# Аспектно - ориентированное программирование

## Инструменты для сборки и управления

Лекция №1

Антонов В.В.

# Что такое maven?

"Maven is a project development management and comprehension tool"

*с сайта maven.apache.org*

- ● инструмент для сборки и управления
- проектами [на Java] (build tool)
- ● инструмент для управления ЖЦ проекта
- ● инструмент для автоматизации

# Другие утилиты для сборки проектов

- shell/bat скрипты
- ● make
- ● cmake
- ● scons
- ● ant

# Почему Maven?

- на текущий момент одна из самых широко распространенных утилит для сборки в мире Java (загляните в исходники почти любого проекта от apache.org - найдете там pom.xml)
- огромный актуальный репозиторий артефактов в репозиториях maven
- поддерживается большинством современных IDE (Eclipse, IntelliJ IDEA, NetBeans и т.д.)

# Ключевые преимущества

- **декларативный язык описания проекта** (POM)
- ● автоматическое управление зависимостями
- ● огромный, поддерживаемый в актуальном состоянии репозиторий артефактов
- ● модульная расширяемая за счет плагинов архитектура, огромное количество плагинов

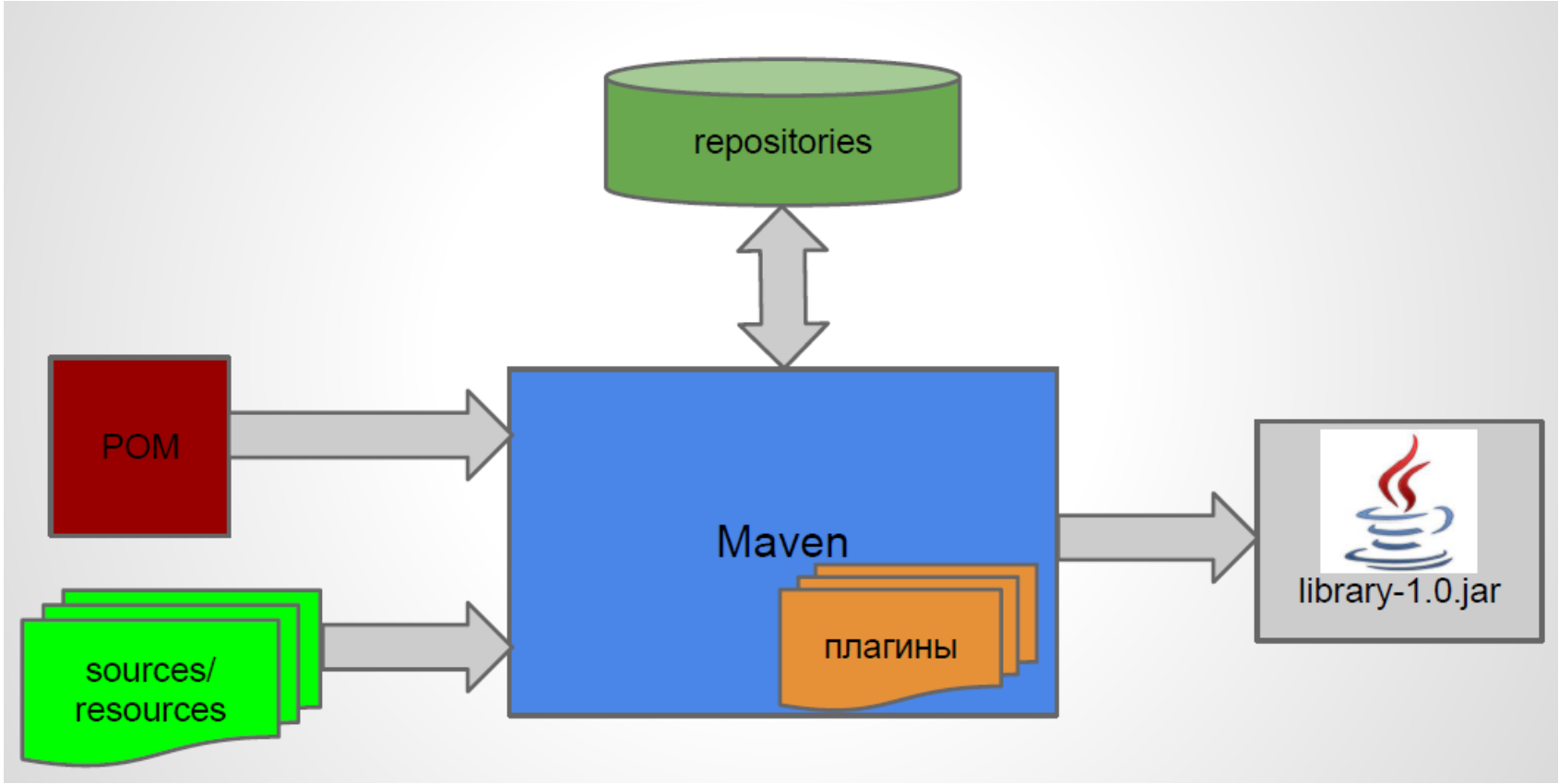
# Главные недостатки

- сложность освоения
- неочевидность (контринтуитивность) в некоторых моментах
- не очень хорошая документация
- огромное количество плагинов (трудно сориентироваться)
- трудно разобраться если что то пошло не так (возникла ошибка)
- необходим доступ в Интернет или собственный репозиторий артефактов

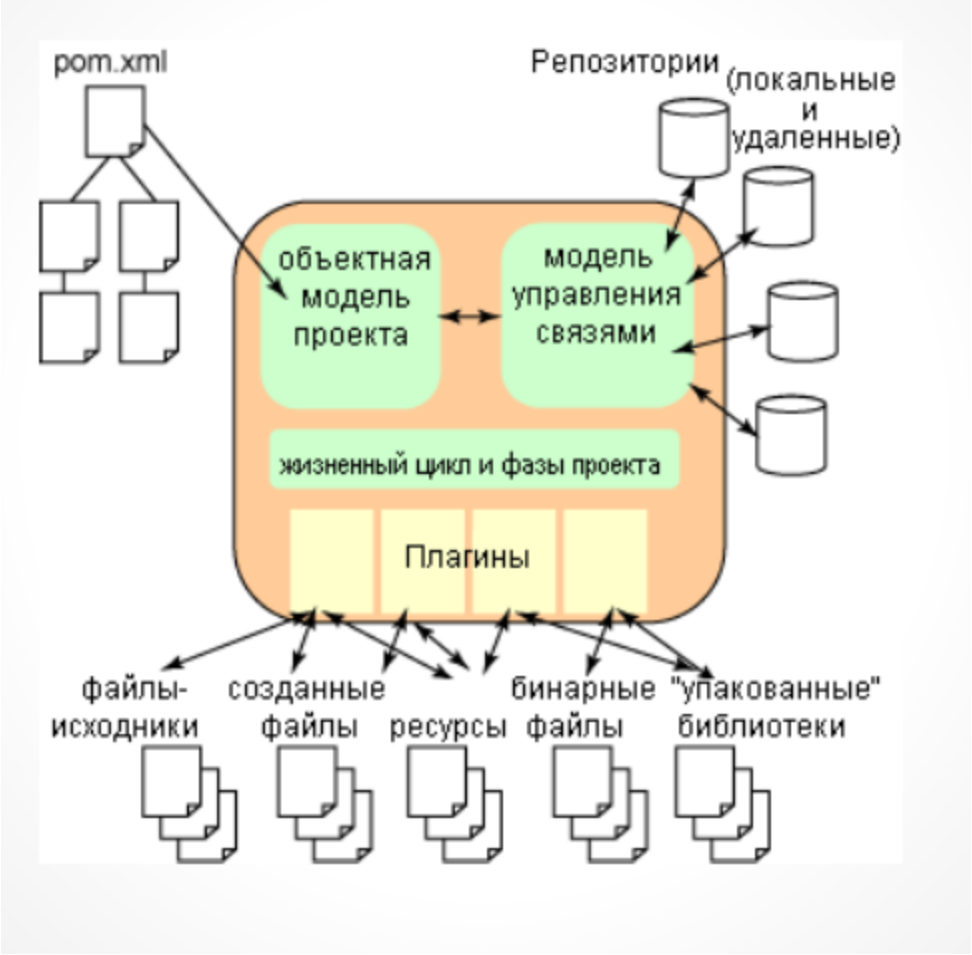
# Установка Maven

- ● требует наличия на машине JDK версии >=1.5
- ● скачиваем с сайта проекта http://apache.maven.org
- ● разворачиваем архив
- ● прописываем переменную окружения M2_HOME
- ● прописываем путь $M2_HOME/bin в PATH
- ● запуск командой mvn

# Как все это работает?



# Еще раз как все ЭТО работает?



# Артефакт

- Что есть Артефакт? Да все что угодно что производит наш проект (jar,war, ear и т.п.) или использует maven (плагин)
- Результатом работы Maven является создание (построение) артефакта, а так же ряд дополнительных действий над ним (тестирование, инсталляция в локальный репозиторий, deployment)
- ● Сам артефакт зависит от других артефактов (наших и внешних, плагинов maven)

