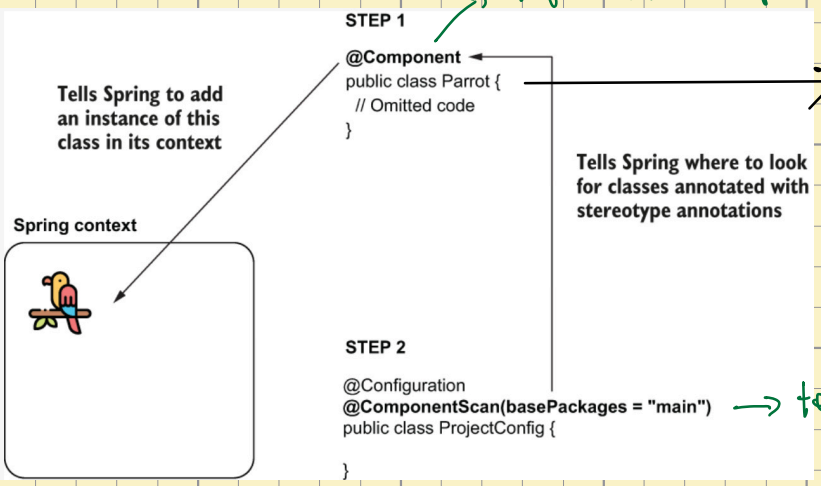
these are just referred as methodname, not Parrot name

2) Use stereotype annotation @Primary to set the default Bean (one and only)

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

- There are a lot of other stereotype annotations such as @Component, @ComponentScan, @Primary ...

'the Bean that is used in application is called first' followed with other Beans




```
@Component
public class Parrot {

  private String name;

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

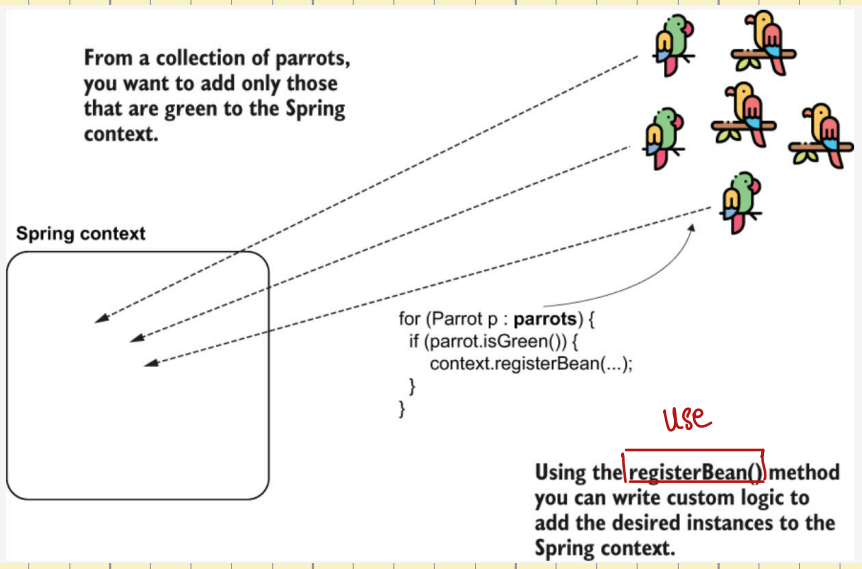
```
@Component
public class Parrot {

  private String name;

  @PostConstruct
  public void init() {
    this.name = "Kiki";
  }

  // Omitted code
}
```

@PostConstruct is called after the class creation

Also have @PreDestroy but not recommended to use

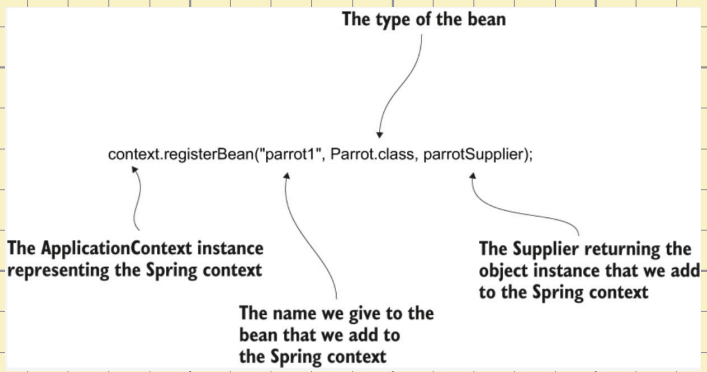
Using the `@Bean` annotation	Using stereotype annotations
1. You have full control over the instance creation you add to the Spring context. It is your responsibility to create and configure the instance in the body of the method annotated with `@Bean`. Spring only takes that instance and adds it to the context as-is. 2. You can use this method to add more instances of the same type to the Spring context. Remember, in section 2.1.1 we added three `Parrot` instances into the Spring context. 3. You can use the `@Bean` annotation to add to the Spring context any object instance. The class that defines the instance doesn't need to be defined in your app. Remember, earlier we added a `String` and an `Integer` to the Spring context. 4. You need to write a separate method for each bean you create, which adds boilerplate code to your app. For this reason, we prefer using `@Bean` as a second option to stereotype annotations in our projects.	1. You only have control over the instance after the framework creates it. 2. This way, you can only add one instance of the class to the context. 3. You can use stereotype annotations only to create beans of the classes your application owns. For example, you couldn't add a bean of type `String` or `Integer` like we did in section 2.1.1 with the `@Bean` annotation because you don't own these classes to change them by adding a stereotype annotation. 4. Using stereotype annotations to add beans to the Spring context doesn't add boilerplate code to your app. You'll prefer this approach in general for the classes that belong to your app.

3) Programmatically adding Bean to Spring context



What if we only want to add the green birds to Spring context?

<T> void registerBean (String beanName, "name"
Class<T> beanClass, Parrot.class
lambda expression ← Supplier<T> supplier,
customize Bean ← BeanDefinitionCustomizer ... customizer)
e.g: make it primary




```
public class Main {

  public static void main(String[] args) {
    var context =
      new AnnotationConfigApplicationContext(
          ProjectConfig.class);

      Parrot x = new Parrot();
      x.setName("Kiki");

      Supplier<Parrot> parrotSupplier = () -> x;

      context.registerBean("parrot1",
        Parrot.class, parrotSupplier);

      Parrot p = context.getBean(Parrot.class);
      System.out.println(p.getName());
  }
}
```

example using BeanDefinitionCustomizer

```
context.registerBean("parrot1",
                     Parrot.class,
                     parrotSupplier,
                     bc -> bc.setPrimary(true));
```