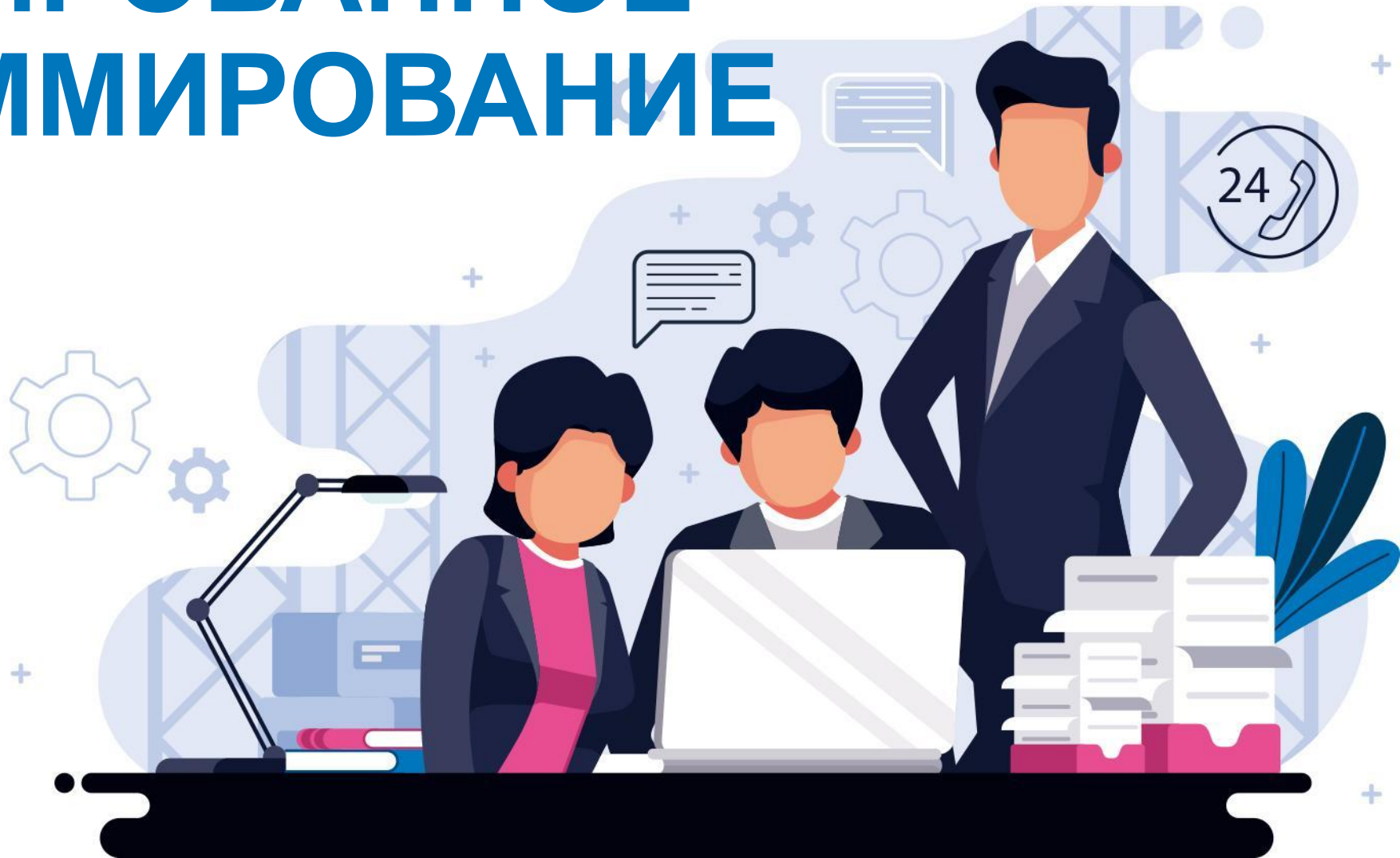
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лекция 1



План

- Spring
- JDK
- Среда разработки и создание проекта
- Inversion of Control
- Dependency Injection

Spring

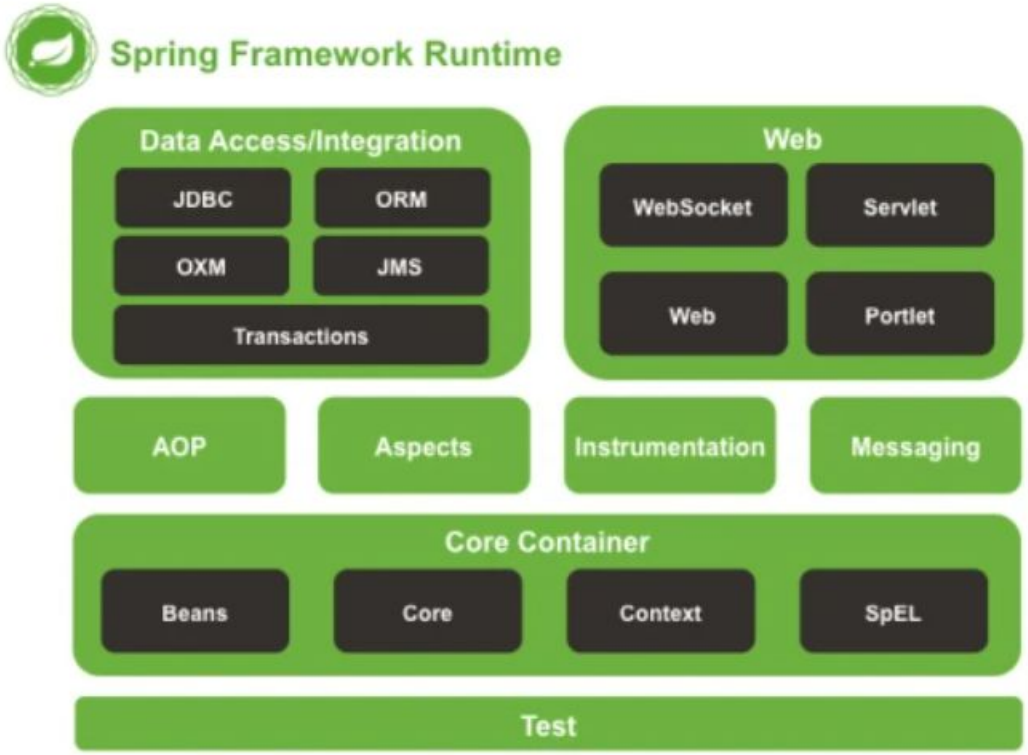
Spring – это фреймворк, предназначенный для более быстрого и простого построения Java приложений.

Spring

Spring предоставляет каркас вашего будущего приложения. При этом фреймворк диктует вам правила построения приложения – есть определенная архитектура приложения, в которую вам нужно встроить свою функциональность.

Эта функциональность собственно и будет бизнес логикой вашего приложения. В состав Spring входит много подпроектов, «заточенных» под определенную функциональность (SpringMVC, Spring Security, SpringData и др. полный список можно увидеть по адресу https://spring.io/projects), из которых разработчик может выбрать наиболее подходящий ему, а остальные не использовать – это модульный принцип построения приложения;

Spring



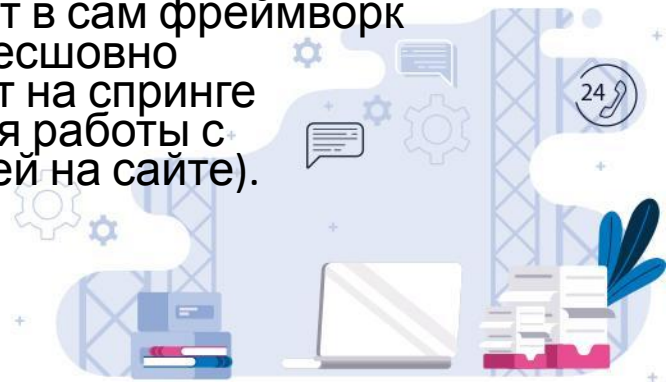
Как видно, у спринга модульная структура. Это позволяет подключать только те модули, что нам нужны для нашего приложения и не подключать те, которыми мы заведомо не будем пользоваться.

На изображении видно, что спринг фреймворк состоит как-бы из нескольких модулей:

- data access;
- web;
- core;
- и других.

Как можно было догадаться, модуль data access содержит в себе средства для работы с данными (в основном, с базами данных), web — для работы в сети.

Кроме того, есть еще так-называемая целая спринг-инфраструктура: множество других проектов, которые не входят в сам фреймворк официально, но при этом бесшовно интегрируются в ваш проект на спринге (например, spring security для работы с авторизацией пользователей на сайте).

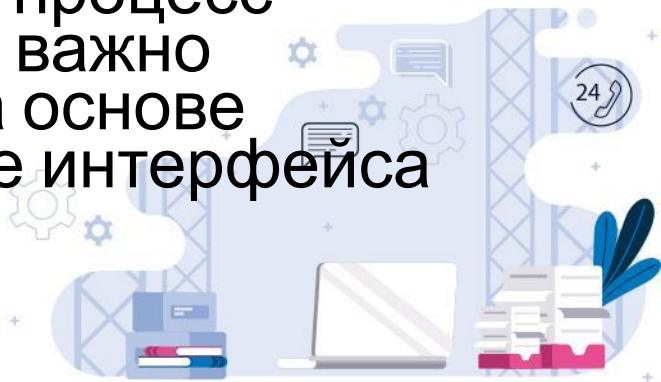


Spring

В приложении на основе Spring объекты слабосвязаны за счет использования внедрения зависимостей. Одной из целью создания Spring было разорвать зависимость одних объектов от других.

Что такое зависимость? Это когда объект Object1 использует методы другого объекта Object2, т.е объект Object1 зависит от объекта Object2, чьи методы он использует. А почему он зависит? А потому, что пока объект Object2 не создан, Object1 не сможет реализовать свою функциональность.

Как разорвать зависимость? В объект Object1 «внедрить» ссылку на объект Object2 через конструктор или сеттер. Этот процесс собственно и есть внедрение зависимости. При этом важно помнить, что в Spring объекты необходимо строить на основе интерфейсов, что бы зависимости внедряются в виде интерфейса для возможной последующей замены реализации.



Spring

Spring освобождает не только от необходимости создавать объекты, но и связывать их.

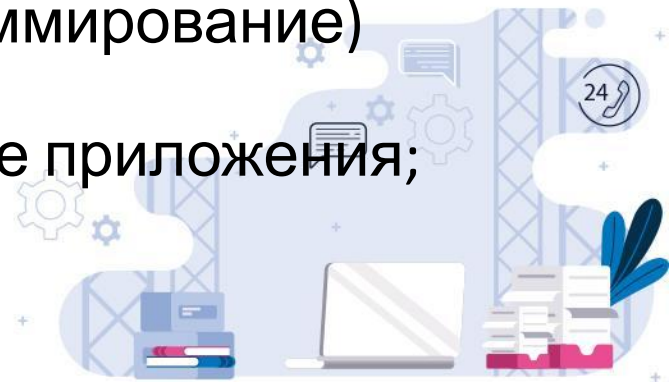
Например аннотация **@Autowired** позволяет автоматически связывать компоненты. Аннотацию спринга @Autowired можно было бы описать по-простому так - дорогой друг, контейнер спринг, посмотри пожалуйста в своей мапе с бинами, нет ли у тебя там класса instanceof or implements того, перед чем я стою. Если есть - дай мне ссылку в поле, перед которым я объявлена.

