

# Chapter-2:

## The Spring Context - Defining Beans

Uppcode Software  
Engineer Team

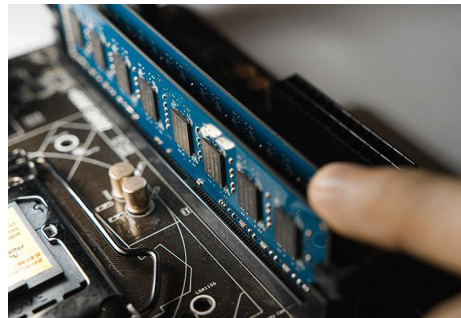


# КОНТЕНТ

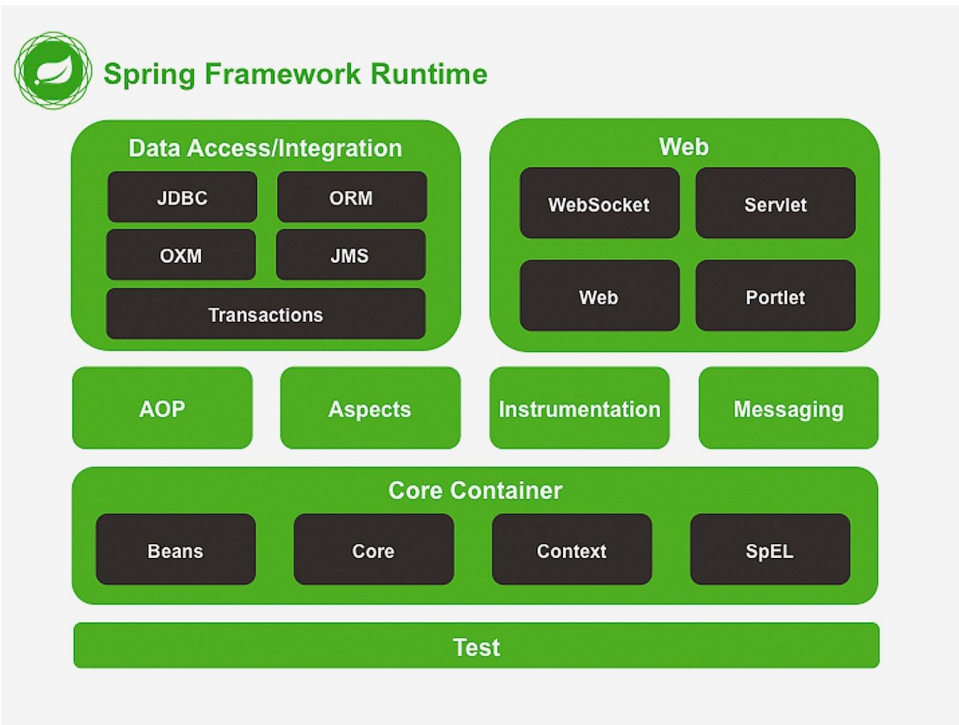
1. Зачем нужен контекст Spring (Spring Context)
2. Добавление бин(bean) в контекст Spring
3. Через аннотация @Bean
4. Через Стереотипных аннотаций
5. Программно
6. Заключение
7. Ссылка

# 1. Что такое Spring Context

- В Spring приложениях его еще называют **Application Context**.
- Imagine the context as a place in the memory of your app in which we add all the object instances that we want the framework to manage. (Представьте контекст как место в памяти приложения, куда добавляются все экземпляры объектов.)
- По умолчанию Spring ничего о них не знает (By default, Spring doesn't know any of the objects you define in your application).
- Чтобы фреймворк эти объекты «увидел», их нужно добавить в контекст.
- Эти экземпляры мы называем бинами (beans).
- **Bean** представляет собой объект, управляемые контейнером Spring.

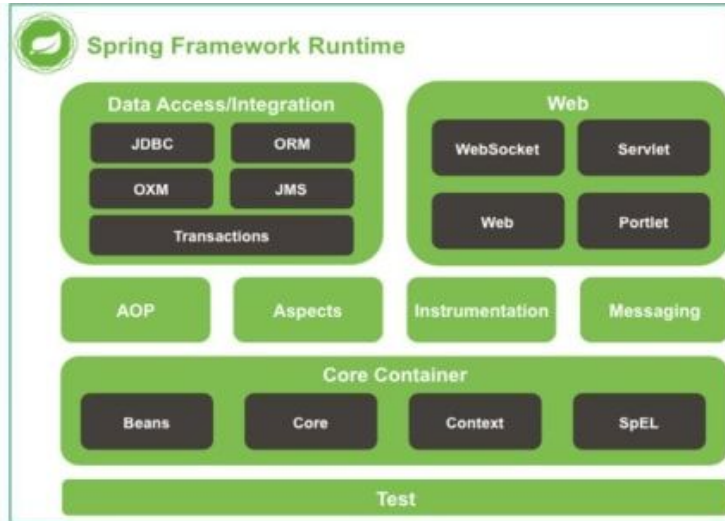
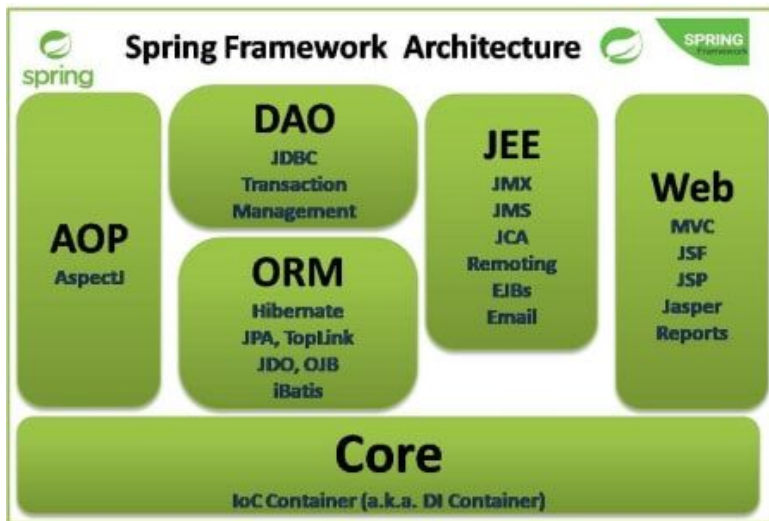


# 1. Архитектура Spring(1/2)



[ссылка на картинку](#)