# Координаты артефакта

- `<modelVersion>` - версия POM-модели (всегда 4.0.0)
- `<groupId>` - группа или организация, к которой принадлежит проект. Чаще всего выражается в виде перевернутого наоборот доменного имени
- `<artifactId>` - имя, которое будет передано библиотеке экземпляра(artifact) проекта (к примеру, имя его JAR или WAR файла)
- `<version>` - версия, с которой будет собран проект
- `<packaging>` - как проект должен быть упакован. По умолчанию, с "jar" упаковывается в JAR-файл, "war" - WAR-файл

# Примеры maven координат

log4j

```
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.16</version>
```

spring

```
<groupId>org.springframework</groupId>
<artifactId>spring-core</artifactId>
<version>3.1.0.RELEASE</version>
```

# POM файл

- POM - Project Object Model, xml файл, обычно называется pom.xml
- POM файл содержит описание нашего проекта (декларативный стиль!) и все специфические его настройки.
- Пример минимального POM файла (данный пример работает!!!):

```
<project xmlns=...>
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>my-project</artifactId>
    <version>1.0</version>
</project>
```

# Минимальные требования к POM

- Минимально POM файл проекта должен содержать лишь версию модели и координаты артефакта проекта.

# Репозитории

- Репозиторий maven это файловое хранилище с метаинформацией и быстрым поиском и доступом
- ● Бывают двух типов
- ○ local ( находятся в ~/.m2/repository )
- ○ remote (например, стандартный http://repo1.maven.org/maven2 или внутренний репозиторий компании, например, Nexus)
- ● используются для хранения и получения зависимостей (dependencies) проекта и плагинов maven

# Lifecycle (жизненный цикл) проекта

Существуют три стандартных lifecycles:

- ● clean - очистка проекта
- ● default - построение проекта из исходных кодов
- ● site - построение вторичных артефактов

(документация, wiki, сайт и т.п.)

- Жизненный цикл состоит из фаз. К каждой фазе может быть привязан ноль или более goal-ов различных плагинов. По умолчанию, набор фаз с привязанными плагинами стандартен и зависит от типа артефакта проекта (конкретно - от типа packaging).

# Сборка кода

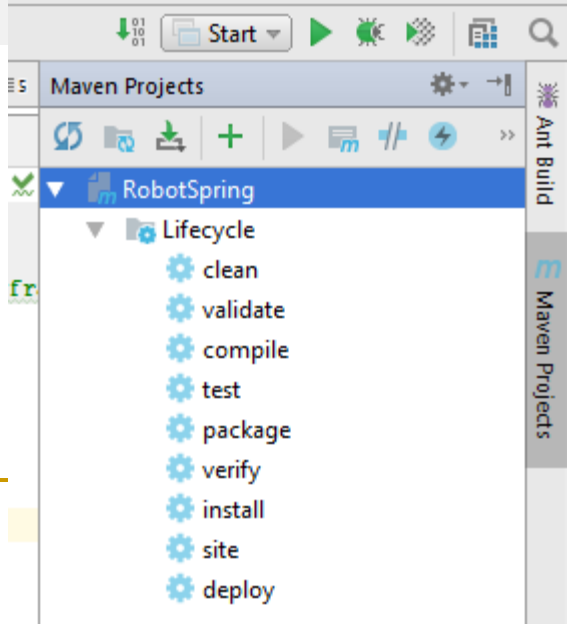
`mvn compile` скомпилированные *.class* файлы в *target/classes*

`mvn package`

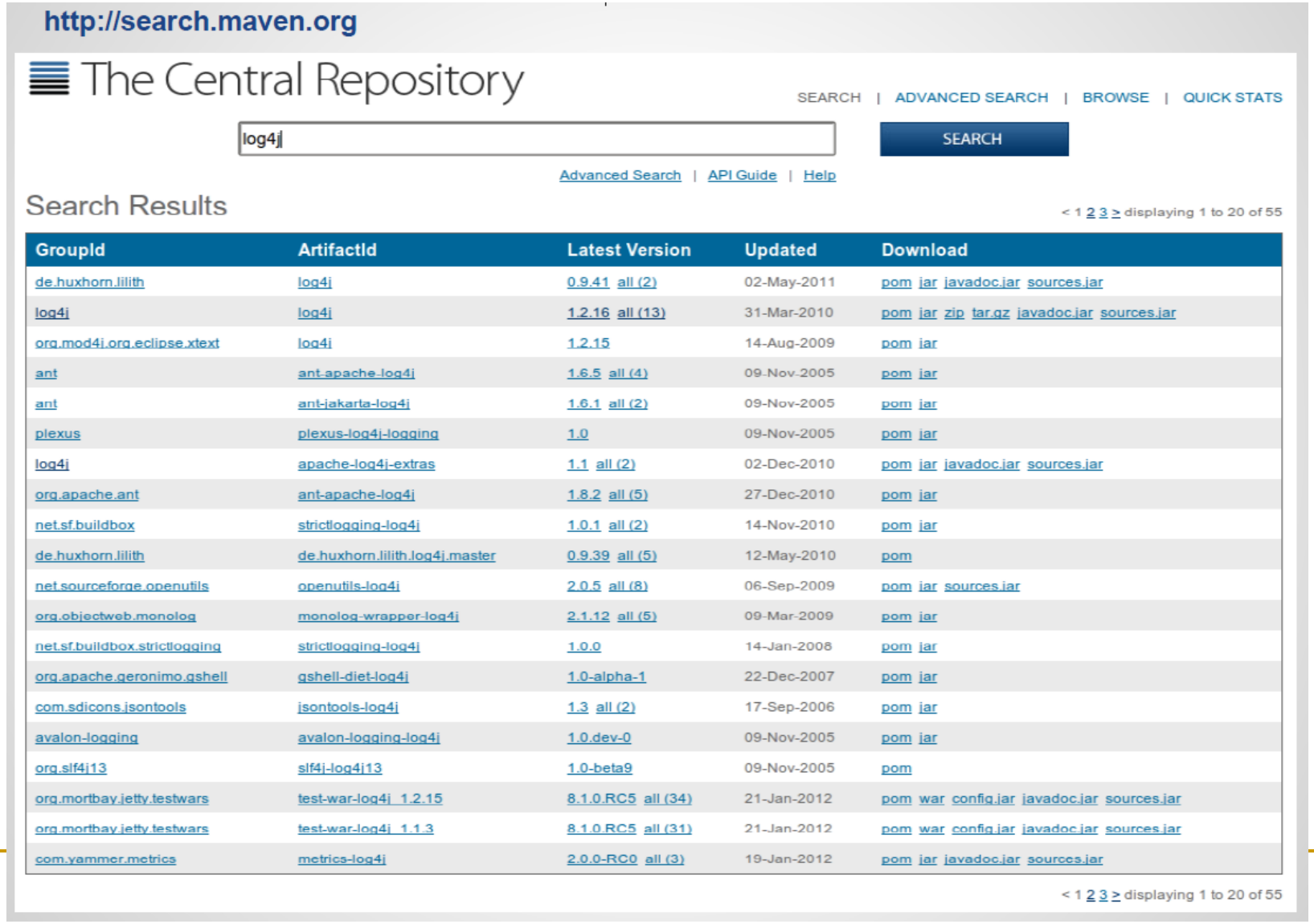
Задача *package* включает компиляцию вашего Java кода, запуск тестов, а в конце упаковывает в JAR-файл в *target* директории. Название JAR-файла будет основано на `<artifactId>` и `<version>`.

```
mvn install
```

Задача *install* включает компиляцию, тестирование, упаковку кода проекта, а затем копирование в локальный репозиторий, тем самым другие проекты смогут ссылаться на него как на зависимость.



# Как найти нужную библиотеку



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/20
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.springframework</groupId>
    <artifactId>gs-maven</artifactId>
    <packaging>jar</packaging>
    <version>0.1.0</version>

    <!-- tag::joda[] -->
    <dependencies>
        <dependency>
            <groupId>joda-time</groupId>
            <artifactId>joda-time</artifactId>
            <version>2.2</version>
        </dependency>
    </dependencies>
    <!-- end::joda[] -->

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>2.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer
                                    implementation="org.apache.maven.plugins.shade
                                    <mainClass>hello.HelloWorld</mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```


```xml
<dependency>
  <groupId>com.squareup.okhttp</groupId>
  <artifactId>okhttp</artifactId>
  <version>2.2.0</version>
</dependency>
```

```
compile 'com.squareup.okhttp:okhttp:2.2.0'
```

# build.gradle

```
apply plugin: "java"
apply plugin: "application"

task wrapper(type: Wrapper) {
    gradleVersion = '1.12'
}
```

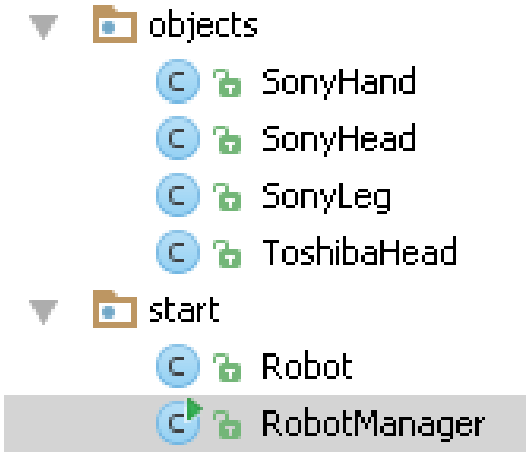
Плагин java содержит в себе задачи:

- jar — собрать jar архив,
- compileJava — скомпилировать исходные коды и др. Подробнее о плагине можно почитать тут.