Автоматическое связывание позволяет уменьшить количество кода при определении зависимостей компонентов;

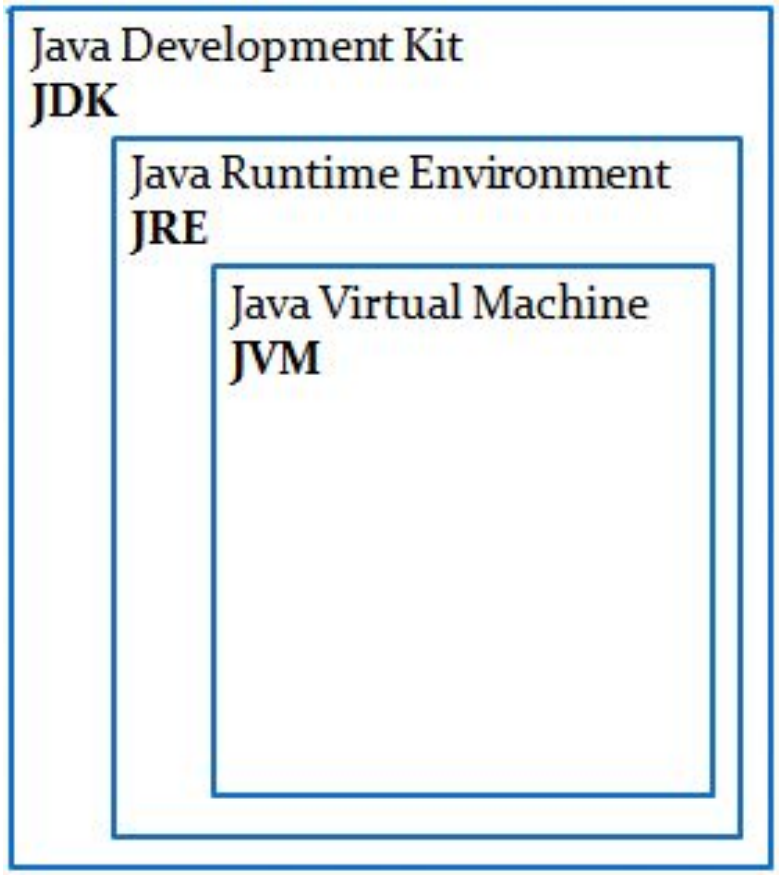
Spring

в Spring настройки компонентов отделены от программного кода. Вынесение конфигурации (управление зависимостями) в отдельный файл облегчает последующие изменения в проекте (замена реализаций):

- улучшенная возможность тестирования. Когда классы проектируются на основе DI (Dependency Injection) и интерфейсов, становится возможной простая замена зависимостей (фейковыми реализациями) при тестировании;
- возможность программирования в декларативном стиле с помощью аннотаций уменьшает количество кода в приложении;
- поддержка и хорошая интеграция с технологиями доступа к данным, транзакциями, AOP (Аспектно-ориентированное программирование) упрощает разработку;
- хорошее документирование очень помогает при отладке приложения;



JDK



Для разработки на языке программирования Java нам потребуется специальный комплект инструментов, который называется JDK или Java Development Kit.

JVM (Java Virtual Machine) - виртуальная машина Java - основная часть исполняющей системы Java, так называемой Java Runtime Environment (JRE). Виртуальная машина Java исполняет байт-код Java, предварительно созданный из исходного текста Java-программы компилятором Java (javac). JVM обеспечивает платформо-независимый способ выполнения кода. Программисты могут писать код не задумываясь как и где он будет выполняться.

JRE (Java Runtime Environment) - минимальная реализация виртуальной машины, необходимая для исполнения Java -приложений, без компилятора и других средств разработки. Состоит из виртуальной машины и библиотек Java классов.

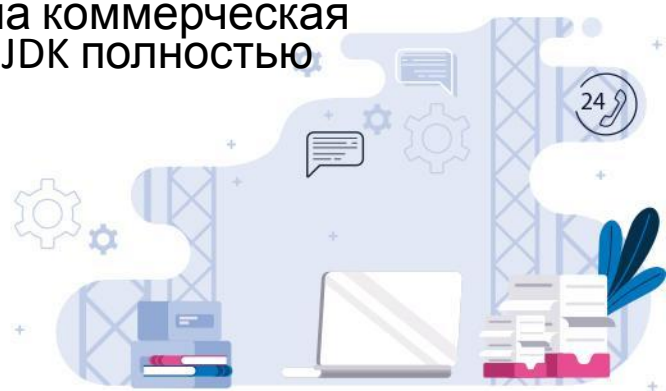
JDK (Java Development Kit) - комплект разработчика приложений на языке Java, включающий в себя компилятор, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему JRE

JDK

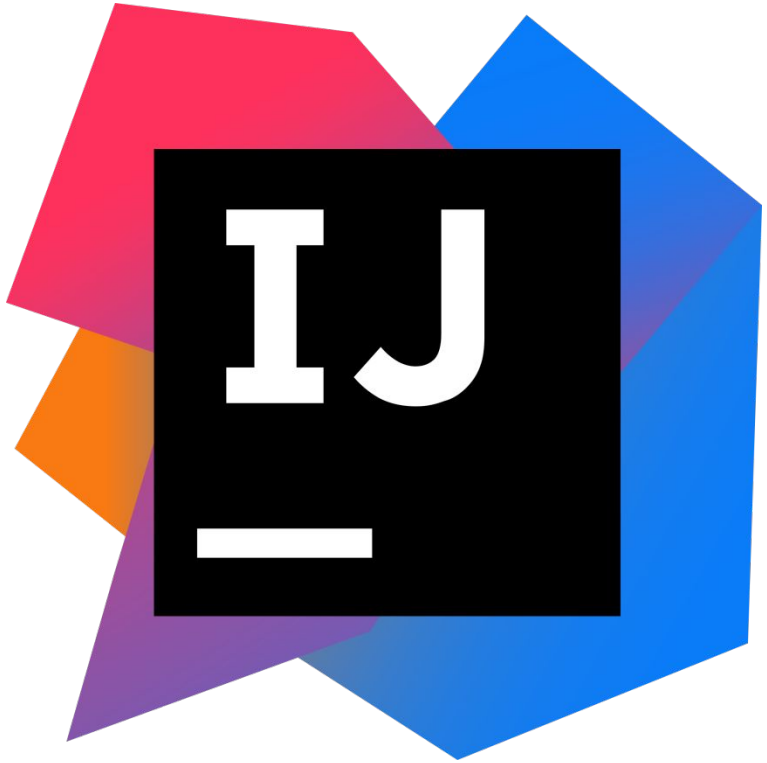
Однако стоит отметить, что существуют разные реализации JDK, хотя все они используют один и тот же язык - Java. Две наиболее популярных реализации - Oracle JDK и OpenJDK. В чем их разница?

Oracle JDK всецело развивается компанией Oracle. OpenJDK же развивается как компанией Oracle, так и еще рядом компаний совместно.

Наибольшие различия с точки зрения лицензирования. Согласно лицензии Oracle JDK можно использовать бесплатно для персональных нужд, а также для разработки, тестирования и демонстрации приложений. В остальных случаях (например, для получения поддержки) необходима коммерческая лицензия в виде подписки. А OpenJDK полностью бесплатна.



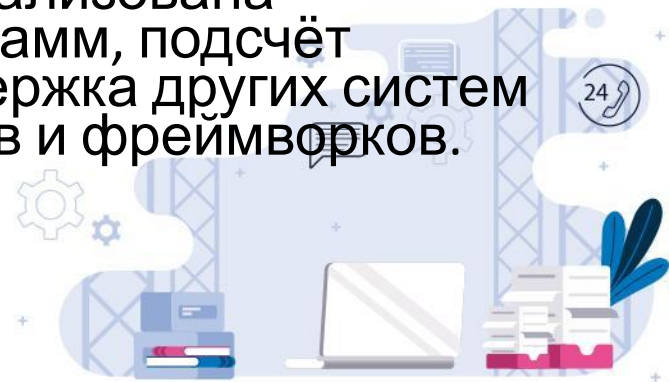
Среда разработки



IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Начиная с версии 9.0, среда доступна в двух редакциях: Community Edition и Ultimate Edition.

Community Edition является полностью свободной версией, доступной под лицензией Apache 2.0, в ней реализована полная поддержка Java SE, Kotlin, Groovy, Scala, а также интеграция с наиболее популярными системами управления версиями. В редакции **Ultimate Edition**, доступной под коммерческой лицензией, реализована поддержка Java EE, UML-диаграмм, подсчёт покрытия кода, а также поддержка других систем управления версиями, языков и фреймворков.



Среда разработки

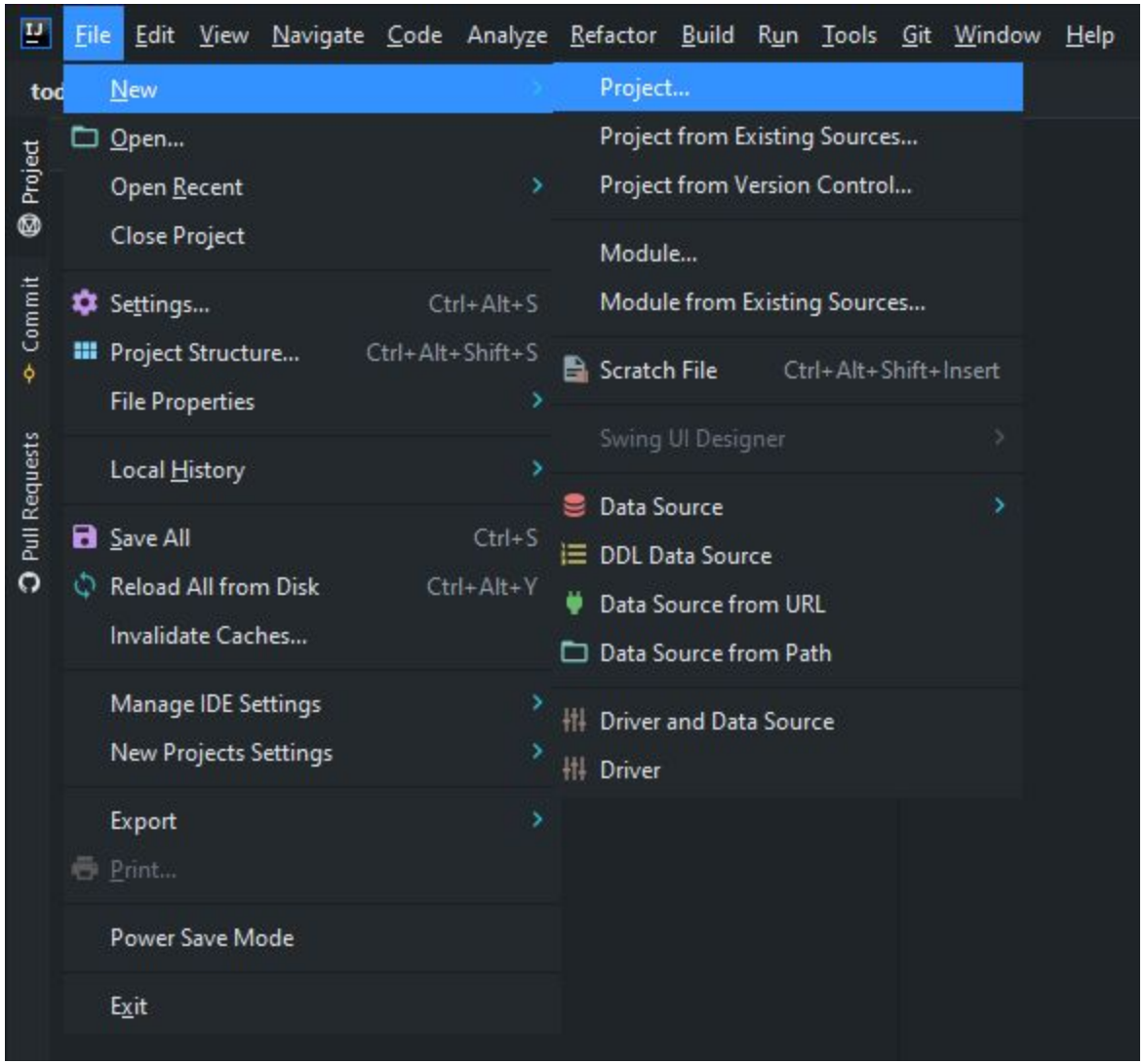
NetBeans IDE — свободная интегрированная среда разработки приложений (IDE) на языках программирования Java, Python, PHP, JavaScript, C, C++, Ада и ряда других.

