17. Spring Framework (Inversion Of Control, Dependency Injection)

- Конфигурация проекта
- Инверсия контроля
- Способы внедрения зависимостей

Интерфейс Pet: (абстракция множества животных с методом say())

```
public interface Pet {
    public void say();
}
```

Example 1

Конфигурация проекта в applicationContext.xml: (определение бинов, аргументов конструктора, значений полей, и propertie файла)

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

    <bean id="myPet"
          class="org.example.Dog">
    </bean>
    <bean id="myPerson"
          class="org.example.Person">
        <constructor-arg ref="myPet"/>
        <property name="pet" ref="myPet"/>
        <property name="surname" value="${person.surname}"/>
        <property name="age" value="${person.age}"/>
    </bean>
</beans>
```

Создание контекста и генерация бина из класса Dog:

```
public class Test2 {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Pet pet = context.getBean("myPet", Pet.class);
        pet.say();
        context.close();
    }
```

Example 2

Определение бинов с конструктором для класса Person:

```
<bean id="myPet"
      class="org.example.Dog">
</bean>
<bean id="myPerson"
      class="org.example.Person">
    <constructor-arg ref="myPet"/>
    <property name="pet" ref="myPet"/>
    <property name="surname" value="${person.surname}"/>
    <property name="age" value="${person.age}"/>
</bean>
```

Получение бина Person и автоматическое внедрение для него зависимости Pet:

```
public static void main(String[] args) {
    ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext ("applicationContext.xml");

    Person person = context.getBean("myPerson", Person.class);
    person.callYourPet();
    System.out.println(person.getSurname());
    System.out.println(person.getAge());
    context.close();
}
```

Example 3

Определение бина Cat с параметром scope “prototype”, и методами init() и destroy():

```
<bean id="myCat"
      class="org.example.Cat"
      scope="prototype"
      init-method="init"
      destroy-method="destroy">
</bean>
```

Получение бинов:

```
public static void main(String[] args) {
    ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext2.xml");
    Cat myCat = context.getBean("myCat", Cat.class);
    Cat yourCat = context.getBean("myCat", Cat.class);

    myCat.setName("katy");
    yourCat.setName("Tom");

    System.out.println(myCat.getName());
    System.out.println(yourCat.getName());
    System.out.println(myCat);
    System.out.println(yourCat);
    context.close();
```

Example 4

Конфигурационный с использованием аннотаций:

```
@Configuration
@ComponentScan("org.example")
public class MyConfig {

}
```

Объявление бинов аннотацией @Component:

```
@Component
public class Puppy implements Pet {

    public Puppy(){
        System.out.println("Puppy bean is created");
    }
    @Override
    public void say() {
        System.out.println("aw-aw");
    }
```

Внедрение зависимостей с помощью аннотации @Autowired, для отдельного поля, конструктора или сеттера, использование аннотации @Qualifier и @Value:

```
@Component
public class Person2 {
    @Autowired
    @Qualifier("puppy")
    private Pet pet;
    private String surname;
    private int age;

    public Person2(){
        System.out.println("Person2 bean is created");
    }

    @Autowired
    public Person2(@Qualifier ("catBean") Pet pet){
        System.out.println("Person2 bean is created");
        this.pet = pet;
    }

    @Autowired
    @Qualifier("kittenBean")
    public void setPet(Pet pet) {
        System.out.println("Class Person2: set Pet");
        this.pet = pet;
    }

    public void callYourPet(){
        System.out.println("Hello my pet");
        pet.say();
    }

    public String getSurname() {
        return surname;
```

```
    }

    @Value("${person.surname}")
    public void setSurname(String surname) {
        System.out.println("Class Person: set surname");
        this.surname = surname;
    }

    public int getAge() {
        return age;
    }

    @Value("${person.age}")
    public void setAge(int age) {
        System.out.println("Class Person: set age");
        this.age = age;
    }
}
```

Создание контекста из конфигурационного файла и связывание зависимостей:

```
public class Test6 {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context =
                new AnnotationConfigApplicationContext(MyConfig.class);
        Person2 person2 = context.getBean("person2", Person2.class);
        person2.callYourPet();
        context.close();
```

Example 5

Использование аннотаций @Scope, @PostConstruct, @PreDestroy:

```
@Component
@Scope("singleton")
public class Lion implements Pet {
    private String name;

    public Lion(){
        System.out.println("Lion bean is created");
    }
    @Override
    public void say() {
        System.out.println("ROOAAARRRRR...");
    }
    @PostConstruct
    public void init(){
        System.out.println("Class Lion: Initialization");
    }

    @PreDestroy
    public void destroy(){
        System.out.println("Class Lion: Destroying");
    }
}
```

Example 6

Конфигурация бинов аннотацией @Bean:

```
@Configuration
@PropertySource("myApp.properties")
public class MyConfig2 {

    @Bean
    public Pet puppyBean(){
        return new Puppy();
    }

    @Bean
    public Pet catBean(){
        return new Cat();
    }

    @Bean
    public Person3 person3Bean(){
        return new Person3(catBean());
    }

    @Bean
    public Person2 person2Bean(){
        return new Person2(catBean());
    }
}
```

Создание контекста и внедрение зависимостей:

```
public static void main(String[] args) {
    AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(MyConfig2.class);

    Pet puppy = context.getBean("puppyBean", Pet.class);
    puppy.say();

    Person3 person3 = context.getBean("person3Bean", Person3.class);
    person3.callYourPet();

    Person2 person2 = context.getBean("person2Bean", Person2.class);
    System.out.println(person2.getAge());
    System.out.println(person2.getSurname() );

    context.close();
}
```