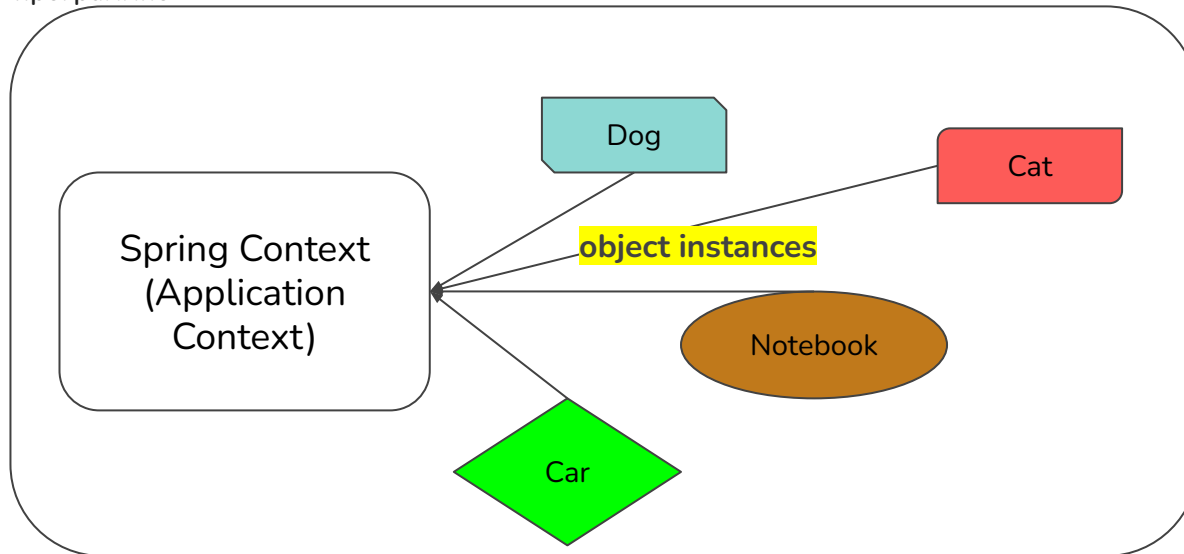
# 1. Архитектура Spring(2/2)



## 2. Добавление Бинов в Контекст Spring(1/6)

Существуют следующие способы включить бин в контекст (далее мы рассмотрим их подробнее):

- посредством аннотации @Bean;
- посредством стереотипных аннотаций;
- программно





## 2. Добавление зависимостей для контекста(2/6)

- Этот модуль предоставляет основные возможности контейнера Spring, включая управление жизненным циклом биннов, внедрение зависимостей и другие функции, необходимые для работы с компонентами Spring-приложения.

```
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-context</artifactId>  
  <version>6.1.6</version>  
</dependency>
```

## 2. Добавление зависимостей для контекста(3/6)

The screenshot shows an IDE interface with a project named 'spring-context-project' in the 'master' branch. The project structure on the left includes 'src/main/java/uz/temur' and 'pom.xml'. The 'pom.xml' file is open in the editor, showing the following XML structure:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.example</groupId>
  <artifactId>spring-context-project</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>maven-plugin</packaging>
  <name>spring-context-project Maven Plugin</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <kotlin.version>1.9.22</kotlin.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.30</version>
    </dependency>
  </dependencies>
</project>
```

The 'External Libraries' section on the left lists several dependencies, with 'Maven: org.springframework:spring-context:5.3.30' highlighted in a red box. Green arrows point from this entry to the corresponding dependency entry in the XML file.



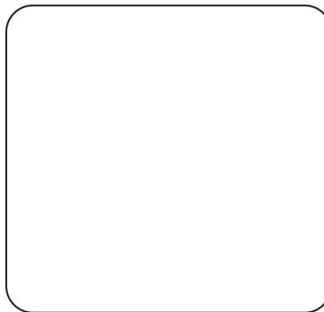
## 2. Что мы хотим получить(4/6)

Что мы хотим получить

Для начала создадим объект типа Parrot  
и — отдельно от него — контекст Spring



Контекст Spring



Изначально контекст Spring пуст. Позже мы добавим  
туда экземпляр Parrot, чтобы сообщить Spring о его  
существовании и чтобы фреймворк мог им управлять

## 2. Создадим объекты(5/6)

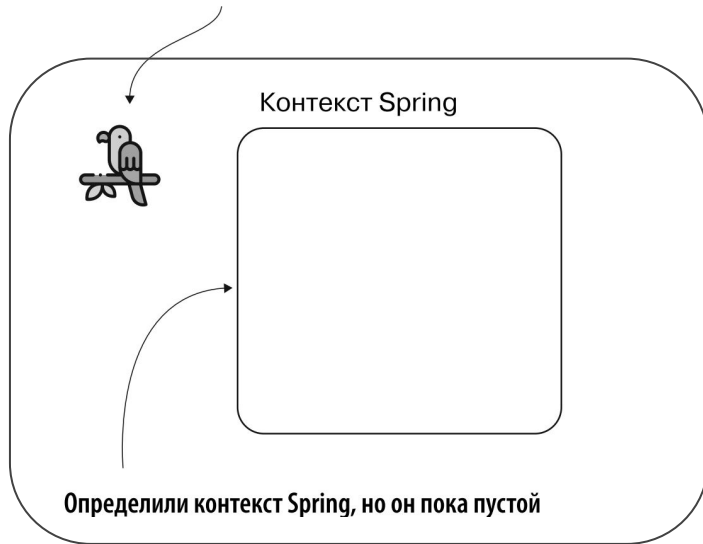
```
public class Parrot {  
  
    private String name;  
  
    //Getter and Setters  
}
```

```
@SpringBootApplication  
public class SpringContextTest5Application {  
  
    public static void main(String[] args) {  
  
        AnnotationConfigApplicationContext context =  
            new AnnotationConfigApplicationContext();  
  