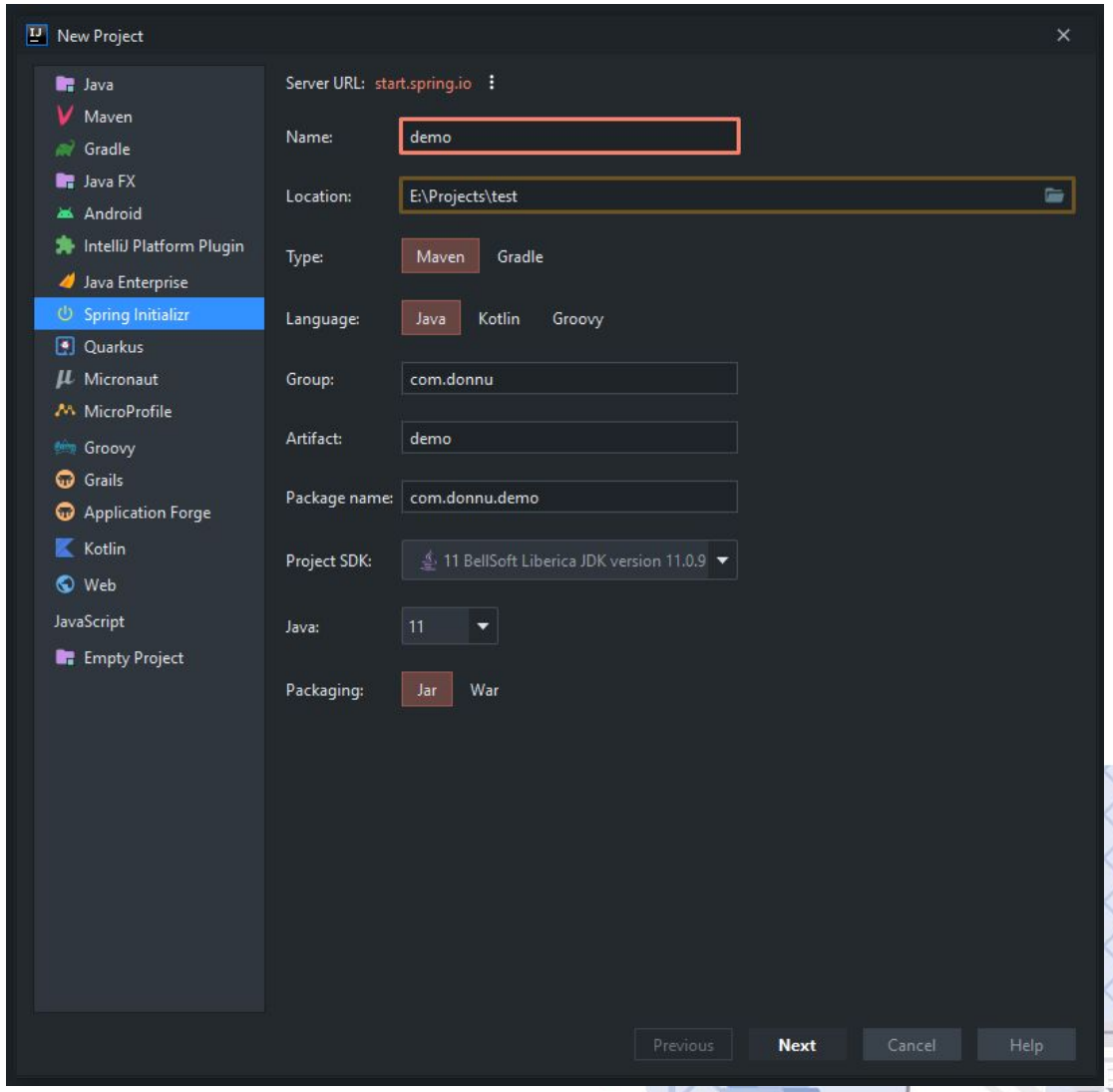
Проект NetBeans IDE поддерживается и спонсируется компанией Oracle, однако разработка NetBeans ведётся независимым сообществом разработчиков-энтузиастов (NetBeans Community) и компанией NetBeans Org.

Последние версии NetBeans IDE поддерживают рефакторинг, профилирование, выделение синтаксических конструкций цветом, автодополнение набираемых конструкций на лету и множество предопределённых шаблонов кода.

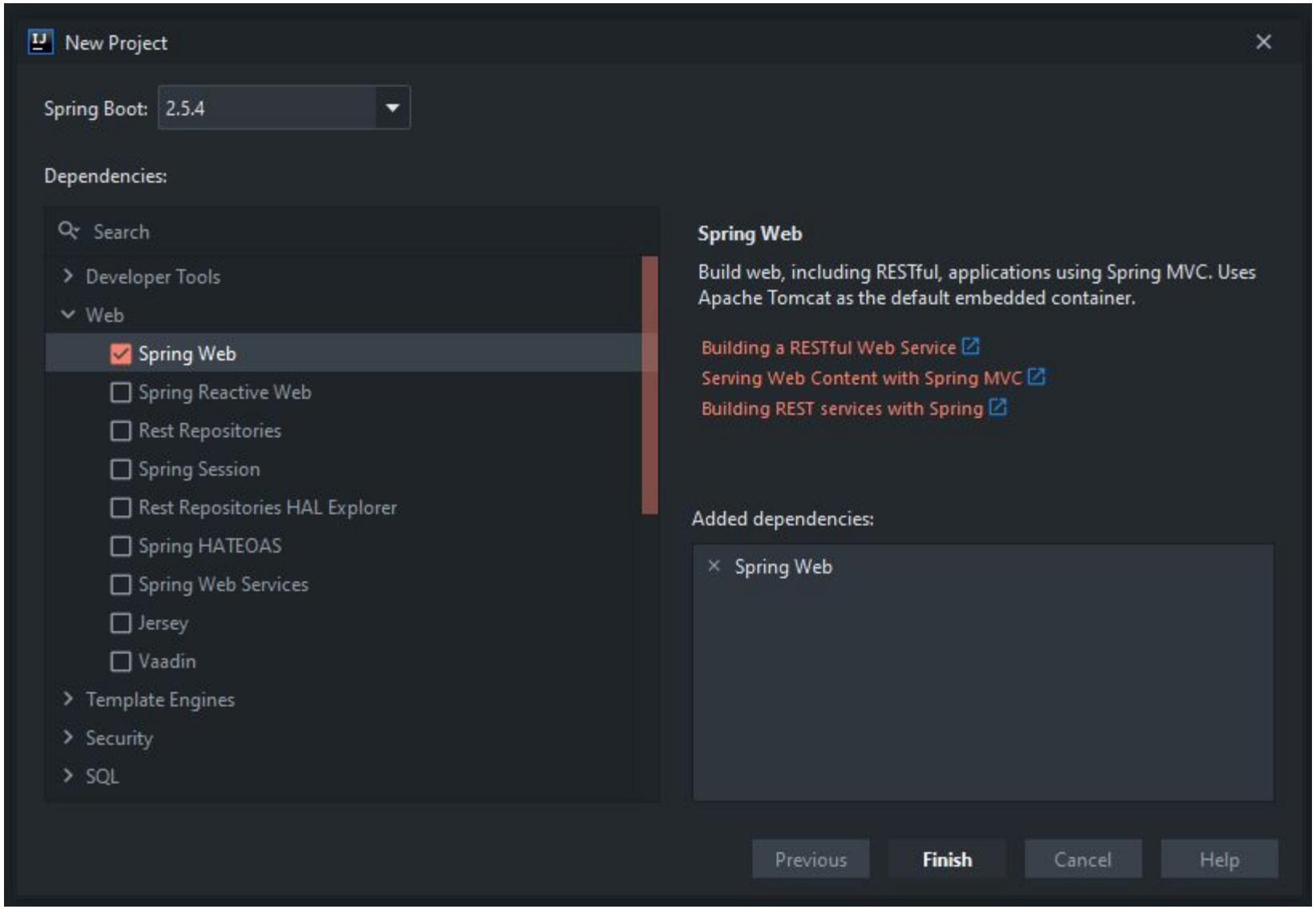
Для разработки программ в среде NetBeans и для успешной инсталляции и работы самой среды NetBeans должен быть предварительно установлен Sun JDK или J2EE SDK подходящей версии. Среда разработки NetBeans по-умолчанию поддерживала разработку для платформ J2SE и J2EE. Начиная с версии 6.0 NetBeans поддерживает разработку для мобильных платформ J2ME, C++ (только g++) и PHP без установки дополнительных компонентов.