Плагин application содержит в себе задачи:

- run — запуск приложения;
- installApp — установка приложения на компьютер, эта задача создает исполняемые файлы для *nix и для windows (bat файл);
- distZip — собирает приложение в zip архив, помещая туда все jar файлы

```
apply plugin: "java"
apply plugin: "application"

mainClassName = "com.example.Main"

sourceCompatibility = JavaVersion.VERSION_1_7
targetCompatibility = JavaVersion.VERSION_1_7

repositories {
    mavenCentral()
}

dependencies {
    compile "log4j:log4j:1.2.17"
}

jar {
    manifest.attributes("Main-Class": mainClassName);
}

task wrapper(type: Wrapper) {
    gradleVersion = "1.12"
}
```



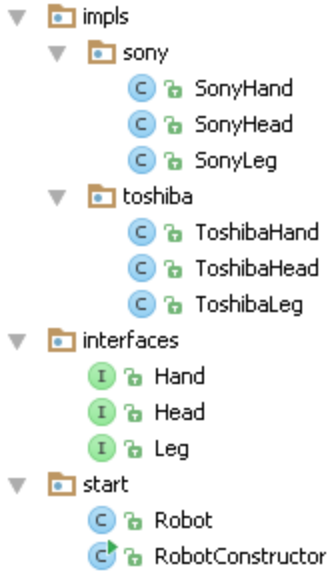

```
public class RobotManager {

    public static void main(String[] args){
        Robot robot = new Robot();
        robot.action();
    }

public class Robot {

    private SonyHand hand = new SonyHand();
    private SonyLeg leg = new SonyLeg();
    //private SonyHead head = new SonyHead();
    private ToshibaHead head = new ToshibaHead();
    public void action(){
        head.calc();
        hand.catchSomething();
        leg.go();
    }
}
```

```java
public class SonyHead {

    public void calc(){
        System.out.println("Thinking...");
    }

}
```

```java
public class SonyLeg {

    public void go(){
        System.out.println("Go!");
    }

}
```

```java
public class ToshibaHead {

    public void calc(){
        System.out.println("Toshiba Thinking...");
    }

}
```

```java
public class SonyHand {

    public void catchSomething() { System.out.println("Catched!"); }

}
```




```
public interface Head {

    public void calc();
}

public class ToshibaHead implements Head{

    public void calc(){
        System.out.println("Thinking about Toshiba...");
    }

public class SonyHead implements Head{

    public void calc() { System.out.println("Thinking about Sony..."); }

}
```

```java
public class RobotConstructor {

    public static void main(String[] args){

        SonyHand sonyHand = new SonyHand();
        SonyLeg toshibaLeg = new SonyLeg();
        SonyHead sonyHead = new SonyHead();

        Robot robot = new Robot(sonyHand, toshibaLeg, sonyHead);

        robot.action();

    }
```

```java
public class Robot {

    private Hand hand;
    private Leg leg;
    private Head head;

    public Robot(Hand hand, Leg leg, Head head) {
        super();
        this.hand = hand;
        this.leg = leg;
        this.head = head;
    }

    public void action(){
        head.calc();
        hand.catchSomething();
        leg.go();
    }
```

# Фреймворк – что это?

- Фреймворк это программная оболочка, позволяющая упростить и ускорить решение типовых задач, характерных для данного языка программирования. Само слово framework означает «каркас» в переводе с английского.

Классические фреймворки:

- Javascript - jQuery – библиотека с готовыми функциями визуальных эффектов, AJAX-запросов и прочих полезных вещей.
- PHP - Yii
- Java – Spring

Фреймворк отличается от понятия библиотеки тем, что библиотека может быть использована в программном продукте просто как набор подпрограмм близкой функциональности, не влияя на архитектуру программного продукта и не накладывая на неё никаких ограничений.

В то время как фреймворк диктует правила построения архитектуры приложения, задавая на начальном этапе разработки поведение по умолчанию, каркас, который нужно будет расширять и изменять согласно указанным требованиям.

# Spring

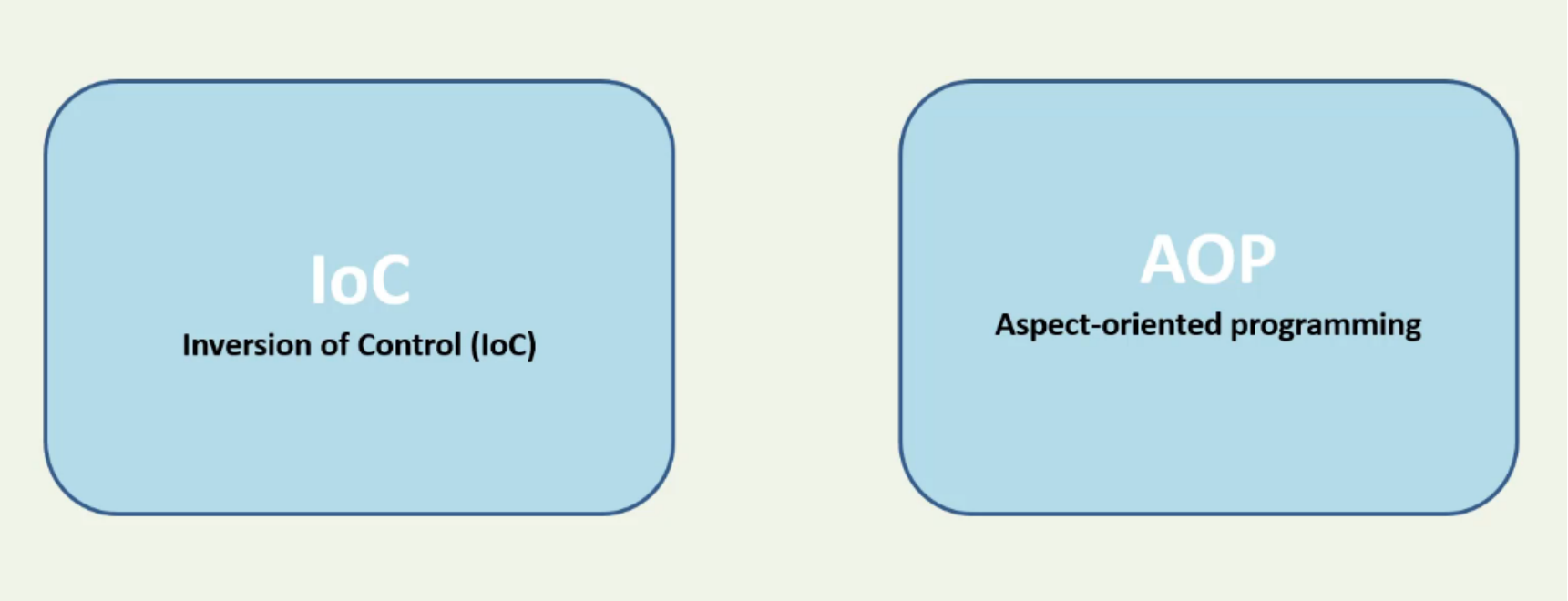
- Основная цель - упрощение разработки любых приложений на Java, разгрузка кода
- Программный код становится проще, связь между объектами слабее
- Каждый объект занимается свои делом (POJO)
- Использование принципов ООП на полную мощность
- Дополнительная логика подключается извне
- Готовые встроенные модули (работа с БД, безопасность, транзакции, авторизация, сервисы и пр.)
- **Spring** – огромный комплекс, который может объединить и упростить использование технологий

# Проекты Spring

Список проектов
http://spring.io/projects

- **Spring Framework - основы**
- Spring Boot
- Spring XD
- Spring Data
- Spring Integration
- Spring Batch
- Spring Security
- Spring Hateoas
- Spring Social
- Spring AMQP
- Spring Mobile
- Spring for Android
- Spring WebFlow
- Spring WebServices
- Spring LDAP
- ...