        Parrot parrot = new Parrot();  
    }  
}
```

## 2. Что нам нужно сделать?(6/6)

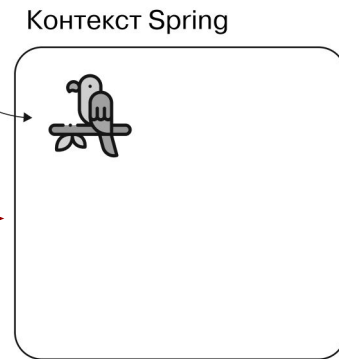
Что мы сделали

Создали экземпляр Parrot,  
но он не находится в контексте Spring



Что мы хотим получить

Добавив экземпляр Parrot в контекст Spring,  
мы сделаем этот экземпляр видимым для Spring





## Create: Project Spring Core (Maven)

### STEP - 1

Add Object instance to Spring Context.

Using: @Bean



### 3. Добавление бинов в контекст Spring с помощью аннотации @Bean(1/4)

1. Определить в проекте класс конфигурации(с а аннотацией @Configuration).
2. Добавить в класс конфигурации метод, возвращающий экземпляр объекта, который мы хотим добавить в контекст, и снабдить этот метод аннотацией @Bean

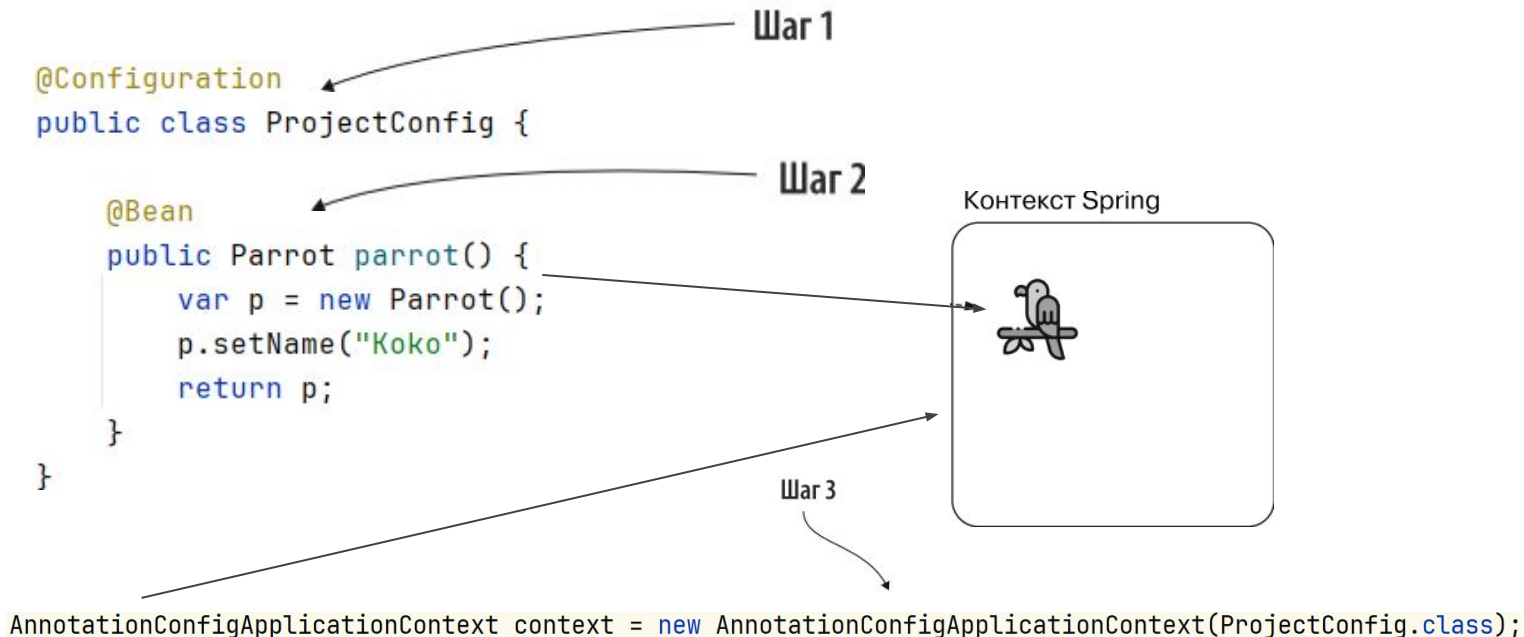
```
→ @Configuration
public class ProjectConfig {

    → @Bean
    public Parrot parrot(){
        Parrot parrot = new Parrot();
        parrot.setName("Koko");
        return parrot;
    }
}
```

#### Примечание:

\*\* В Java слова ключевое слово **var** используются для объявления локальных переменных с поддержкой вывода типов.

### 3. Добавление(2/4)



Операции, которые нужно выполнить, чтобы добавить бин в контекст с помощью аннотации `@Bean`. Добавив экземпляр в контекст Spring, мы сообщаем фреймворку об этом объекте, после чего фреймворк сможет им управлять

### 3. Определение конфигурационного класса и метода @Bean(3/4)

1. Создадим файла конфигурации

```
@Configuration
public class ProjectConfig {
```

2. Создание метода, которые возвращает бин, с аннотацией @Bean

```
@Configuration
public class ProjectConfig {
```

```
    @Bean
```

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

Добавив аннотацию @Bean, мы сообщаем Spring, что при инициализации контекста нужно вызвать этот метод и добавить в контекст возвращенное им значение

Назначаем имя попугая, которое далее будем использовать при тестировании приложения

Spring добавляет в контекст экземпляр класса Parrot, возвращаемый методом

```
}
```

### 3. Инициализация контекста Spring на основании созданного класса конфигурации и обращение к экземпляру Parrot из контекста.(4/4)

@SpringBootApplication

```
public class SpringContextTest5Application {
```

```
    public static void main(String[] args) {
```

```
        AnnotationConfigApplicationContext context =  
            new AnnotationConfigApplicationContext(ProjectConfig.class);
```

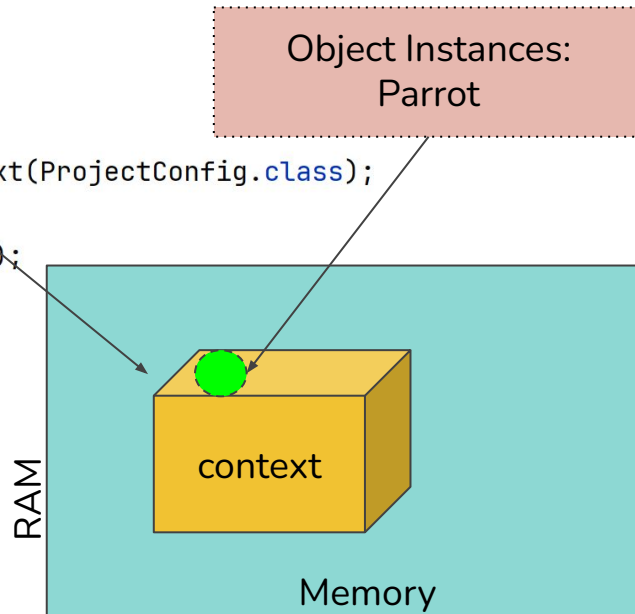
```
        Parrot parrot = context.getBean(Parrot.class);  
        System.out.println(parrot.getName());
```