Создание проекта в IntelliJ IDEA

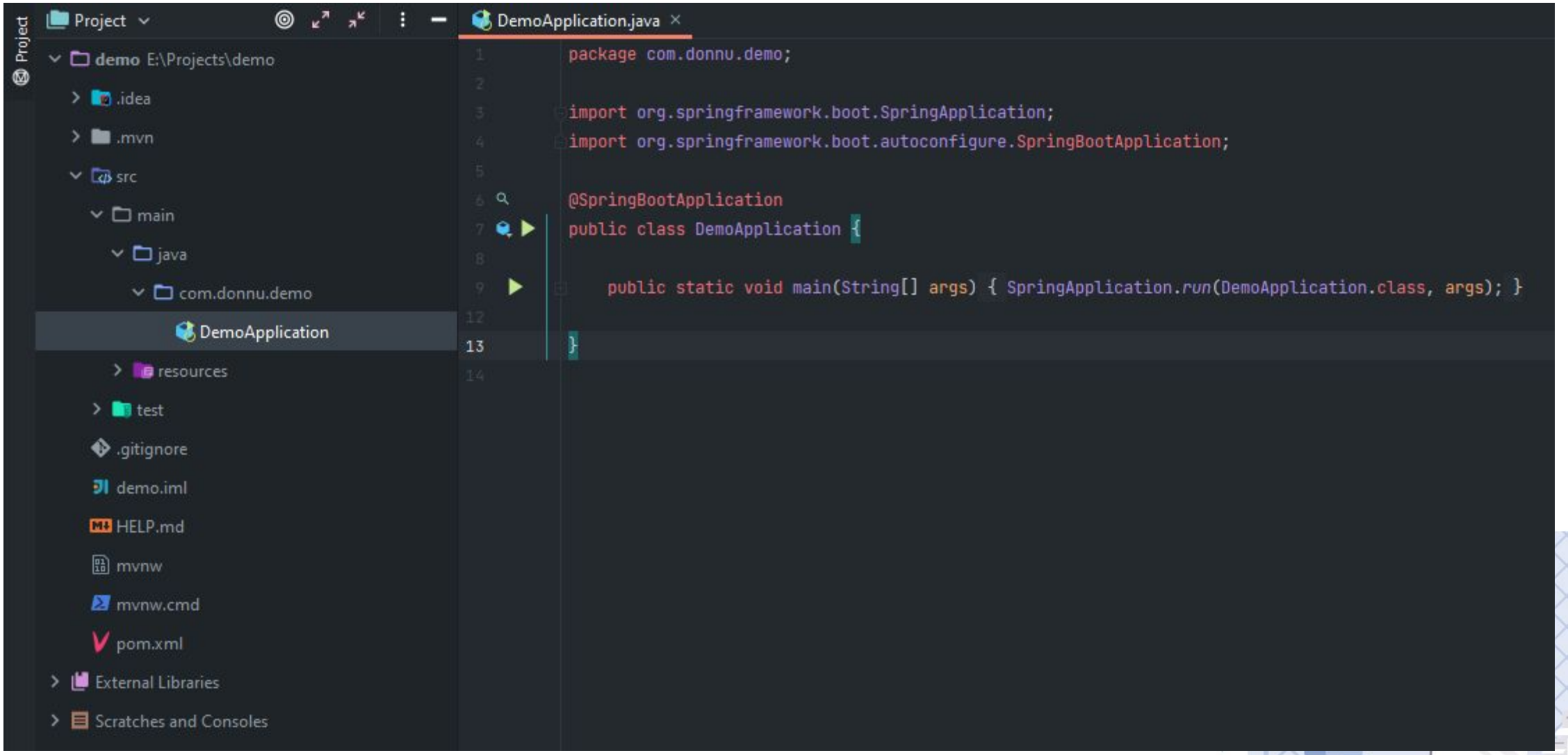




Создание проекта в IntelliJ IDEA



Создание проекта в IntelliJ IDEA



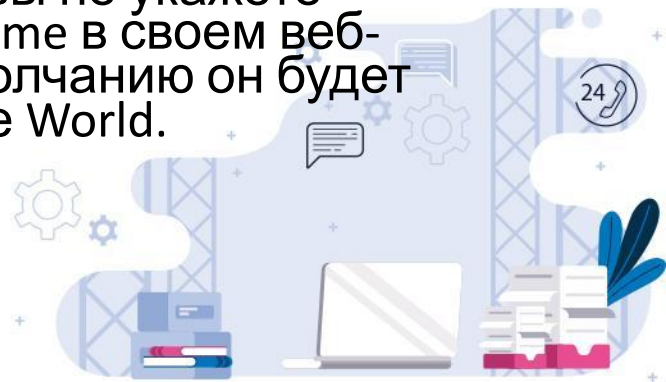
Создание проекта в IntelliJ IDEA

DemoApplication.java

```java
package com.donnu.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class DemoApplication {

    public static void main(String[] args) { SpringApplication.run(DemoApplication.class, args); }

    @GetMapping("/hello")
    public String sayHello(@RequestParam(value = "myName", defaultValue = "World") String name) {
        return String.format("Hello %s!", name);
    }

}
```

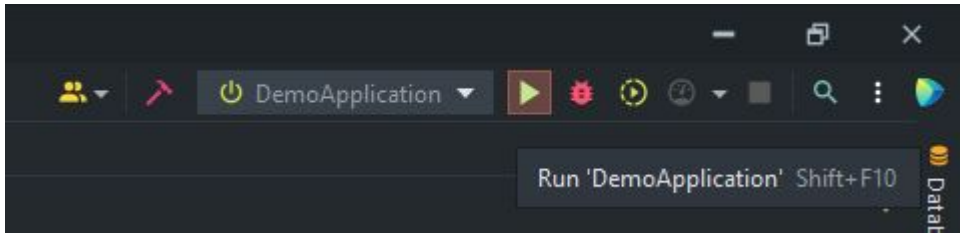
Метод **sayHello()** принимает параметр имени и возвращает слово Hello в сочетании со значением параметра. Все остальное обрабатывается добавлением аннотаций Spring:

- Аннотация **@RestController** отмечает класс DemoApplication как обработчик запроса (контроллер REST).
- Аннотация **@GetMapping** ("/ hello") отображает метод sayHello () на запросы GET для / hello.
- Аннотация **@RequestParam** сопоставляет параметр метода имени с параметром веб-запроса myName. Если вы не укажете параметр myName в своем веб-запросе, по умолчанию он будет иметь значение World.

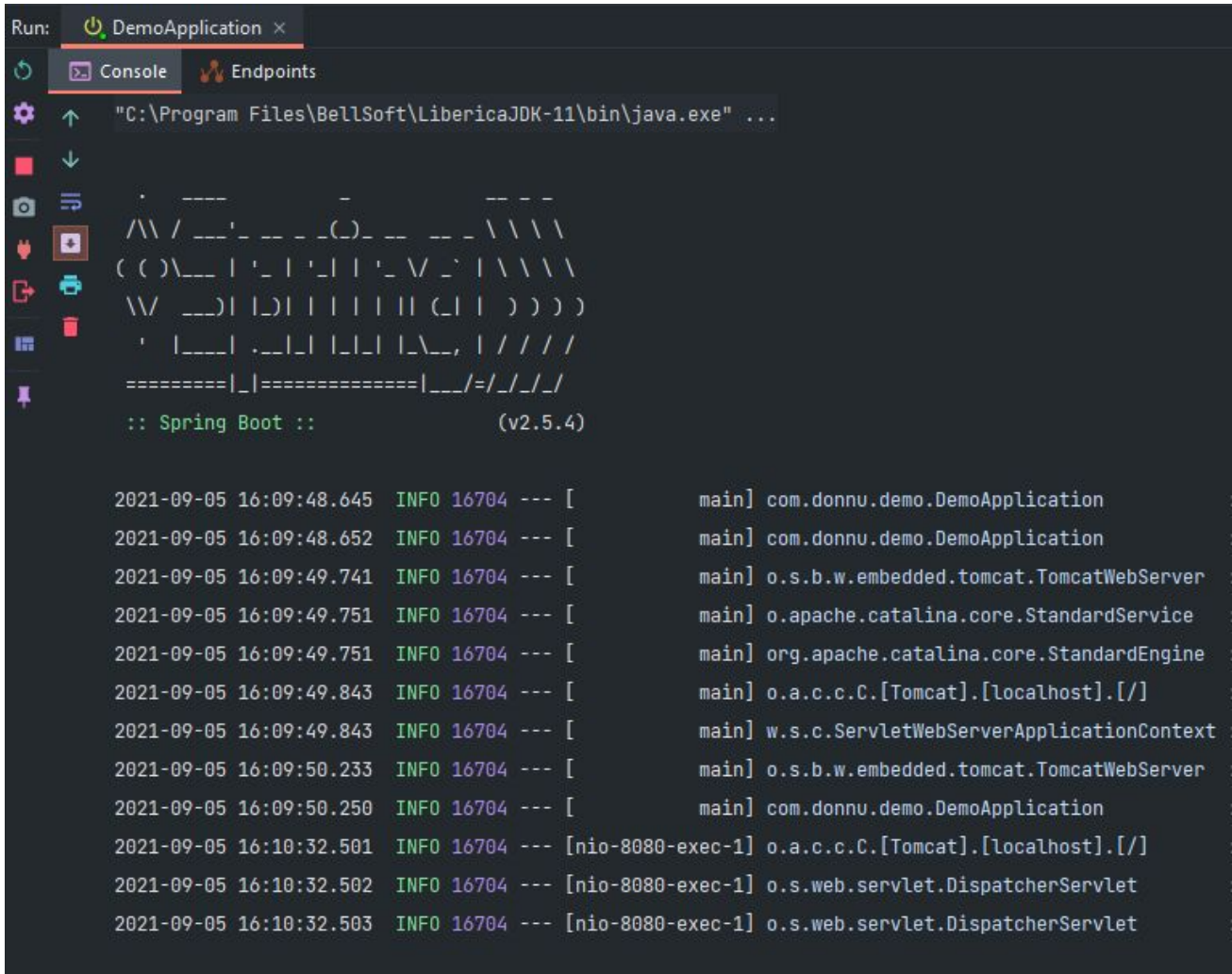


Создание проекта в IntelliJ IDEA

Запускаем приложение

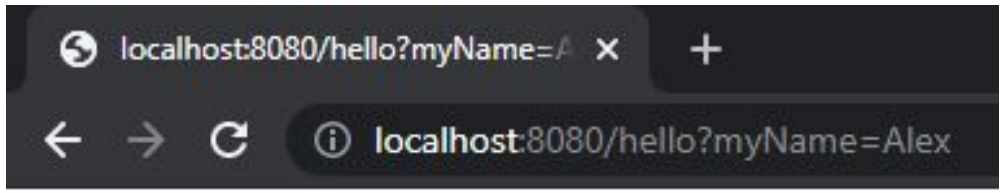


Вывод в терминал

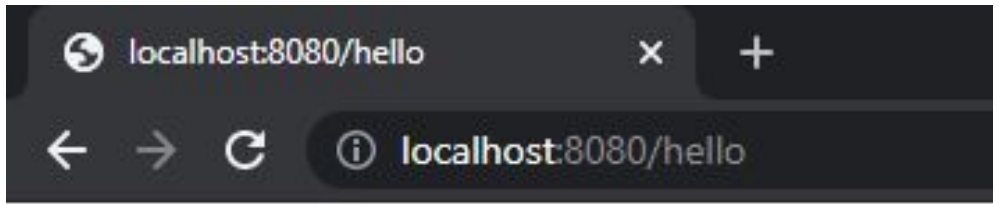


Создание проекта в IntelliJ IDEA

Получаем ответ от сервера



Hello Alex!



Hello World!

Создание проекта start.spring.io

spring initializr

Project
- Maven Project
- Gradle Project

Language
- Java
- Kotlin
- Groovy

Spring Boot
- 2.6.0 (SNAPSHOT)
- 2.6.0 (M2)
- 2.5.5 (SNAPSHOT)
- 2.5.4
- 2.4.11 (SNAPSHOT)
- 2.4.10

Project Metadata

Group	com.demo
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.demo.demo
Packaging	Jar / War
Java	16 / 11 / 8

Dependencies — ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...

Заходим на сайт

https://start.spring.io/

Прописываем аналогичные характеристики проекта и скачиваем архив

Создание проекта start.spring.io

spring initializr

Project
- Maven Project
- Gradle Project

Language
- Java
- Kotlin
- Groovy

Spring Boot
- 2.6.0 (SNAPSHOT)
- 2.6.0 (M2)
- 2.5.5 (SNAPSHOT)
- 2.5.4
- 2.4.11 (SNAPSHOT)
- 2.4.10

Project Metadata

Group	com.demo
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.demo.demo
Packaging	Jar / War
Java	16 / 11 / 8

Dependencies — ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...

Заходим на сайт

https://start.spring.io/

Прописываем аналогичные характеристики проекта и скачиваем архив

Создание проекта start.spring.io

spring initializr

Project
- Maven Project
- Gradle Project

Language
- Java
- Kotlin
- Groovy

Spring Boot
- 2.6.0 (SNAPSHOT)
- 2.6.0 (M2)
- 2.5.5 (SNAPSHOT)
- 2.5.4
- 2.4.11 (SNAPSHOT)
- 2.4.10

Project Metadata

Group	com.demo
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.demo.demo
Packaging	Jar / War
Java	16 / 11 / 8

Dependencies — ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...