# Spring


IoC
Inversion of Control (IoC)
AOP
Aspect-oriented programming

# IoC


Автомобиль Lexus
Летние шины Bridgestone
Бензин 92
Медиасистема Sony
Автомобиль
Летние шины
Бензин
Медиасистема
• «Жесткое» связывание
• Трудно изменять код
• Трудно тестировать
• Трудно заменять объекты
• Слабые связи (на уровне абстракций)
• Код изменять легче
• Легко тестировать
• Делегирование создания объектов контейнеру

# IoC контейнер

**Основной момент IoC:**

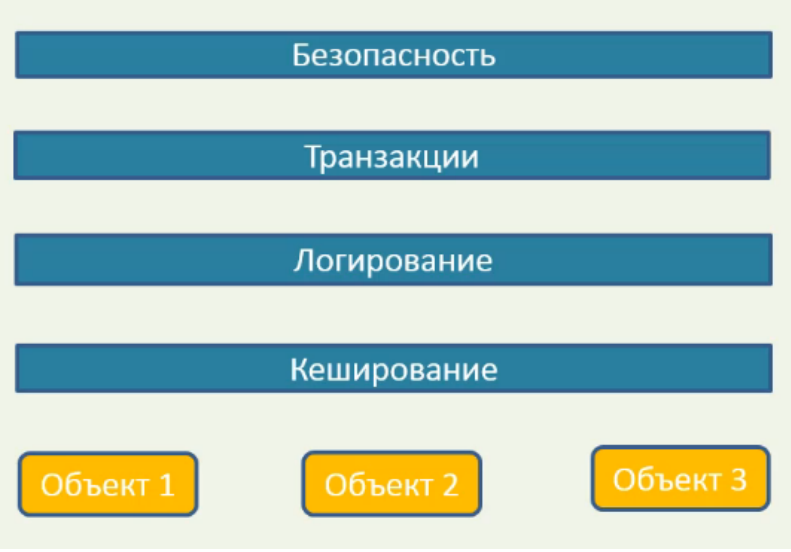
- Сначала создаете абстракцию
- Потом подставляете реализацию

**Контейнер**

- Управляет зависимостями
- Связывает объекты между собой
- Управляет их жизненным циклом
- **Dependency Injection –** в объект внедряется ссылка на другой объект

# АОП

- Разделение основного функционала и дополнительного – без перемешивание их между собой
- Кеширование, логирование, транзакции, безопасность и пр.
- Аспект - функциональность, не относящаяся напрямую к бизнес логике
- Аспекты можно использовать в любых проектах



# Spring

Основной момент:

- Создание объектов
- Настройка взаимодействия (вне объектов)
- Подключение дополнительных аспектов

```java
public class ModelT1000 implements Robot {

    private Hand hand;
    private Leg leg;
    private Head head;

    public ModelT1000() {
    }

    public ModelT1000(Hand hand, Leg leg, Head head) {
        super();
        this.hand = hand;
        this.leg = leg;
        this.head = head;
    }

    @Override
    public void fire() {
        head.calc();
        hand.catchSomething();
        leg.go();
    }

    @Override
    public void dance() { System.out.println("T1000 is dancing!"); }

}
```

```java
public interface Robot {
    void fire();

    void dance();
}
```

```java
public interface Leg {

    public void go();

}
```

```java
public interface Head {

    public void calc();
}
```

```java
public interface Hand {

    public void catchSomething();

}
```

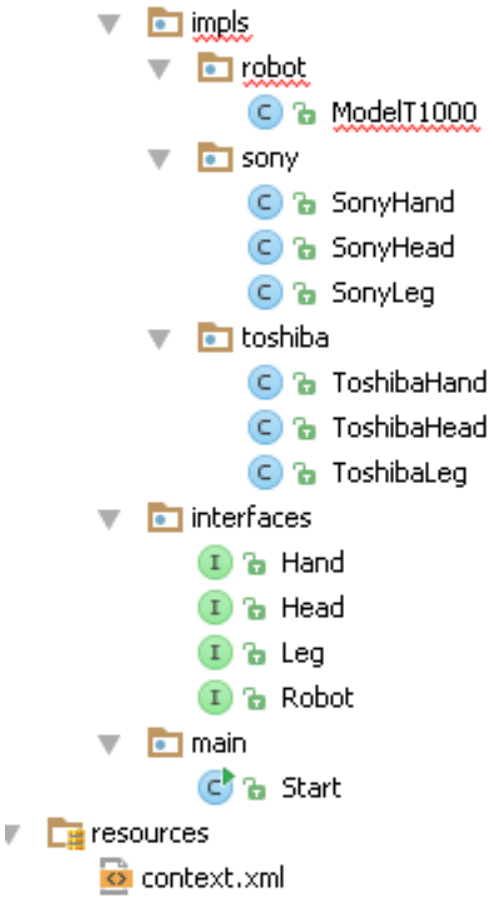
```java
public class SonyHand implements Hand{

    public void catchSomething(){
        System.out.println("Catched from Sony!!");
    }

}
```

```java
public class ToshibaHand implements Hand{

    public void catchSomething(){
        System.out.println("Catched from Toshiba!");
    }

}
```

```java
public class SonyHead implements Head{

    public void calc(){
        System.out.println("Thinking about Sony...");
    }

}
```

```java
public class SonyLeg implements Leg {

    public void go(){
        System.out.println("Go to Sony!");
    }

}
```

# Проект робот в Spring



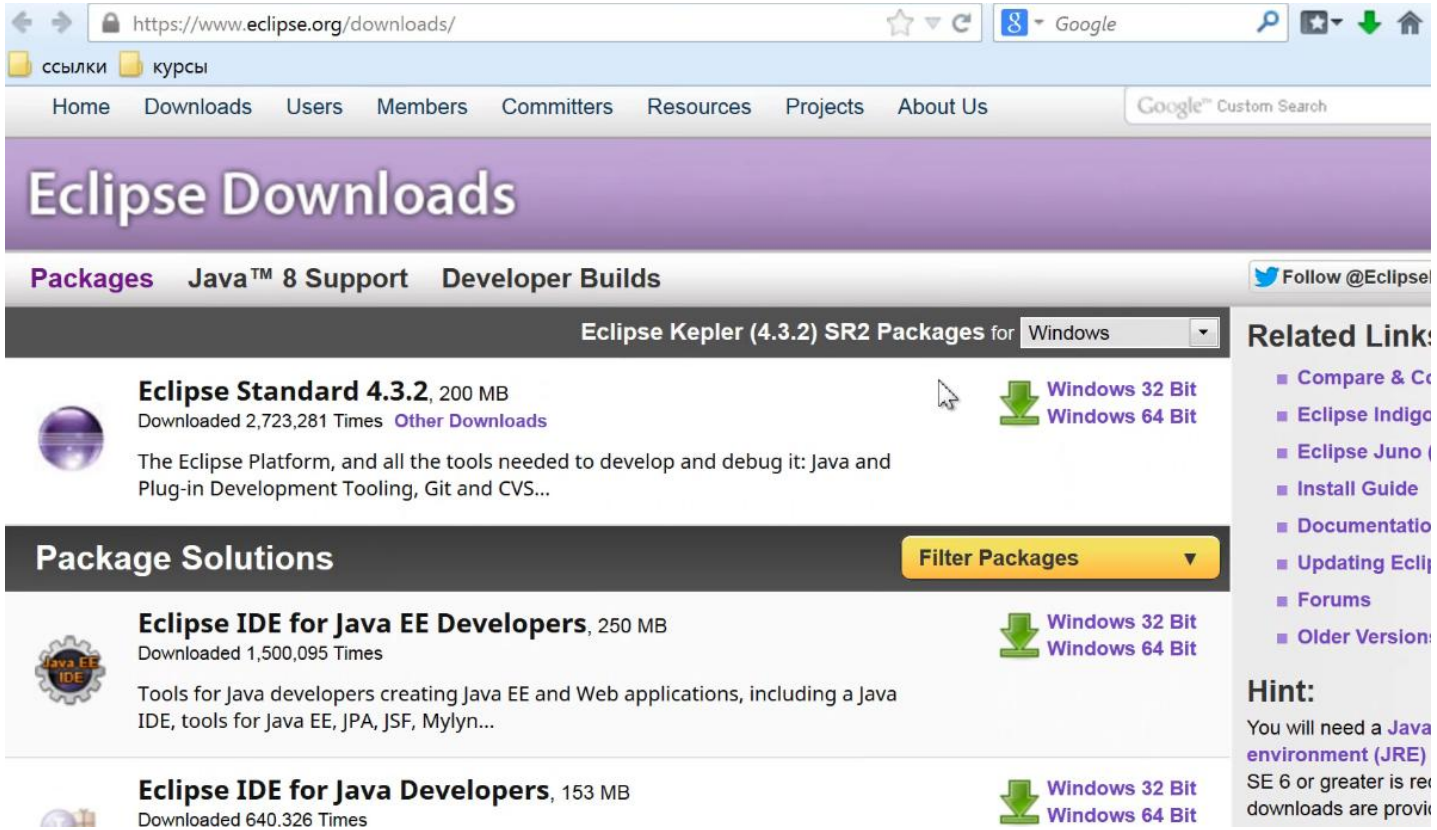

```java
public class Start {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
        ModelT1000 t1000 = (ModelT1000) context.getBean("t1000");
        t1000.fire();
    }
}
```