```
    }
```

```
}
```

Результат: Koko

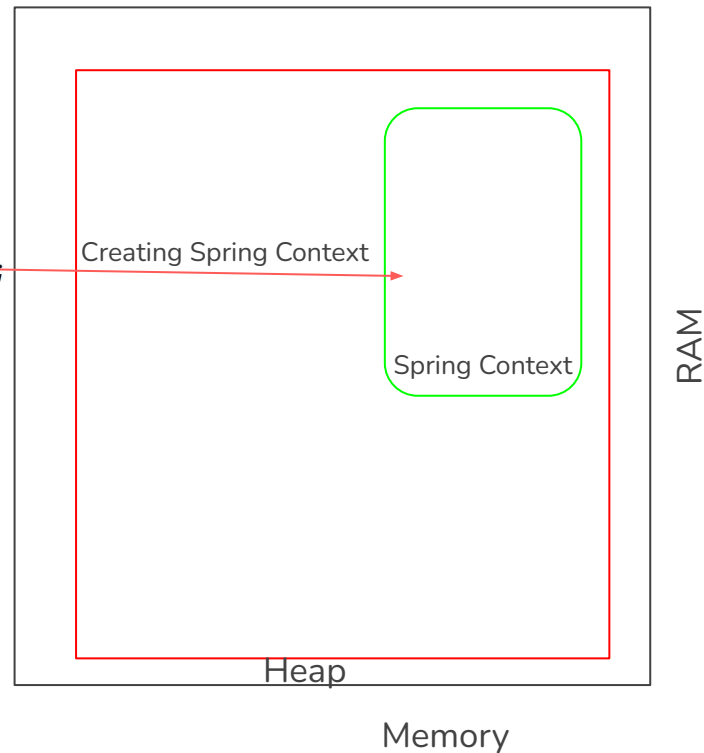
Process finished with exit code 0





### 3. Как это работает(1/3)

```
public class SpringContextTest5Application {  
  
    public static void main(String[] args) {  
  
        AnnotationConfigApplicationContext context =  
            new AnnotationConfigApplicationContext();  
  
    }  
}
```



### 3. Как это работает(2/3)

```
@Configuration
public class ProjectConfig {
```

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

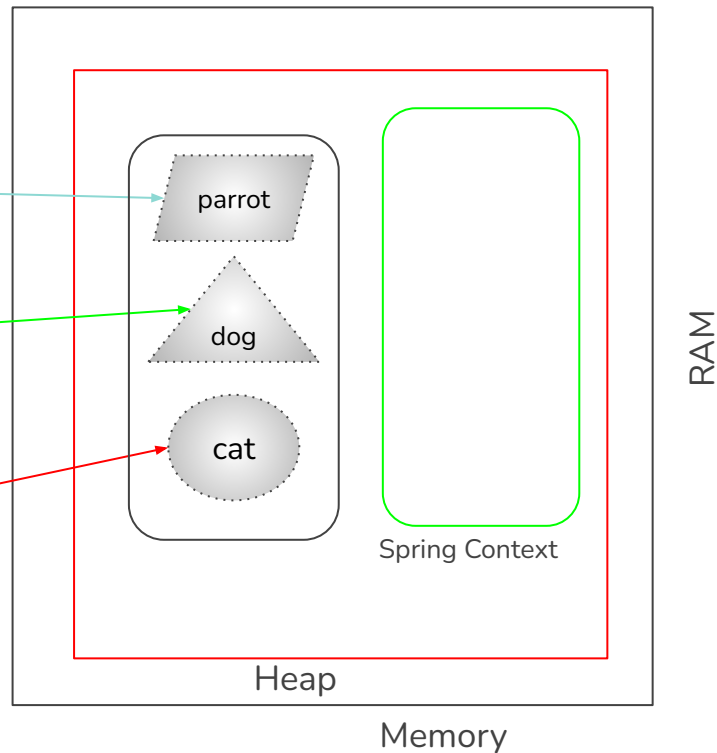
Creating Object

```
    @Bean
    public Dog dog() {
        Dog dog = new Dog();
        dog.setName("Rex");
        return dog;
    }
```

Creating Object

```
    @Bean
    public Cat cat() {
        Cat cat = new Cat();
        cat.setName("Tom");
        return cat;
    }
```

Creating Object



### 3. Как это работает(3/3)

```
@Configuration
public class ProjectConfig {

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

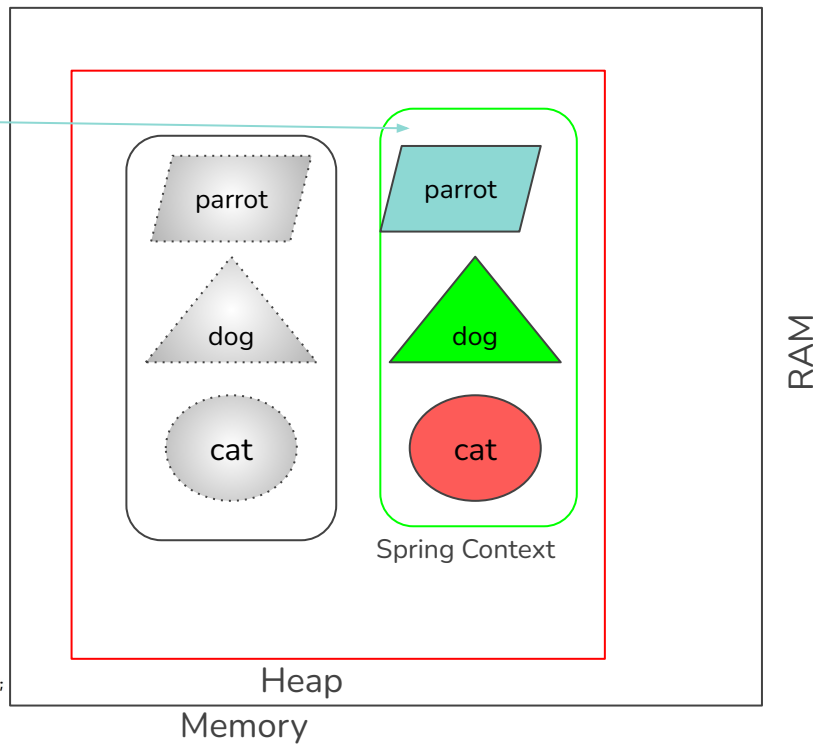
    @Bean
    public Dog dog() {
        Dog dog = new Dog();
        dog.setName("Rex");
        return dog;
    }

    @Bean
    public Cat cat() {
        Cat cat = new Cat();
        cat.setName("Tom");
        return cat;
    }
}

public class SpringContextTest5Application {

    public static void main(String[] args) {

        AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(ProjectConfig.class);
    }
}
```



### 3. Ещё примеры(1/2)

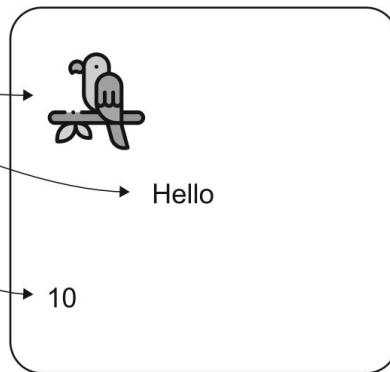
Добавим в контекст еще двух бинов

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

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

Добавление бинов  
разных типов  
в контекст Spring

Контекст Spring



### 3. Вывод в консоль значение двух бинов(2/2)

@SpringBootApplication