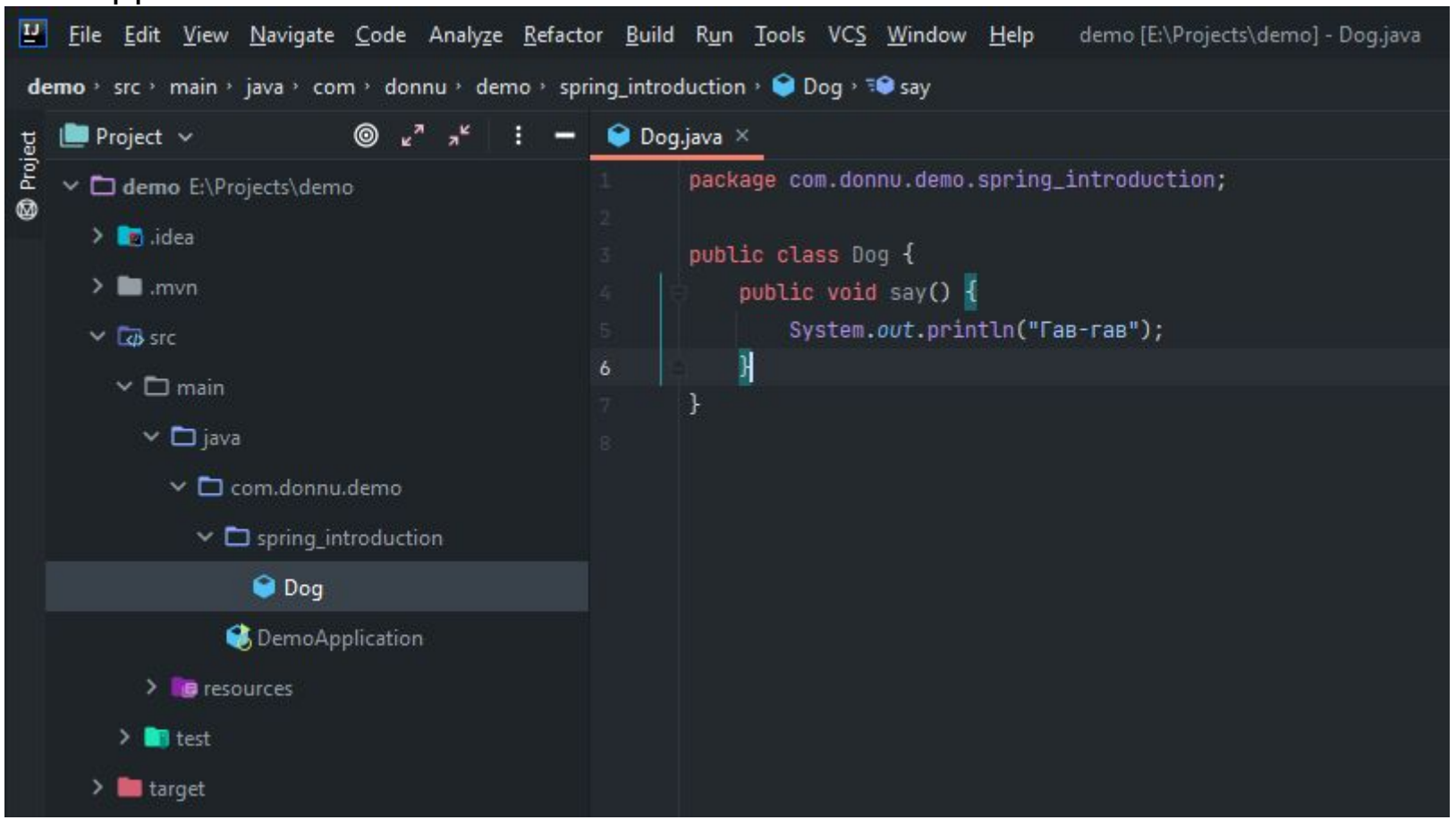
Заходим на сайт

https://start.spring.io/

Прописываем аналогичные характеристики проекта и скачиваем архив

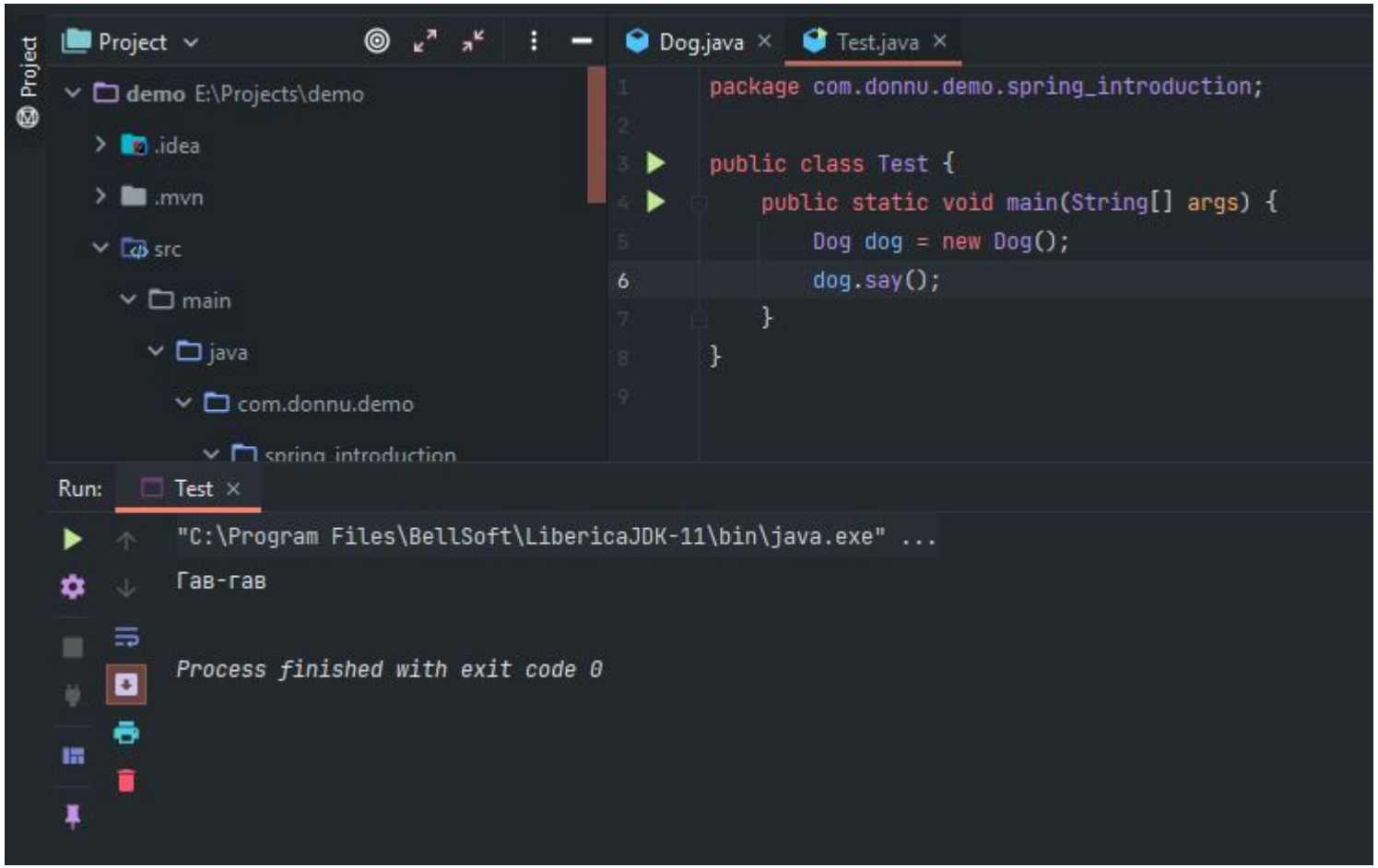
Inversion of Control

Рассмотрим простейший пример. Создадим примитивный класс с единственным методом.



Inversion of Control

Создаем класс с функцией main и проверяем работоспособность ранее созданного класса.



Inversion of Control

```java
package com.donnu.demo.spring_introduction;

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

```java
package com.donnu.demo.spring_introduction;

public class Dog implements Pet {
    @Override
    public void say() {
        System.out.println("Гав-гав");
    }
}
```

Предположим, что мы хотим, чтобы не только собаки, но и другие животные издавали звуки. Чтобы реализовать это грамотно используем интерфейс Pet.

Далее возвращаемся в класс Dog и имплементируем интерфейс Pet.

Inversion of Control

```java
package com.donnu.demo.spring_introduction;

public class Test {
    public static void main(String[] args) {
        Pet pet = new Dog();
        pet.say();
    }
}
```

```
Test ×
"C:\Program Files\BellSoft\LibericaJDK-11\bin\java.exe" ...
Гав-гав

Process finished with exit code 0
```

Заменяем тип в классе Test и получаем ожидаемый результат. Все по-прежнему работает.

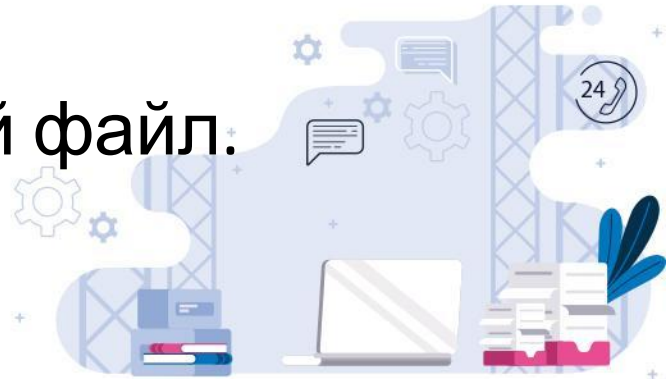
Inversion of Control

```java
package com.donnu.demo.spring_introduction;