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http

    <bean id="t1000" class="ru.spring.impls.robot.ModelT1000" />
    <!--bean id="t1000" class="ru.spring.impls.robot.ModelT1000New"/-->

</beans>
```

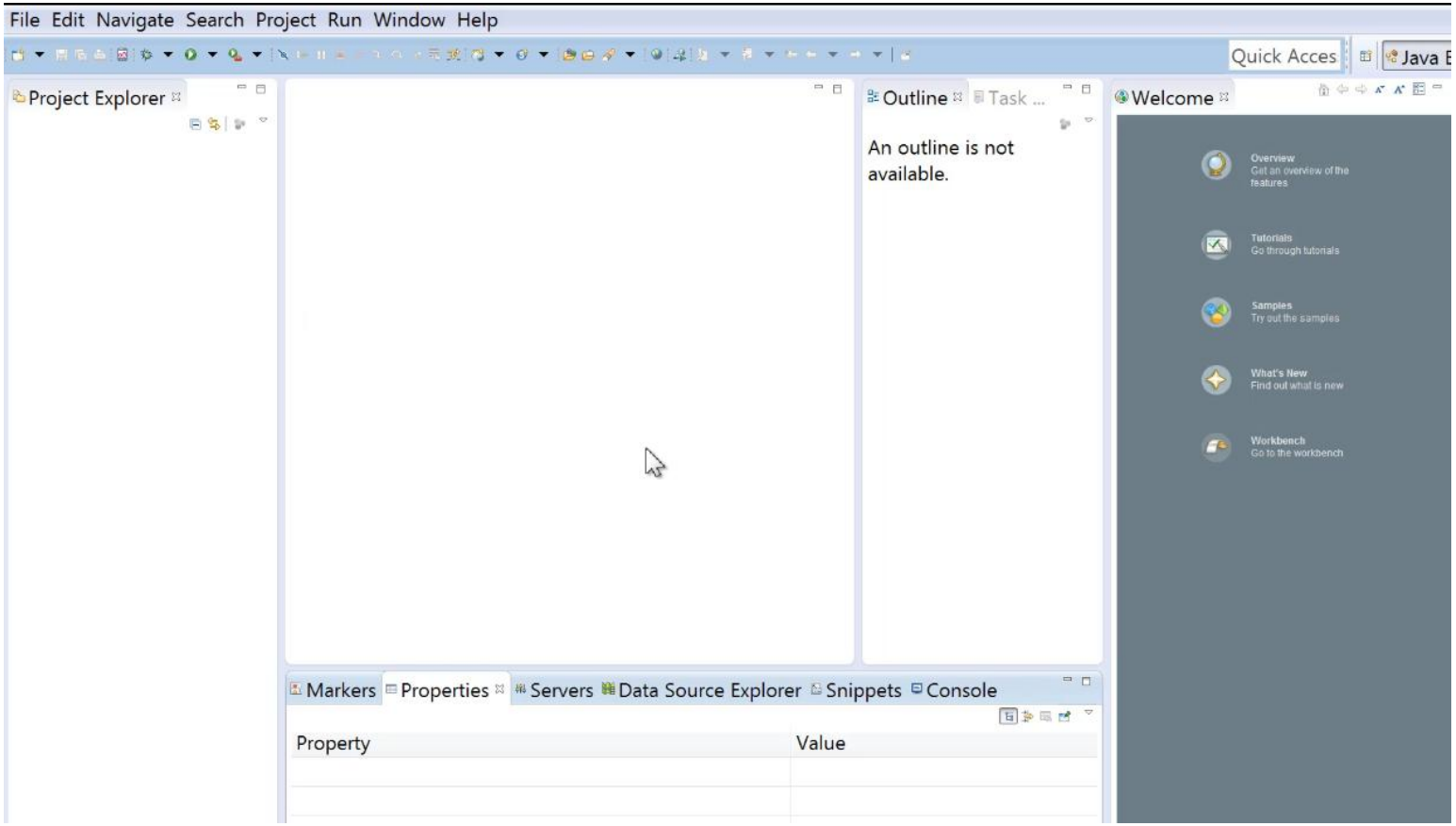
- Установка eclipse – среда разработки
- Установка Maven – инструмент сборки
- Установка Spring – core библиотеки
- Создание тестового проекта – проверка работоспособности

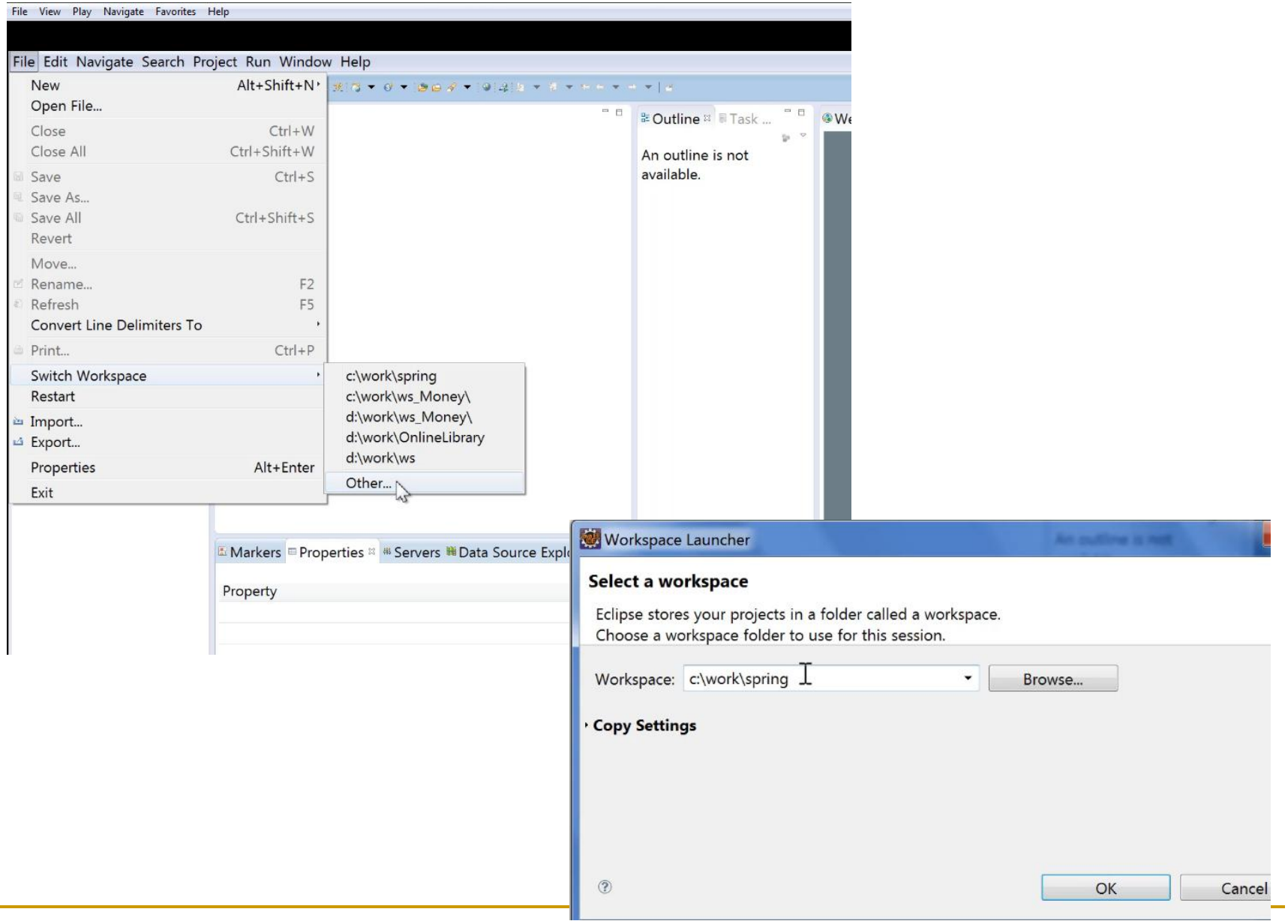
# Инструменты

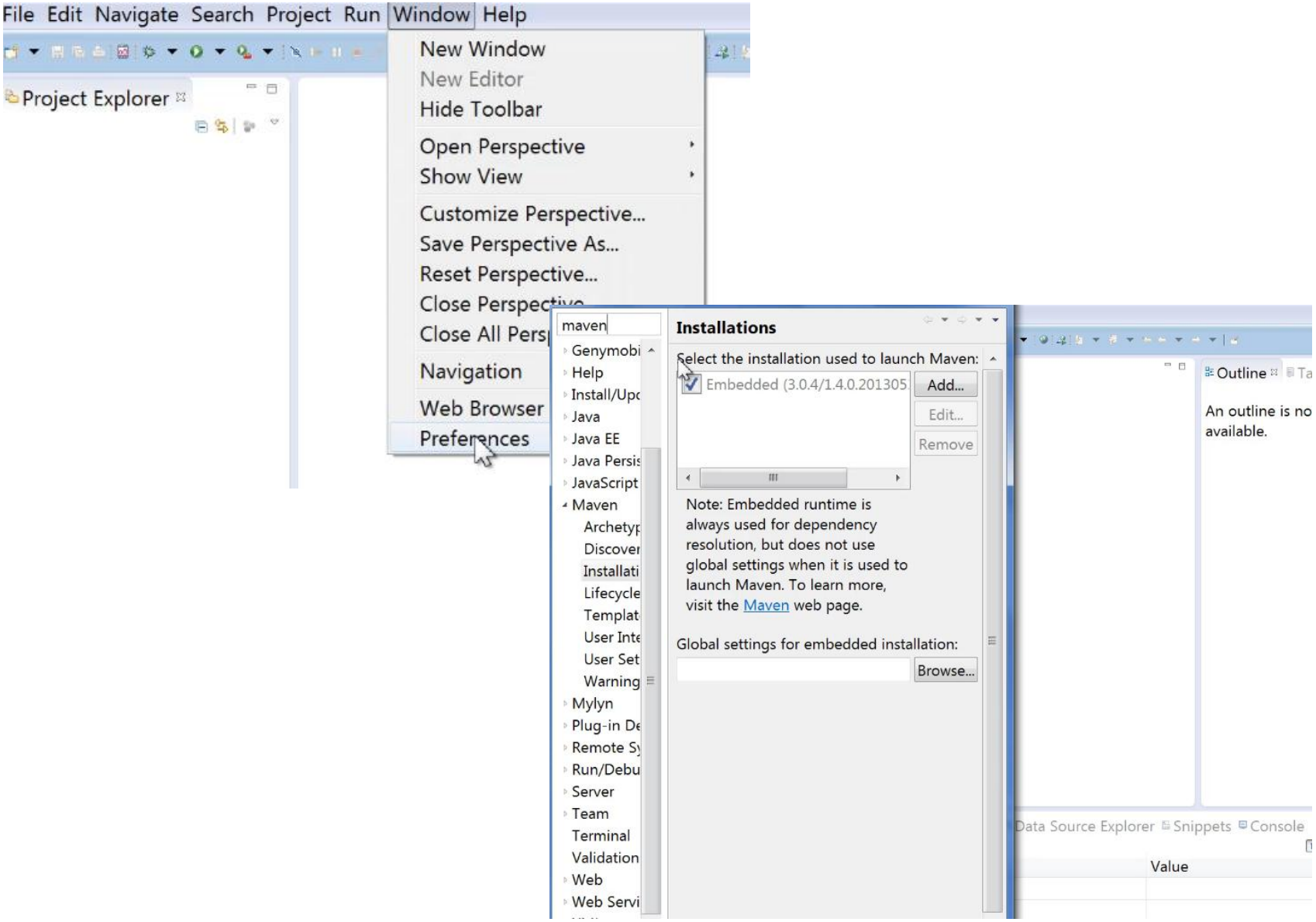
- Eclipse IDE for Java EE Developers https://www.eclipse.org/downloads/



# Eclipse

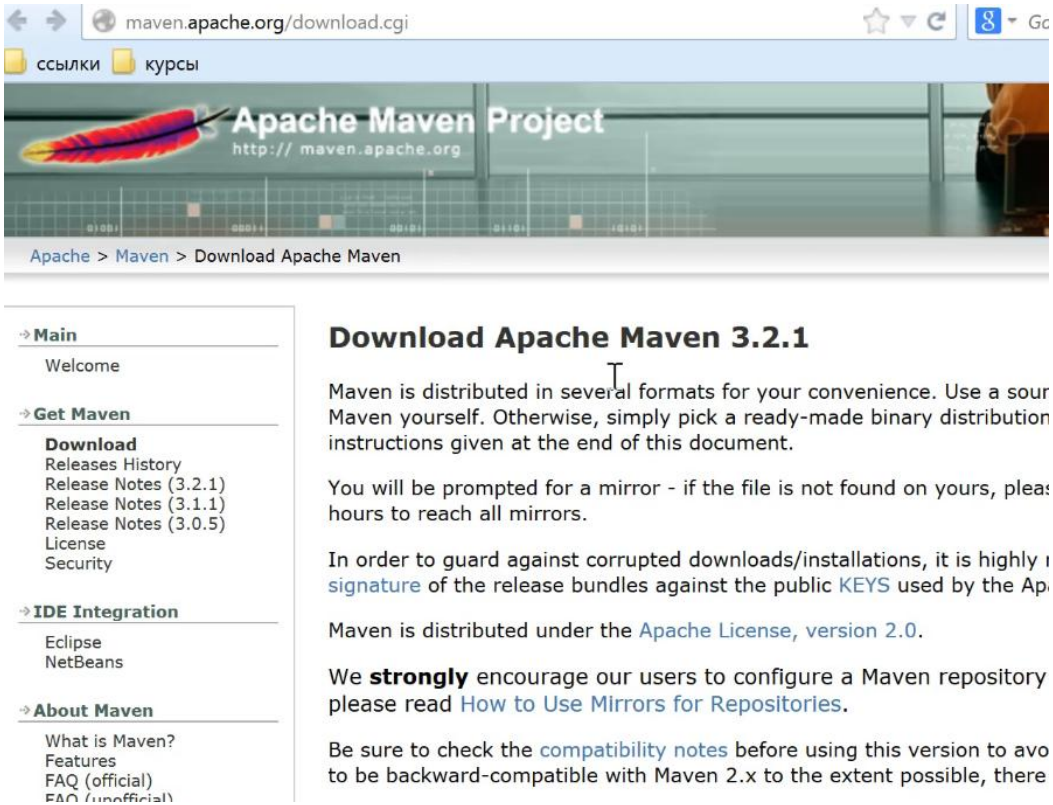

File Edit Navigate Search Project Run Window Help