```
public class SpringContextTest5Application {
```

```
    public static void main(String[] args) {
```

```
        AnnotationConfigApplicationContext context =
```

```
            new AnnotationConfigApplicationContext(ProjectConfig.class);
```

```
        Parrot parrot = context.getBean(Parrot.class);
```

```
        System.out.println(parrot.getName());
```

```
        String s = context.getBean(String.class);
```

```
        System.out.println(s);
```

```
        Integer n = context.getBean(Integer.class);
```

```
        System.out.println(n);
```

```
    }
```

```
}
```

В КОНСОЛЕ: Koko  
Hello  
10

Process finished with exit code 0

### 3. Можно ли добавить туда несколько объектов одного типа?(1/4)

@Configuration

```
public class ProjectConfig {
```

@Bean

```
public Parrot parrot() {  
    Parrot parrot = new Parrot();  
    parrot.setName("Koko");  
    return parrot;  
}
```

@Bean

```
public Parrot parrot1() {  
    Parrot parrot = new Parrot();  
    parrot.setName("Kasha");  
    return parrot;  
}
```

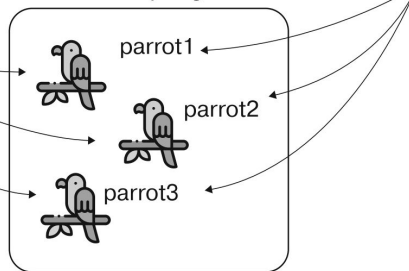
@Bean

```
public Parrot parrot2() {  
    Parrot parrot = new Parrot();  
    parrot.setName("Miki");  
    return parrot;  
}
```

```
}
```

Добавление в контекст Spring  
нескольких бинов одного типа

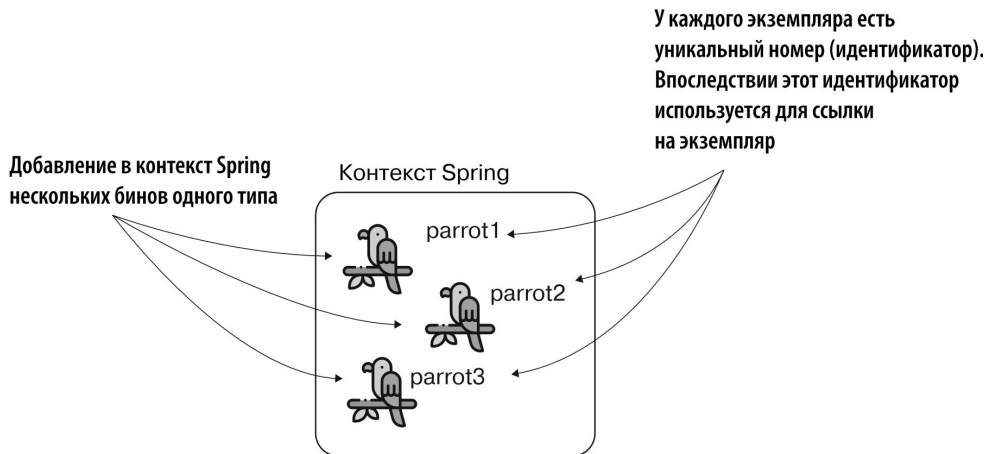
Контекст Spring



У каждого экземпляра есть  
уникальный номер (идентификатор).  
Впоследствии этот идентификатор  
используется для ссылки  
на экземпляр

### 3. Можно ли добавить туда несколько объектов одного типа?(2/4)

- Не путайте имя бина с кличкой попугая.
- В нашем примере именами(идентификаторами) бинов в контексте Spring являются parrot1, parrot2 parrot3(имена методов с аннотации @Bean, определяющих соответствующие бина).
- Для Spring атрибут объекта не имеет никакого значения.



### 3. Как их использовать?(3/4)

```
@SpringBootApplication
public class SpringContextTest5Application {

    public static void main(String[] args) {

        AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(ProjectConfig.class);

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

В этой строке мы получим  
исключение, так как Spring не может  
догадаться, на какой из трех  
экземпляров Parrot вы ссылаетесь

- При выполнении приложения получим исключение, подобное представленному в следующем примере кода.

Exception in thread "main" org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'uz.springcontexttest5.Parrot' available: expected single matching bean but found 3: parrot,parrot1,parrot2



### 3. Как исправить ошибку (Exception) (4/4)

The screenshot shows an IDE with two Java files. The left file, `ProjectConfig.java`, contains three `@Bean` methods: `parrot()` returning a `Parrot` named 'Koko', `parrot1()` returning a `Parrot` named 'Kesha', and `parrot2()` returning a `Parrot` named 'Miki'. The right file, `SpringContextTest5Application.java`, is a `@SpringBootApplication` class with a `main` method that creates an `AnnotationConfigApplicationContext` with `ProjectConfig.class` and attempts to retrieve a bean named 'parrot1' of type `Parrot`. The console at the bottom shows the application running successfully, with messages for connecting to the target VM, printing 'Kesha', disconnecting, and finishing with exit code 0. Red arrows indicate the flow of data: one arrow points from the `parrot1()` method in `ProjectConfig.java` to the `context.getBean("parrot1", Parrot.class)` line in the application, and another arrow points from the `System.out.println(koko.getName());` line to the console output 'Kesha'.

```
ProjectConfig.java
6
7
8 @Configuration
9 public class ProjectConfig {
10
11     @Bean
12     public Parrot parrot() {
13         Parrot parrot = new Parrot();
14         parrot.setName("Koko");
15         return parrot;
16     }
17
18     @Bean
19     public Parrot parrot1(){
20         Parrot parrot = new Parrot();
21         parrot.setName("Kesha");
22         return parrot;
23     }
24
25     @Bean
26     public Parrot parrot2(){
27         Parrot parrot = new Parrot();
28         parrot.setName("Miki");
29         return parrot;
30     }
31 }

SpringContextTest5Application.java
1 package uz.springcontexttest5;
2
3 import org.springframework.boot.autoconfigure.SpringBootApplication;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6 @SpringBootApplication
7 public class SpringContextTest5Application {
8
9     public static void main(String[] args) {
10
11         AnnotationConfigApplicationContext context =
12             new AnnotationConfigApplicationContext(
13                 ProjectConfig.class);
14
15         Parrot koko = context.getBean( name: "parrot1", Parrot.class);
16         System.out.println(koko.getName());
17     }
18 }
19
20
21
```

Debug: SpringContextTest5Application

Debugger Console

```
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
Connected to the target VM, address: '127.0.0.1:11821', transport: 'socket'
Kesha
Disconnected from the target VM, address: '127.0.0.1:11821', transport: 'socket'
Process finished with exit code 0
```

### 3. Дополнительное информация(1/2)

Если вы хотите присвоить бину другое имя то можете его указать в аннотации @Bean в качестве атрибута name или value.

```
@Bean(name = "koko")
```

```
@Bean(value = "koko")
```

```
@Bean("koko")
```

```
@Bean(name = "koko")  
public Parrot parrot() {  
    var parrot = new Parrot();  
    parrot.setName("Koko");  
    return parrot;  
}
```

Любой из следующих вариантов кода меняет имя бина на **koko**.  
Any of the following code options changes the bean name to **koko**.