public class Cat implements Pet{
    @Override
    public void say() {
        System.out.println("Мяу-мяу");
    }
}
```

Создадим аналогичный класс Cat. И выведем его в классе Test.

Чтобы каждый раз не менять код, чтобы услышать очередное животное, а соответственно, чтобы не было нужды в перекомпиляции, самым лучшим решением будет вынести эти детали, например, какое животное будет говорить в отдельный файл.

Конфигурационный файл.



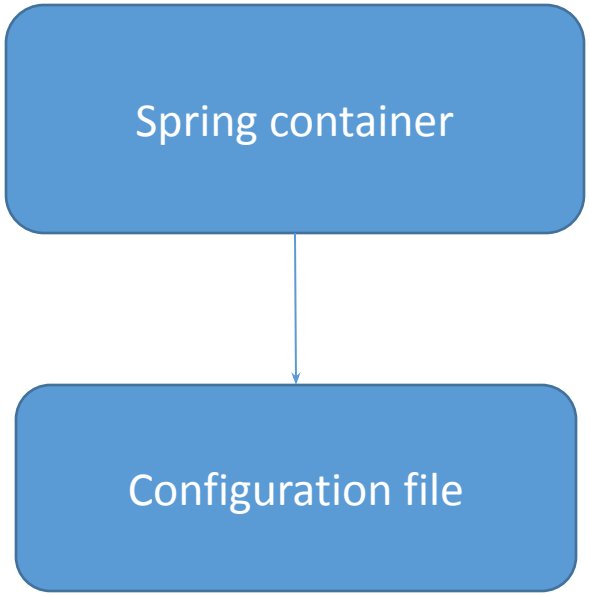
Inversion of Control

Spring container

Как это будет работать в Spring?

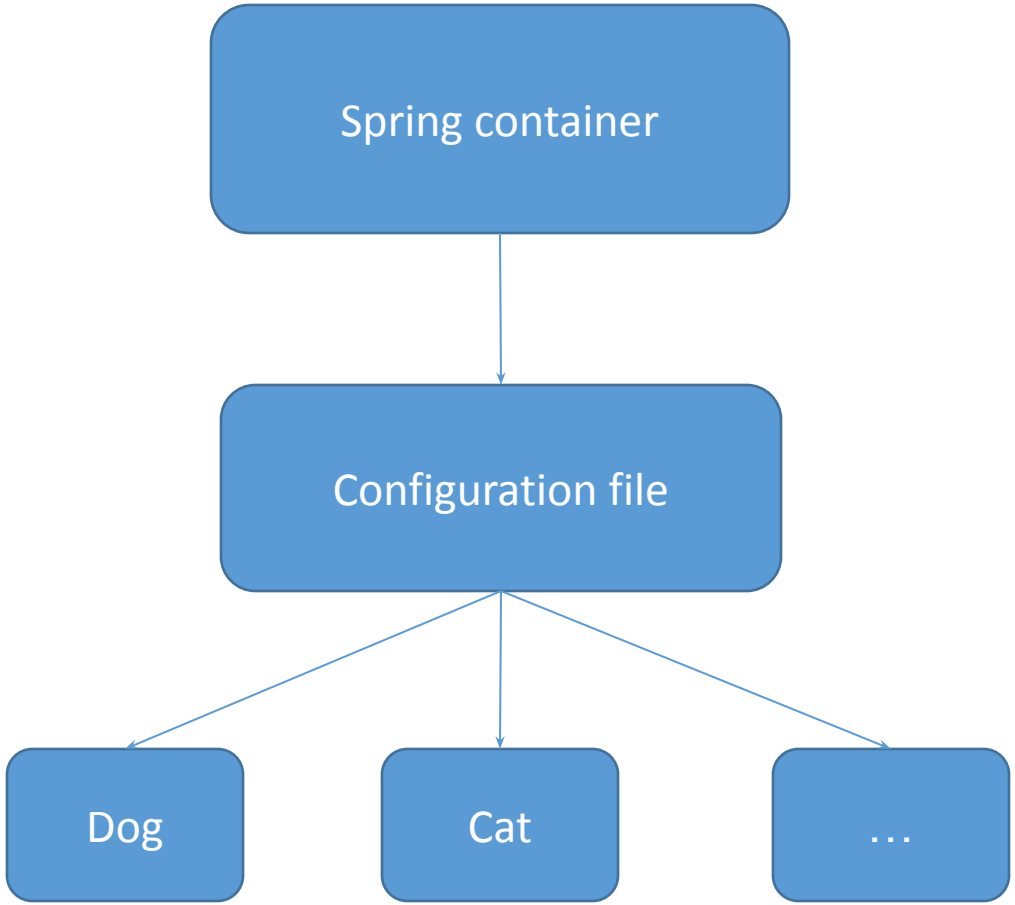
Ответственным за создание и управление объектом является Spring Container. Объекты создаются в контейнере, а мы, когда нам нужно, их получаем.

Inversion of Control



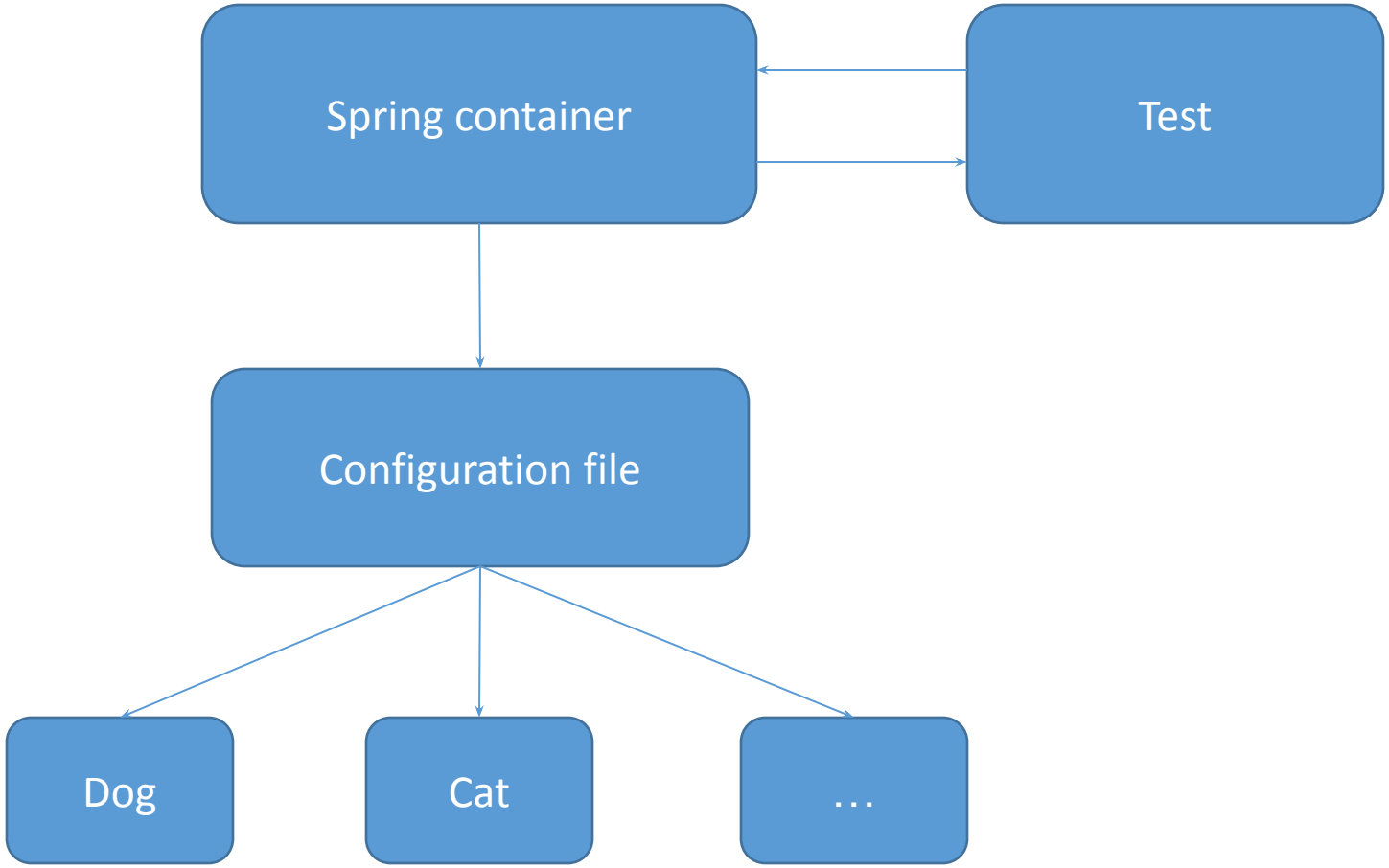
Контейнер будет читать наш конфигурационный файл, и в контейнере будет создаваться тот объект(ы), которые мы опишем.

Inversion of Control



Таким образом в конфиг. файле мы описываем наш объект, например: Dog, Cat и т.д. Spring container их создает, содержит.

Inversion of Control



И когда такой объект понадобится классу Test, он получит его напрямую из контейнера.

В классе Test объекты не создаются!

Inversion of Control

Основные функции, которые выполняет Spring Container:

- IoC – инверсия управления, создание и управление объектами
- DI – Dependency Injection (Внедрение зависимостей)

IoC – аутсорсинг создания и управления объектами, т.е. передача прав на создание и управление Spring-у

Inversion of Control

<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

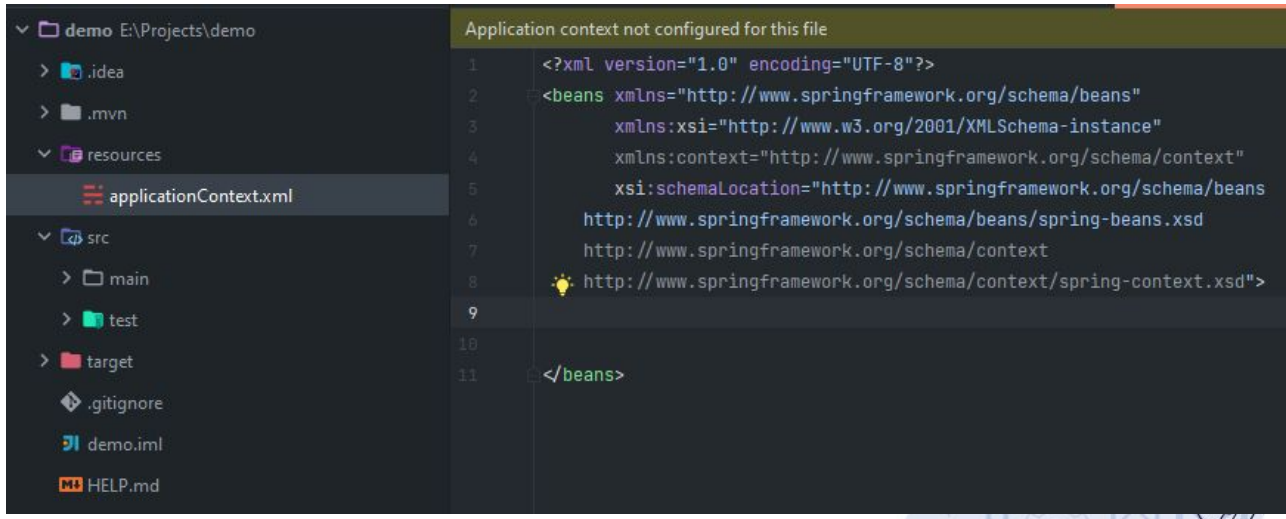
xmlns:context="http://www.springframework.org/schema/context"

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">

Создадим папку resources файл applicationContext со следующим содержимым:



Inversion of Control

Конфигурация XML файла:

```
<bean id=“идентификатор бина”
      class = “полное название класса”>
</bean>
```

Inversion of Control

```xml
<?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">

    <bean id="myPet"
          class="com.donnu.demo.spring_introduction.Dog">

    </bean>

</beans>
```

Inversion of Control

Spring bean (или просто bean) – это объект, который создается и управляется Spring Container.

ApplicationContext представляет собой **Spring Container**. Поэтому для получения бина из Spring Container нам нужно создать ApplicationContext.

Inversion of Control

Создаем контекст и получаем их него объект

```java
package com.donnu.demo.spring_introduction;

import org.springframework.context.support.ClassPathXmlApplicationContext;

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

        Pet pet = context.getBean( name: "myPet", Pet.class);
        pet.say();

        context.close();
    }
}
```

Inversion of Control

Запускаем

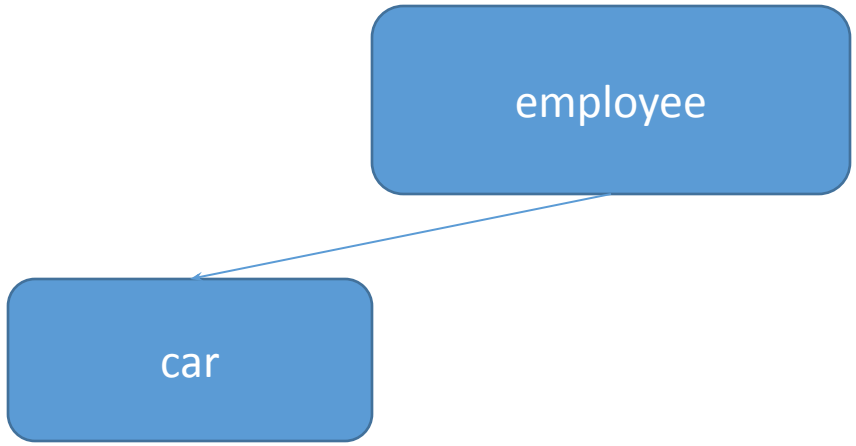
```
Test ×
"C:\Program Files\BellSoft\LibericaJDK-11\bin\java.exe" ...
21:50:32.599 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@401e7803
21:50:32.918 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 1 bean definitions from class path resource [applicationContext.xml]
21:50:32.960 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'myPet'
Гав-гав
21:50:33.176 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.context.support.ClassPathXmlApplicationContext@401e7803, start