Quick Acces
Java E
Project Explorer
Outline
Task ...
An outline is not available.
Welcome
Overview
Get an overview of the features
Tutorials
Go through tutorials
Samples
Try out the samples
What's New
Find out what is new
Workbench
Go to the workbench
Markers
Properties
Servers
Data Source Explorer
Snippets
Console
Property
Value

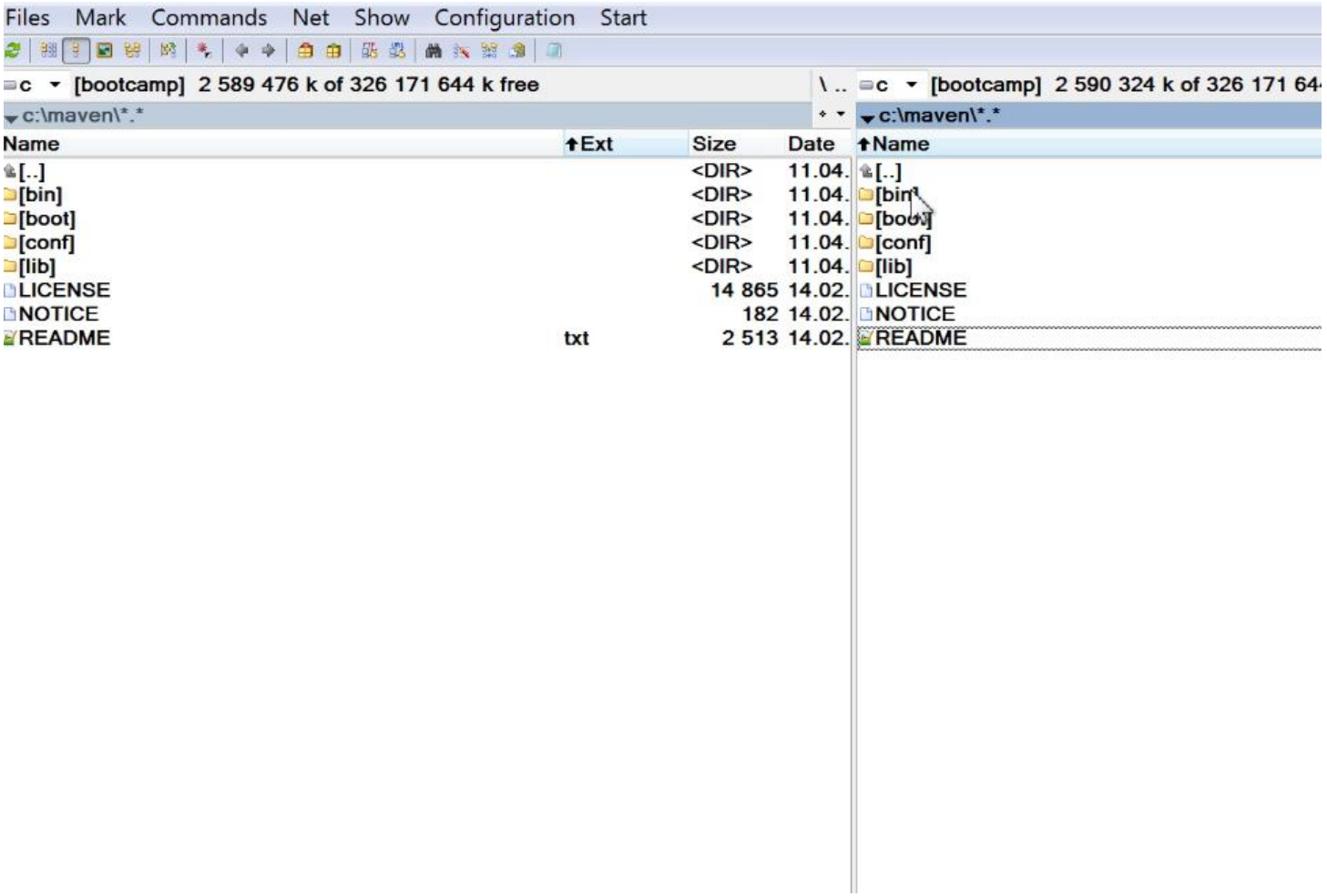

File View Play Navigate Favorites Help
File Edit Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
Exit
c:\work\spring
c:\work\ws_Money\
d:\work\ws_Money\
d:\work\OnlineLibrary
d:\work\ws
Other...
Outline
Task ...
An outline is not available.
Markers
Properties
Servers