### 3. Дополнительное информация(2/2)

```
package uz.springcontexttest5;

import jakarta.annotation.PostConstruct;
import org.springframework.stereotype.Component;

public class Parrot {

    private String name;

    public String getName() {
        return name;
    }

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

```
package uz.springcontexttest5;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;

@Configuration
public class ProjectConfig {

    @Bean(name = "koko")
    public Parrot parrot() {
        Parrot parrot = new Parrot();
        parrot.setName("Koko");
        return parrot;
    }

    @Bean(value = "kesha")
    public Parrot parrot1() {
        Parrot parrot = new Parrot();
        parrot.setName("Kesha");
        return parrot;
    }

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

```
package uz.springcontexttest5;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

@SpringBootApplication
public class SpringContextTest5Application {

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

        Parrot koko = context.getBean("koko", Parrot.class);
        System.out.println(koko.getName());

        Parrot kesha = context.getBean("kesha", Parrot.class);
        System.out.println(kesha.getName());

        Parrot miki = context.getBean("miki", Parrot.class);
        System.out.println(miki.getName());
    }
}
```

Debug: SpringContextTest5Application

Debugger Console

C:\Program Files\Java\jdk-17\bin\java.exe ...

Connected to the target VM, address: '127.0.0.1:1187', transport: 'socket'

Koko

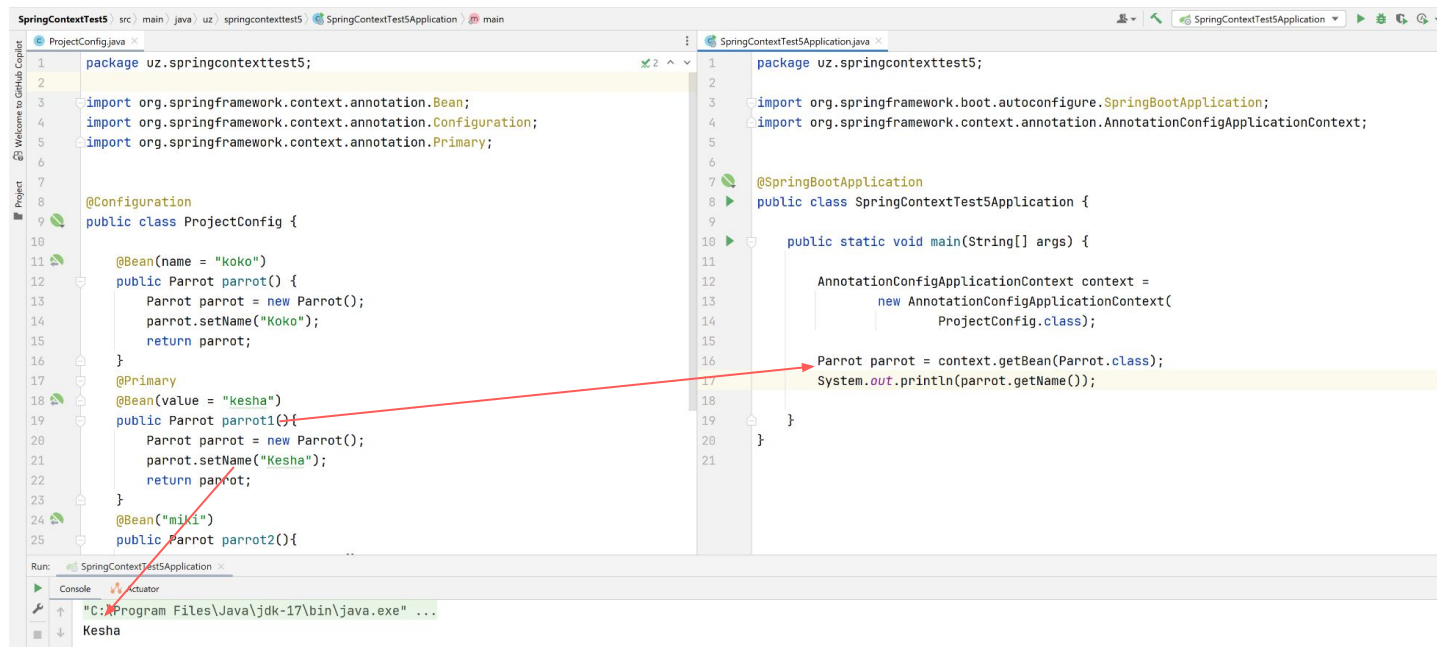
Kesha

Miki

Disconnected from the target VM, address: '127.0.0.1:1187', transport: 'socket'

Process finished with exit code 0

### 3. Определение бина в качестве первичного (Defining a bean as primary(@Primary))(1/1)



```
1 package uz.springcontexttest5;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.context.annotation.Primary;
6
7 @Configuration
8 public class ProjectConfig {
9
10     @Bean(name = "koko")
11     public Parrot parrot() {
12         Parrot parrot = new Parrot();
13         parrot.setName("Koko");
14         return parrot;
15     }
16
17     @Primary
18     @Bean(value = "kesha")
19     public Parrot parrot1() {
20         Parrot parrot = new Parrot();
21         parrot.setName("Kesha");
22         return parrot;
23     }
24
25     @Bean("miki")
26     public Parrot parrot2() {
27
28     }
29 }
```

```
1 package uz.springcontexttest5;
2
3 import org.springframework.boot.autoconfigure.SpringBootApplication;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6 @SpringBootApplication
7 public class SpringContextTest5Application {
8
9     public static void main(String[] args) {
10
11         AnnotationConfigApplicationContext context =
12             new AnnotationConfigApplicationContext(
13                 ProjectConfig.class);
14
15         Parrot parrot = context.getBean(Parrot.class);
16         System.out.println(parrot.getName());
17     }
18 }
```

Run: SpringContextTest5Application

Console

"C:\Program Files\Java\jdk-17\bin\java.exe" ...

Kesha



## STEP - 2

Add Object instance to Spring Context.