Process finished with exit code 0
```

Inversion of Control

Стоит отметить, что если мы захотим поменять класс Dog на Cat и вызвать его, перекомпиляция нам не потребуется, также как и изменение класса Test. Необходимо будет изменить только конфигурацию.

Dependency Injection

Начнем с простого примера. Допустим, у нас есть объект employee (работник), а у него есть машина, объект car.

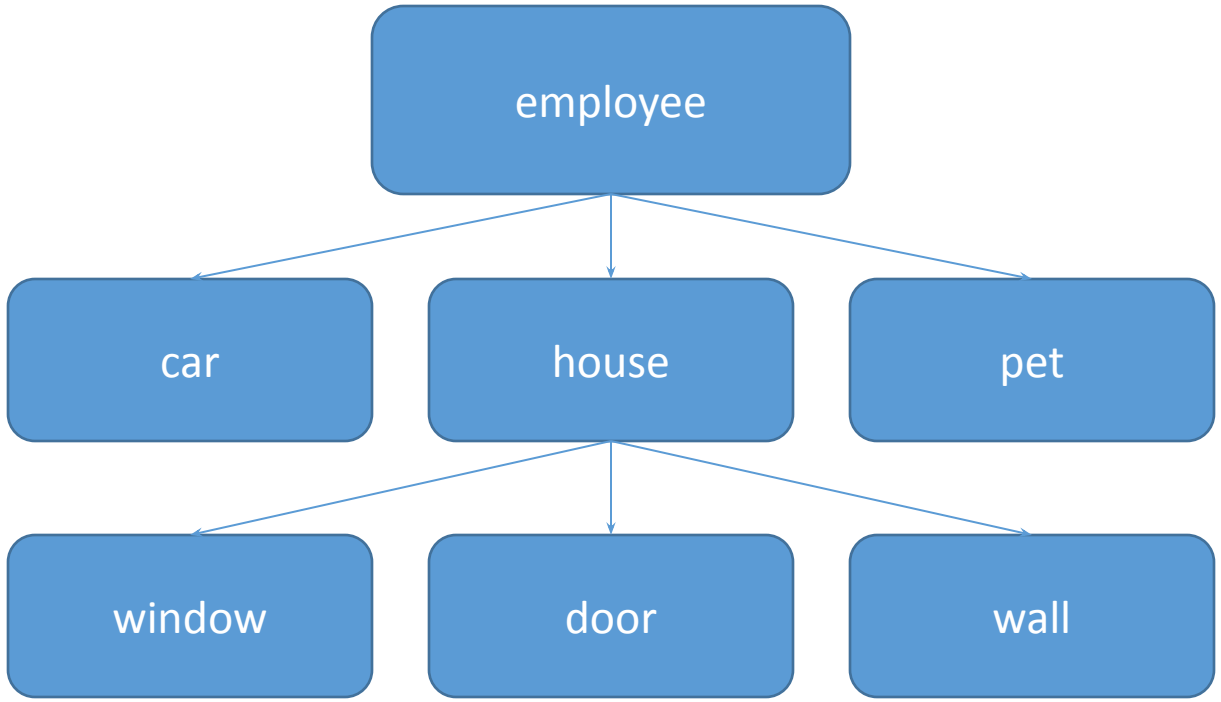


Dependency Injection

То есть при создании объекта employee, он имеет ссылку на объект car. Это называется зависимостью. Теперь представим, что у нас сложное приложение со множеством зависимостей, которые мы создаем вручную.

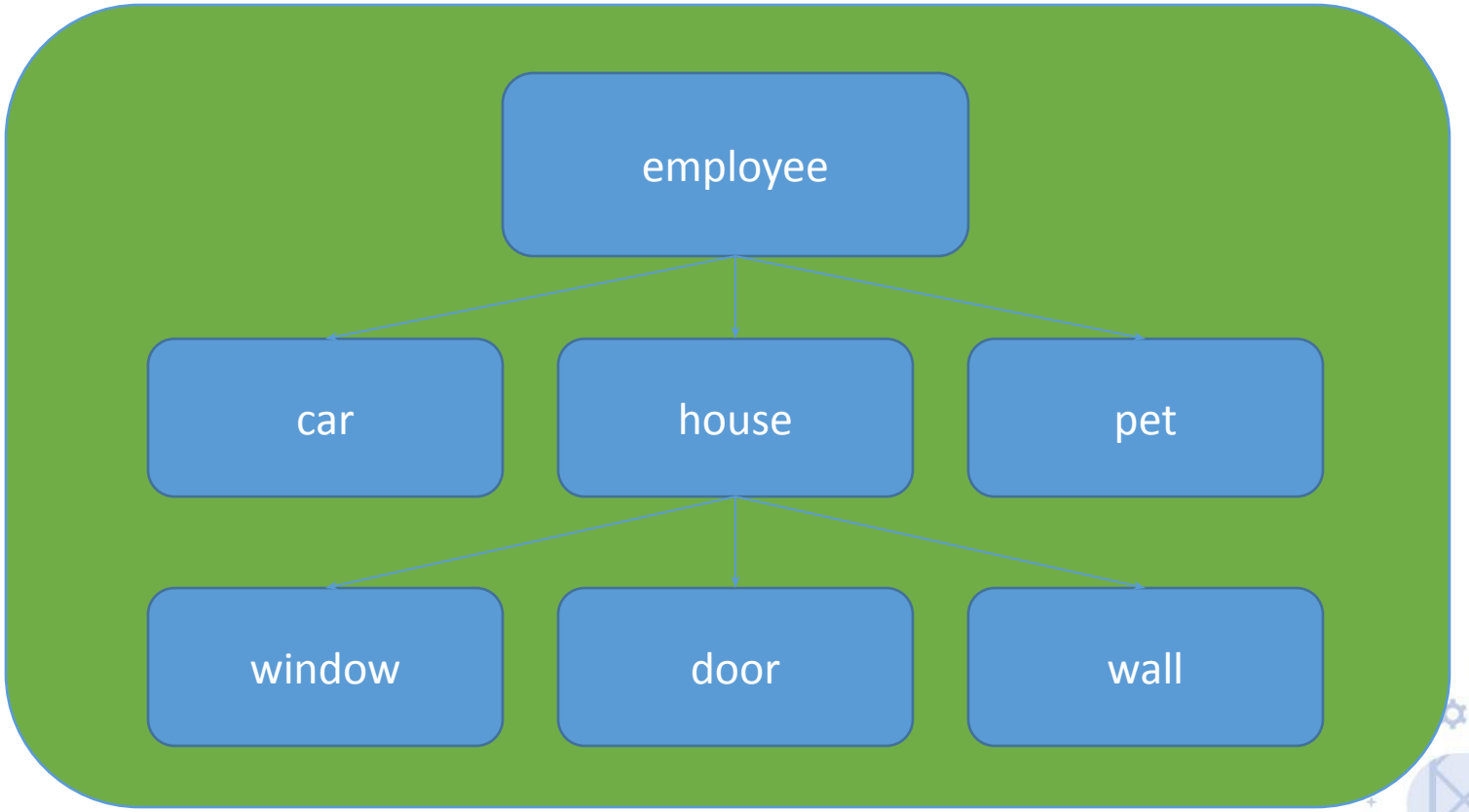
Dependency Injection

Например, у нашего employee есть еще ряд зависимостей. Получается, что для создания employee необходимо создать все эти объекты. Легко запутаться и много лишнего кода.



Dependency Injection

Вместо этого можно все описать в конфиг. файле, и потом в Spring Container создаются описанные объекты, и Spring без нашего участия внедряет эти зависимости.



Dependency Injection

```java
package com.donnu.demo.spring_introduction;

public class Person {
    private Pet pet;

    public Person(Pet pet) {
        this.pet = pet;
    }

    public void callYourPet() {
        System.out.println("Hey!");
        pet.say();
    }
}
```

Для демонстрации создадим класс Person. Это будет хозяин домашнего животного. У него будет единственный метод – позвать животное, при котором оно будет откликаться.

Dependency Injection

```java
package com.donnu.demo.spring_introduction;

public class Test2 {
    public static void main(String[] args) {
        Pet pet = new Dog();

        Person person = new Person(pet);

        person.callYourPet();
    }
}
```

Без применения Spring работа с этим классом выглядела бы следующим образом.

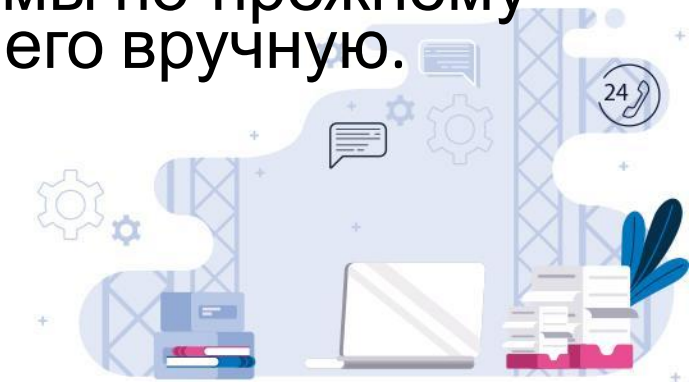
Dependency Injection

```java
package com.donnu.demo.spring_introduction;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test2 {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext context =
                new ClassPathXmlApplicationContext( configLocation: "applicationContext.xml");

        //Pet pet = new Dog();
        Pet pet = context.getBean( name: "myPet", Pet.class);

        Person person = new Person(pet);

        person.callYourPet();