Data Source Explo
Property
Workspace Launcher
Select a workspace
Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.
Workspace: c:\work\spring
Browse...
Copy Settings
OK
Cancel

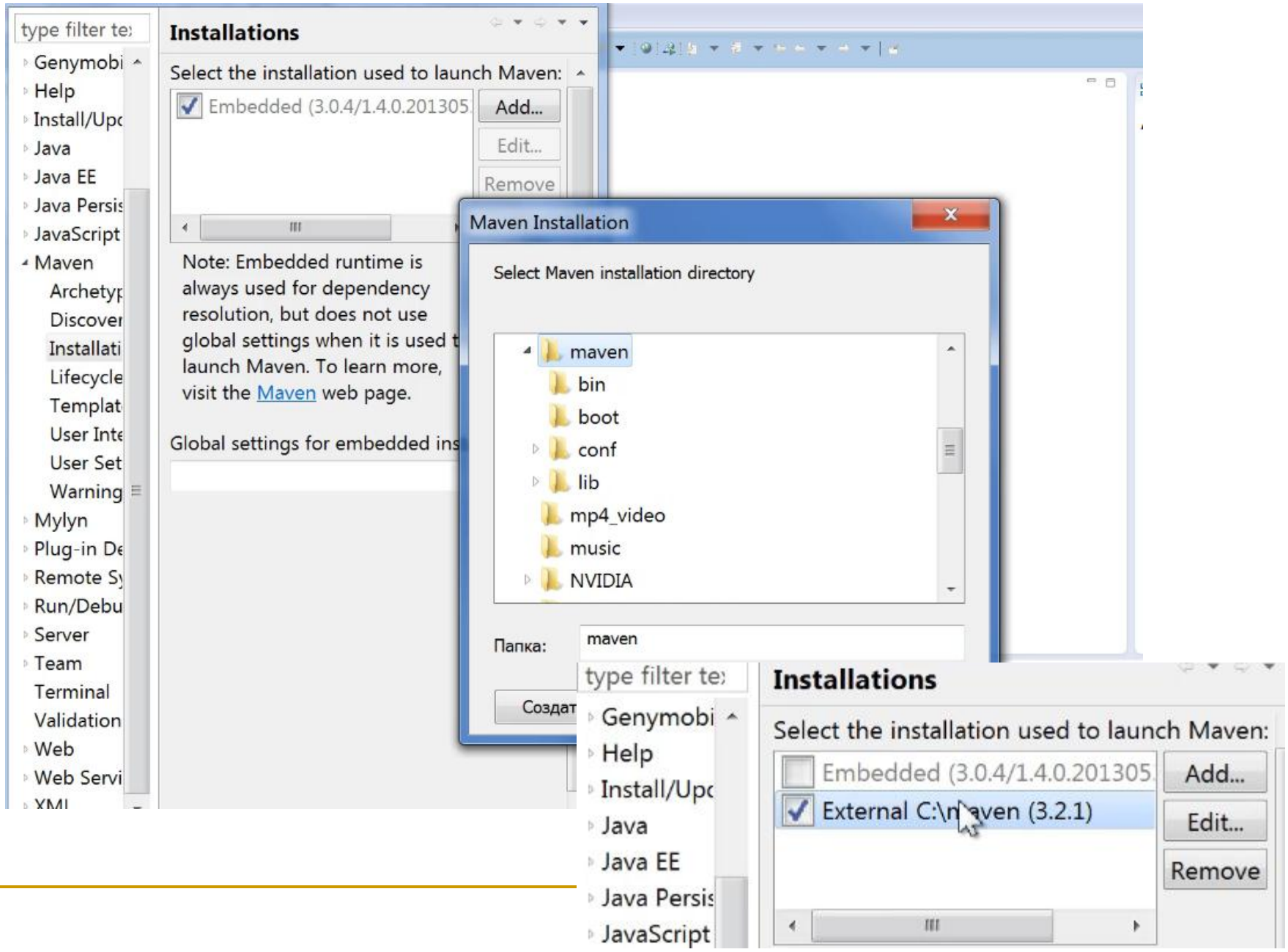

File Edit Navigate Search Project Run Window Help
Project Explorer
New Window
New Editor
Hide Toolbar
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective...
Close Perspective
Close All Perspectives
Navigation
Web Browser
Preferences
maven
Genymobi
Help
Install/Upd
Java
Java EE
Java Persis
JavaScript
Maven
Archetyp
Discover
Installati
Lifecycle
Templat
User Inte
User Set
Warning
Mylyn
Plug-in De
Remote Sy
Run/Debu
Server
Team
Terminal
Validation