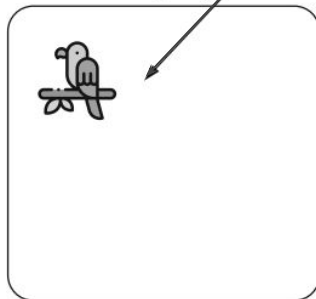
Using: Stereotype Annotation

### 3. Добавление бинов в контекст Spring с помощью стереотипных аннотации(1/3)

- Отметить аннотацией **@Component** те классы, экземпляры которых вы хотите поместить в контексте Spring.
- Использовать аннотацию **@ComponentScan** в классе конфигурации, сообщить Spring, где находится классы, отмеченные аннотацией **@Component**

Даем Spring инструкцию добавить  
экземпляр этого класса в контексте

Контекст Spring



```
@Component
public class Parrot {
    //Код класса
}
```

```
@Configuration
@ComponentScan(basePackages = "uz.springcontexttest5")
public class ProjectConfig {
}
```

## 4. Применение стереотипной аннотации к классу Parrot(2/3)

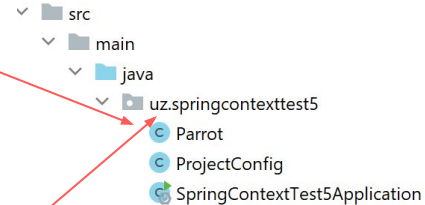
```
@Component
public class Parrot {

    private String name;

    public String getName() {
        return name;
    }

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

Поставив перед классом аннотацию `@Component`, мы даем Spring команду создать экземпляр этого класса и добавить его в контекст



```
@Configuration
@ComponentScan(basePackages = "uz.springcontexttest5")
public class ProjectConfig {

}
```

Используя в аннотации атрибут `basePackages`, мы указываем Spring, где находятся классы со стереотипными аннотациями

## 4. Тестирование конфигурации Spring в методе main(3/3)

```
@SpringBootApplication
public class SpringContextTest5Application {

    public static void main(String[] args) {

        AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(ProjectConfig.class);

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

Выводит в консоль стандартное представление экземпляра `String`, полученное из контекста Spring

Выводит в консоль `null`, поскольку мы не дали кличку этому попугаю, добавленному в контекст Spring

```
2024-05-03T10:52:41.224+05:00 INFO 22240 --- [SpringContextTest5] [
2024-05-03T10:52:41.229+05:00 INFO 22240 --- [SpringContextTest5] [
2024-05-03T10:52:41.739+05:00 INFO 22240 --- [SpringContextTest5] [
uz.springcontexttest5.Parrot@41477a6d
null
```





## 4. Стереотип аннотации(1/1)

1. @Component
2. @Service
3. @Repository
4. @Controller

## 4. Преимущества и недостатки двух способов добавление бинов в контекст Spring (1/1)

|                 | Аннотации @Bean   | Стереотипные аннотации   |
|-----------------|---|--|
| Контроль        | Вы сами контролируете создания экземпляра и описываете конфигурацию в теле метода   | После того, как фреймворк его создаст.   |
| Гибкость        | С аннотацией @Bean в Spring можно добавить в контекст экземпляр любого объекта, даже если класс этого объекта не определен в приложении. Например, мы добавили в контекст Spring экземпляры типов String и Integer. | Нет. Например не получится загрузить бин типа String или Integer.  |
| Перегрузка кода | Да. Для каждого бина, добавляемого в контекст Spring приходится писать отдельный метод.   | Нет. При добавлении бинов в контекст Spring посредством стереотипных аннотаций в приложении не появляется новый шаблонный код. Как правило, этот способ более предпочтителен, если класс принадлежит приложению. |
| Разнообразность | Spring позволяет добавить несколько экземпляров одного типа в свой контекст, что полезно для управления множеством объектов одного класса.  | Таким образом можно добавить в контекст только один экземпляр класса.  |



## 4. Использование `@PostConstruct` для управления созданным экземпляром(1/4)

- `@Bean` мы могли определить имя каждого экземпляра.
- `@Component` у нас нет возможности сделать что-либо после того, как Spring вызвал конструктор класса.
  - Для этого можно воспользоваться аннотацией `@PostConstruct`
- Spring вызовет метод с этой аннотацией после того, как закончится выполнение конструктора.
- Точнее, метод, помеченный аннотацией `@PostConstruct`, будет вызван после создания бина и внедрения всех его зависимостей, но до того, как этот бин будет предоставлен для использования другим бинам или компонентам в контексте приложения.

```
@Component
public class Parrot {

    private String name;

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



## 4. Using `@PostConstruct`(2/4)

- **Spring calls the methods annotated with `@PostConstruct` only once, just after the initialization of bean properties.** Keep in mind that these methods will run even if there's nothing to initialize.
- The method annotated with `@PostConstruct` can have any access level, but it can't be static.
- One possible use of `@PostConstruct` is populating a database. For instance, during development, we might want to create some default users:

## 4. Как работает @PostConstruct(3/4)

The screenshot shows an IDE with two main code files: `SpringContextTest5Application.java` and `Parrot.java`. The `SpringContextTest5Application` class is annotated with `@SpringBootApplication` and contains a `main` method. The `Parrot` class is annotated with `@Component` and has a `@PostConstruct` annotated `init` method. Red annotations and arrows highlight the execution flow: 1. `war` (warning) points to the `main` method. 2. `war` (warning) points to the `context` variable declaration. 3. `war` (warning) points to the `init` method call. 4. `war` (warning) points to the `println` statement. The console output shows the execution of the `main` method, printing the name of the `Parrot` object, which is `Koko`. The process finished with exit code 0.

```
SpringContextTest5Application.java
5
6
7 @SpringBootApplication
8 public class SpringContextTest5Application {
9
10     public static void main(String[] args) {
11
12         AnnotationConfigApplicationContext context =
13             new AnnotationConfigApplicationContext(
14                 ProjectConfig.class);
15
16         Parrot parrot = context.getBean(Parrot.class);
17
18         System.out.println(parrot.getName());
19     }
20 }
21
22
23
24
```

```
Parrot.java
5
6
7 @Component
8 public class Parrot {
9
10     private String name;
11
12     @PostConstruct
13     public void init(){
14         this.name = "Koko";
15     }
16
17     public String getName() {
18         return name;
19     }
20
21     public void setName(String name) {
22         this.name = name;
23     }
24 }
25
26
27
```

Run: SpringContextTest5Application

Console

"C:\Program Files\Java\jdk-17\bin\java.exe" ...

Koko

Process finished with exit code 0

## 4. Using @PreDestroy(4/4)

### **@PreDestroy**

- A method annotated with *@PreDestroy* runs only once, just before Spring removes our bean from the application context.
- Same as with *@PostConstruct*, the methods annotated with *@PreDestroy* can have any access level, but can't be static.