        context.close();
    }
}
```

Модифицируем код с применением уже имеющихся знаний о Spring Container и созданного bean.

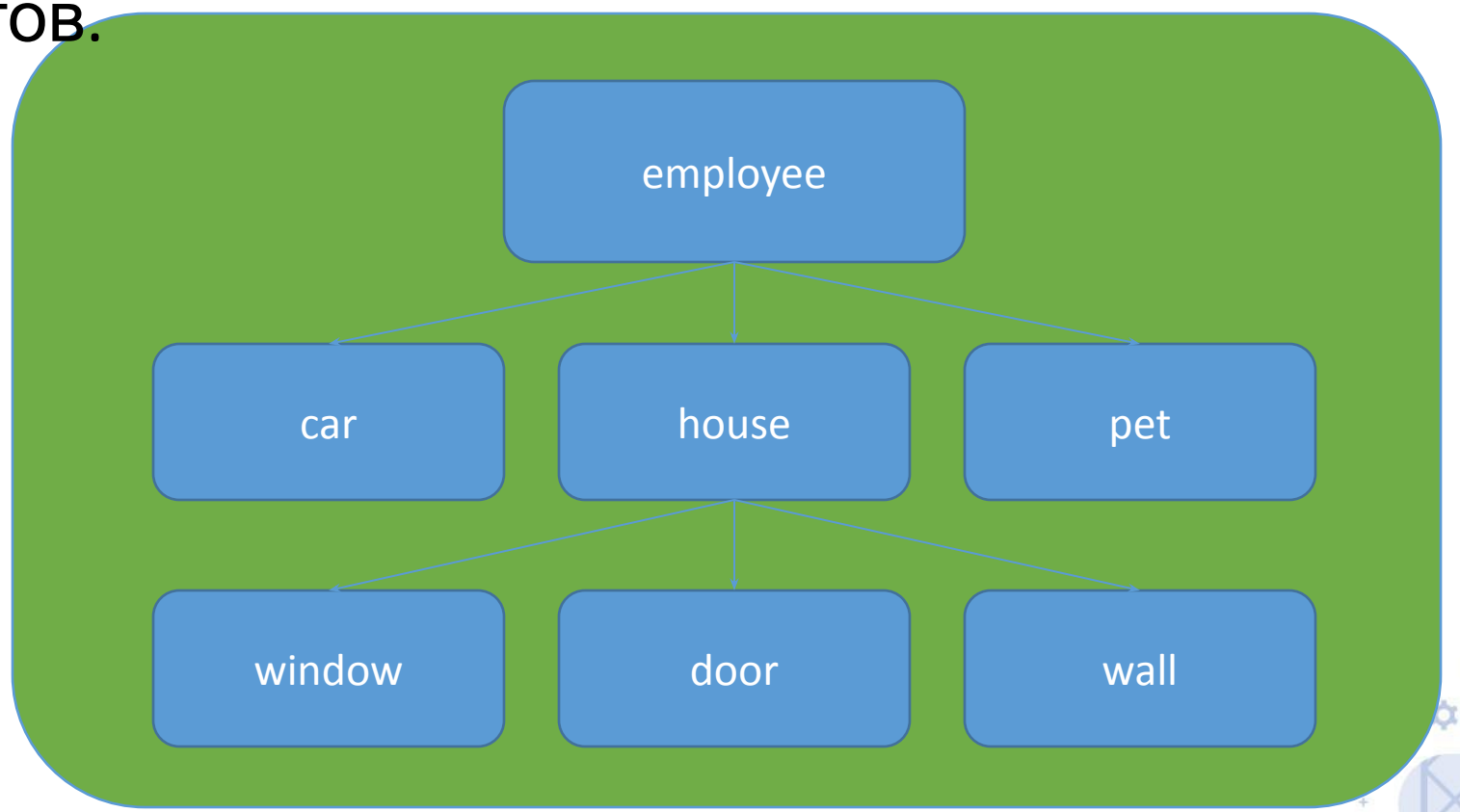
Обратите внимание, что для изменения животного опять достаточно поменять конфиг. файл.

Но проблема с person не решена, мы по-прежнему создаем его вручную.



Dependency Injection

Вернемся к упомянутой выше концепции. На рисунке показана сборка объекта employee из составляющих его элементов.



Dependency Injection

Dependency Injection – это аутсорсинг добавления/внедрения зависимостей. DI делает объекты нашего приложения слабо зависимыми друг от друга.

Способы внедрения зависимостей:

- с помощью конструктора
- с помощью сеттеров
- Autowiring

Dependency Injection

DI с помощью конструктора. Добавим bean myPerson. Внутри него создадим тег constructor-arg и передадим ссылку на уже созданный ранее bean myPet.

```xml
<?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">

    <bean id="myPet"
          class="com.donnu.demo.spring_introduction.Dog">
    </bean>

    <bean id="myPerson"
          class="com.donnu.demo.spring_introduction.Person">
        <constructor-arg ref="myPet"/>
    </bean>

</beans>
```

Dependency Injection

```java
package com.donnu.demo.spring_introduction;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test2 {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext context =
                new ClassPathXmlApplicationContext( configLocation: "applicationContext.xml");

        //Pet pet = new Dog();
        //Pet pet = context.getBean("myPet", Pet.class);

        Person person = context.getBean( name: "myPerson", Person.class);

        person.callYourPet();

        context.close();
    }
}
```

Теперь нам достаточно получить bean myPerson. В bean уже будет внедрена зависимость.

С объектом Pet мы в коде уже не сталкиваемся.

Dependency Injection

DI с помощью сеттера. Добавим сеттер в класс Person, создадим конструктор без параметров и закомментируем старый конструктор.

```java
package com.donnu.demo.spring_introduction;

public class Person {
    private Pet pet;

//    public Person(Pet pet) {
//        this.pet = pet;
//    }

    public void setPet(Pet pet) { this.pet = pet; }

    public void callYourPet() {
        System.out.println("Hey!");
        pet.say();
    }

    public Person() {
    }
}
```

Dependency Injection

Видоизменим bean.

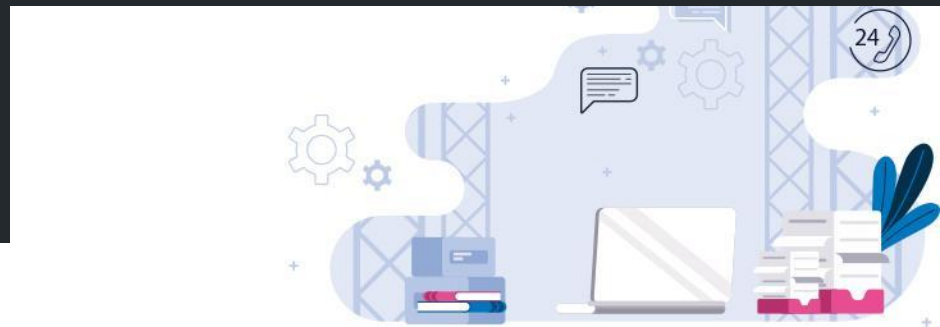
```
<bean id="myPerson"
      class="com.donnu.demo.spring_introduction.Person">

    <!--        <constructor-arg ref="myPet"/>  -->
    <!--        pet => setPet()                 -->
    <property name="pet" ref="myPet"/>
</bean>
```

Dependency Injection

Запускаем класс Test2. Обратите внимание, что в нем не изменилось абсолютно **ничего**. Вывод корректен.

```java
package com.donnu.demo.spring_introduction;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test2 {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext context =
                new ClassPathXmlApplicationContext( configLocation: "applicationContext.xml");

        //Pet pet = new Dog();
        //Pet pet = context.getBean("myPet", Pet.class);

        Person person = context.getBean( name: "myPerson", Person.class);

        person.callYourPet();

        context.close();
    }
```

```
Test2 ×
"C:\Program Files\BellSoft\LibericaJDK-11\bin\java.exe" ...
23:30:05.065 [main] DEBUG org.springframework.context.support
23:30:05.501 [main] DEBUG org.springframework.beans.factory.x
23:30:05.554 [main] DEBUG org.springframework.beans.factory.s
23:30:05.576 [main] DEBUG org.springframework.beans.factory.s
Hey!
Гав-гав
23:30:05.846 [main] DEBUG org.springframework.context.support

Process finished with exit code 0
```



Dependency Injection

Внедрение строк и других значений. Добавим имя и возраст в наш класс Person. И редактируем bean.

```
<bean id="myPerson"
      class="com.donnu.demo.spring_introduction.Person">

    <!--        <constructor-arg ref="myPet"/>  -->
    <!--        pet ⇒ setPet()                  -->
    <property name="pet" ref="myPet"/>
    <property name="name" value="Alex"/>
    <property name="age" value="28"/>
</bean>
```

```
package com.donnu.demo.spring_introduction;

public class Person {
    private Pet pet;
    private String name;
    private int age;

//    public Person(Pet pet) {
//        this.pet = pet;
//    }

    public void setPet(Pet pet) { this.pet = pet; }

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

    public void setAge(int age) { this.age = age; }

    public String getName() { return name; }

    public int getAge() { return age; }

    public void callYourPet() {
        System.out.println("Hey!");
        pet.say();
    }

    public Person() {
    }
}
```

Dependency Injection

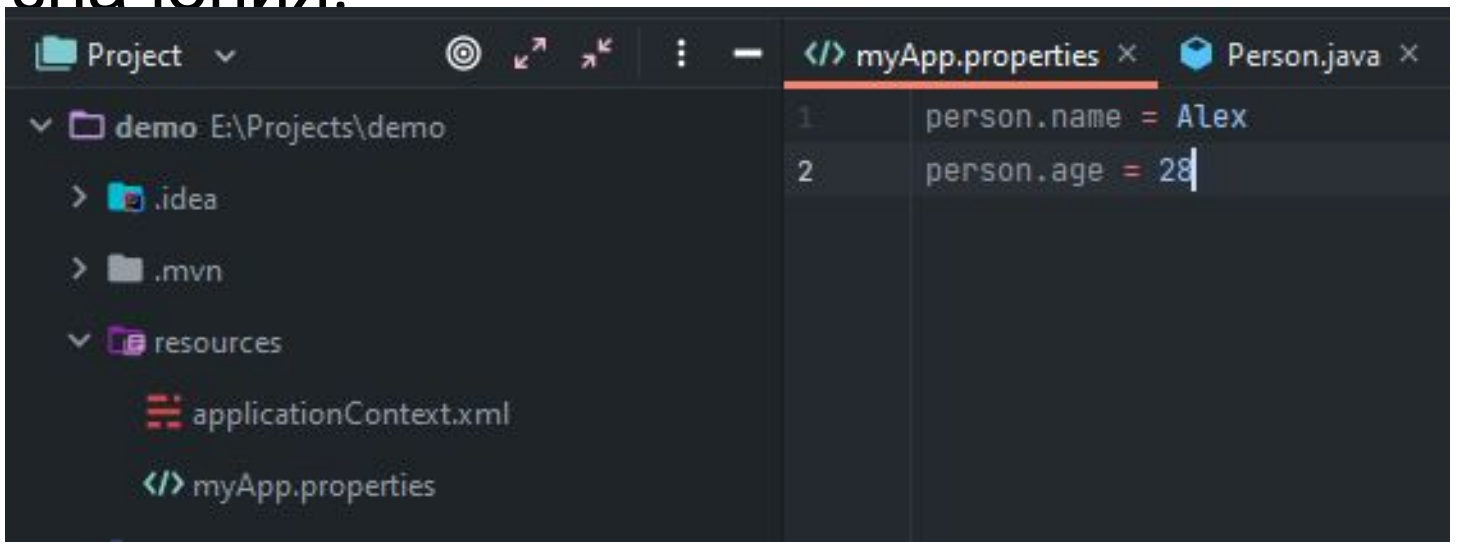
```java
package com.donnu.demo.spring_introduction;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test2 {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext context =
                new ClassPathXmlApplicationContext( configLocation: "applicationContext.xml");

        //Pet pet = new Dog();
        //Pet pet = context.getBean("myPet", Pet.class);

        Person person = context.getBean( name: "myPerson", Person.class);

        person.callYourPet();

        System.out.println(person.getName());
        System.out.println(person.getAge());

        context.close();
    }
}
```

Выведем заданные параметры при помощи геттеров.

```
"C:\Program Files\BellSoft\LibericaJDK-11\bin\java.exe"
23:40:57.413 [main] DEBUG org.springframework.context.su
23:40:57.791 [main] DEBUG org.springframework.beans.fact
23:40:57.831 [main] DEBUG org.springframework.beans.fact
23:40:57.852 [main] DEBUG org.springframework.beans.fact
Hey!
Гав-гав
Alex
28
23:40:58.053 [main] DEBUG org.springframework.context.su

Process finished with exit code 0
```

Dependency Injection

Внедрение строк и других значений из properties файла. Для начала создадим файл myApp.properties и пропишем в нем значения.



Dependency Injection

```xml
<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="com.donnu.demo.spring_introduction.Dog">
    </bean>

    <bean id="myPerson"
          class="com.donnu.demo.spring_introduction.Person">

        <!--        <constructor-arg ref="myPet"/>  -->
        <!--        pet => setPet()                 -->
        <property name="pet" ref="myPet"/>
        <property name="name" value="${person.name}"/>
        <property name="age" value="${p}"/>
    </bean>

</beans>
```

```
person.age=28                    myApp
person.name=Alex                 myApp
Press Enter to insert, Tab to replace  Next Tip
```

Добавляем контекст в наш конфигурационный файл. Теперь мы можем обращаться к значениям из properties.

Итоги

IoC – аутсорсинг создания и управления объектами, т.е. передача прав на создание и управление Spring-у

DI – это аутсорсинг добавления/внедрения зависимостей. DI делает объекты нашего приложения слабо зависимыми друг от друга.

Большое количество программистов использует эти термины как взаимозаменяемые.