```
@Component
public class UserRepository {

    private DbConnection dbConnection;

    @PreDestroy
    public void preDestroy() {
        dbConnection.close();
    }
}
```



## STEP - 3

Add Object instance to Spring Context.

Using: Register Bean



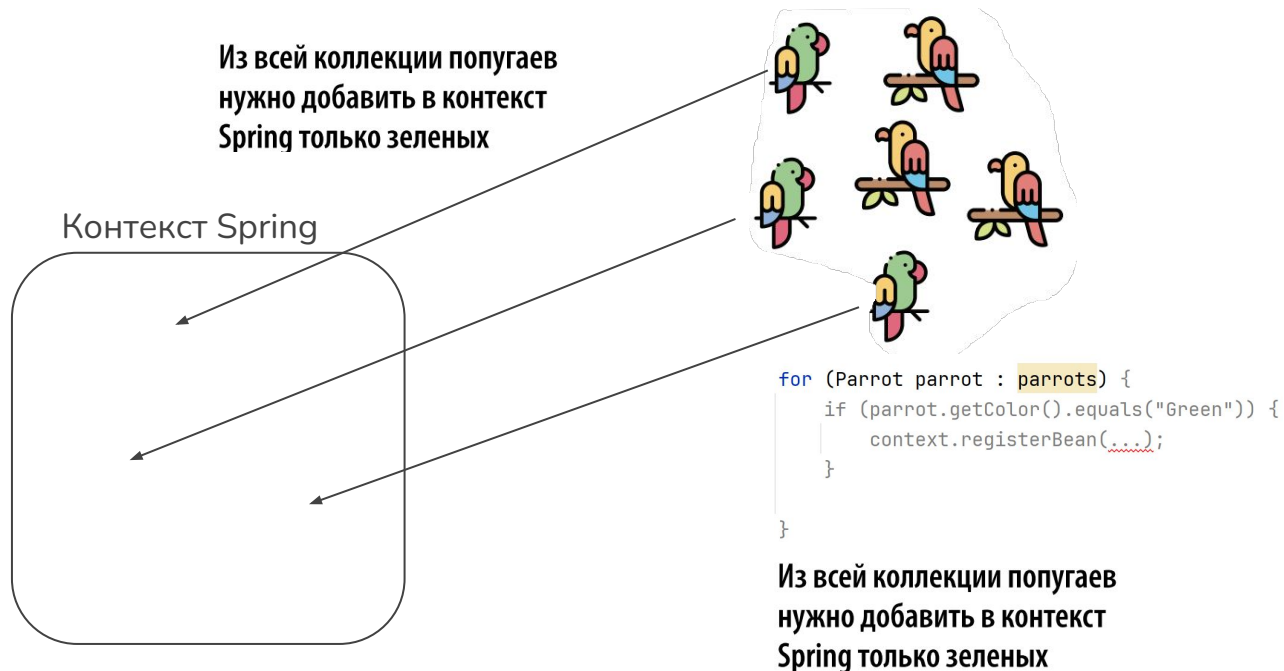
## 5. Программное добавление бинов в контекст Spring(1/6)

- Данный способ используется в тех случаях, когда нужно реализовать нестандартное добавление бинов в контекст в возможностей **@Bean**.
- Стереотипные аннотаций для этого недоступно.

```
if (condition) {  
    registerBean(b1); ← Если условие истинно, добавить в контекст Spring некий бин  
  
} else {  
  
    registerBean(b2); ← Иначе: добавить в контекст Spring другой бин  
}
```



## 5. Основные принципы(2/6)





## 5. Метод registerBean()(3/6)

- Метод позволяет добавлять в контекст Spring выбранные экземпляры объектов.
- Метод принимает четыре параметра

```
<T> void registerBean(  
    String beanName,  
    Class<T> beanClass,  
    Supplier<T> supplier,  
    BeanDefinitionCustomizer... customizers);
```



## 5. Метод `registerBean()` (4/6)

1. `beanName`, - это имя бина, если не хотите присвоить имя бину при вызове метода присвойте этому параметру значение **`null`(пустой)**.
2. Класс который определяет бин, добавляемые в контекст. Например класс `Parrot`. То значение будет `Parrot.class`
3. Реализация **`Supplier`** нужна, чтобы возвращать значение экземпляра. Возвращает заданное значение, не принимая параметров.
4. `BeanDefinitionCustomizer` - это просто интерфейс, который используется для настройки различные настройки бина, например, чтобы их сделать бин первичными.

## 5. Примеры к методу registerBean() (5/6)

@SpringBootApplication

```
public class SpringContextTest5Application {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.
```

```
            run(SpringContextTest5Application.class, args);
```

```
        AnnotationConfigApplicationContext context =
```

```
            new AnnotationConfigApplicationContext(ProjectConfig.class);
```

```
        Parrot x = new Parrot();
```

```
        x.setName("Kiki");
```

← Создание экземпляра, который будет  
добавлен в контекст Spring

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

← Определение Supplier, который  
будет возвращать этот экземпляр

```
        context.registerBean( beanName: "parrot1",
```

```
            Parrot.class, parrotSupplier);
```

← Вызов метода registerBean(), добавляющего  
экземпляр в контекст Spring

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

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

← Чтобы убедиться, что бин добавлен  
в контекст, мы обращаемся к бину класса  
Parrot и выводим в консоль кличку попугая

```
    }  
}  
  
2024-05-03T14:41:00.585+05:00 INFO 20888 --- [SpringContextTest5] [  
2024-05-03T14:41:00.587+05:00 INFO 20888 --- [SpringContextTest5] [  
2024-05-03T14:41:00.982+05:00 INFO 20888 --- [SpringContextTest5] [  
Kiki
```

## 5. Дополнение к registerBean() (6/6)

- Чтобы задать добавляемому бину различные характеристики, например, можно сделать бин первичным изменив метода registerBean().

```
public static void main(String[] args) {
    SpringApplication
        .run(SpringContextTest5Application.class, args);
    AnnotationConfigApplicationContext context =
        new AnnotationConfigApplicationContext(ProjectConfig.class);

    Parrot x = new Parrot();
    x.setName("Kiki");
    Supplier<Parrot> parrotSupplier = () -> x;
    context.registerBean( beanName: "parrot1",
        Parrot.class, parrotSupplier);

    Parrot y = new Parrot();
    y.setName("Kesha");
    Supplier<Parrot> parrotSupplier1 = () -> y;
    context.registerBean( beanName: "parrot2", Parrot.class,
        parrotSupplier1,
        bc->bc.setPrimary(true));

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



## Заключение

- `@Bean` позволяет добавлять экземпляры объектов в контекст приложения, обеспечивая гибкость, но требует написания дополнительного кода.
- `@Component` позволяет создавать бины для собственных классов, требуя меньше кода и делая конфигурацию более читаемой.
- Метод `registerBean()` позволяет добавлять бины в контекст Spring с собственной логикой. Он доступен начиная с версии Spring 5 и более поздних.



# REFERENCE

1: [Spring Start Here](#)



## RESOURCES







**Thank you!**

Presented by

**Asadbek Quronboyev**

**(asadbek9805@gmail.com)**