Web
Web Servi
Installations
Select the installation used to launch Maven:
Embedded (3.0.4/1.4.0.201305
Add...
Edit...
Remove
Note: Embedded runtime is always used for dependency resolution, but does not use global settings when it is used to launch Maven. To learn more, visit the Maven web page.
Global settings for embedded installation:
Browse...
Outline
An outline is no available.
Data Source Explorer
Snippets
Console
Value

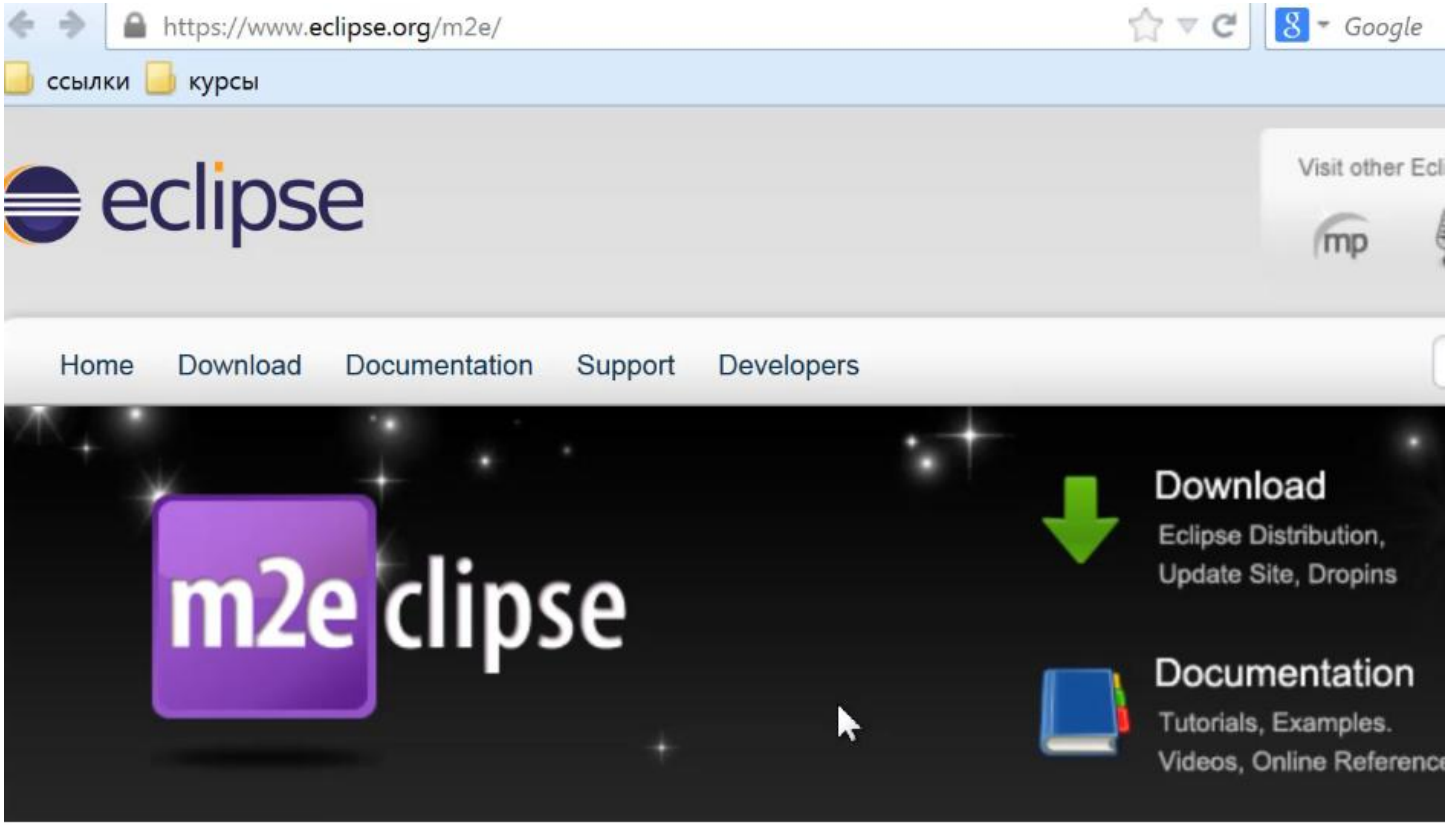
# Инструменты

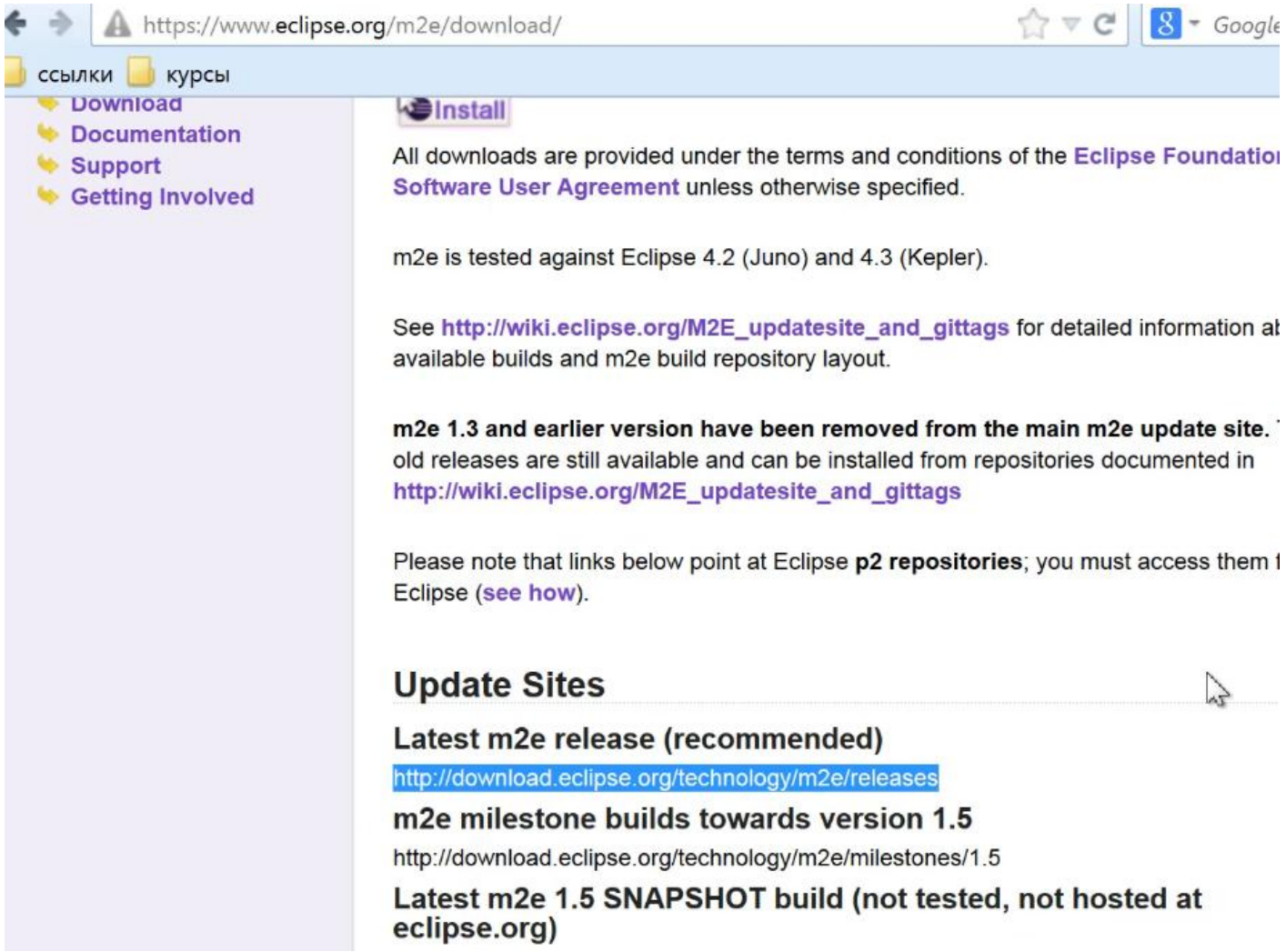
- Maven http://maven.apache.org/download.cgi
- Maven Eclipse Integration https://www.eclipse.org/m2e/

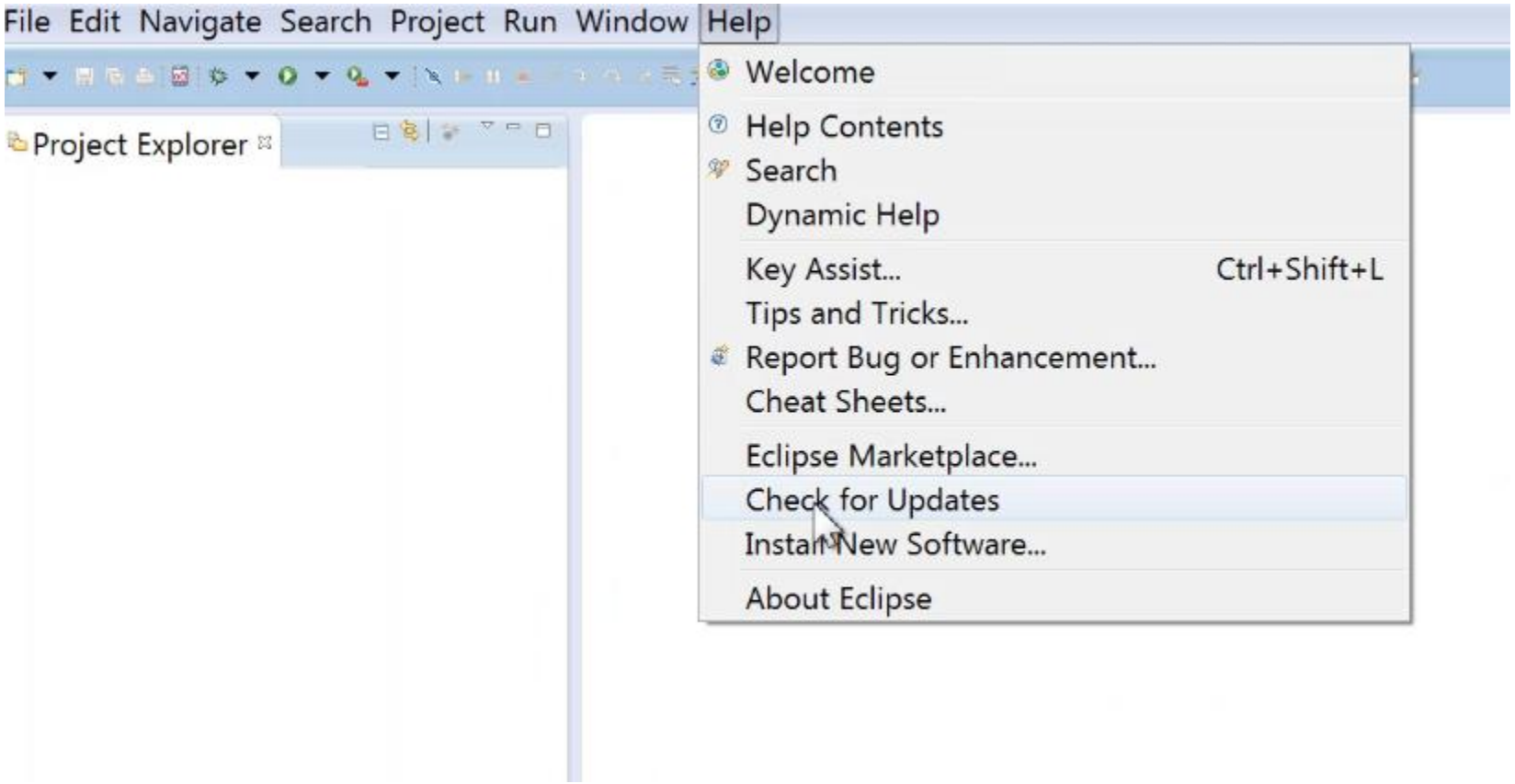


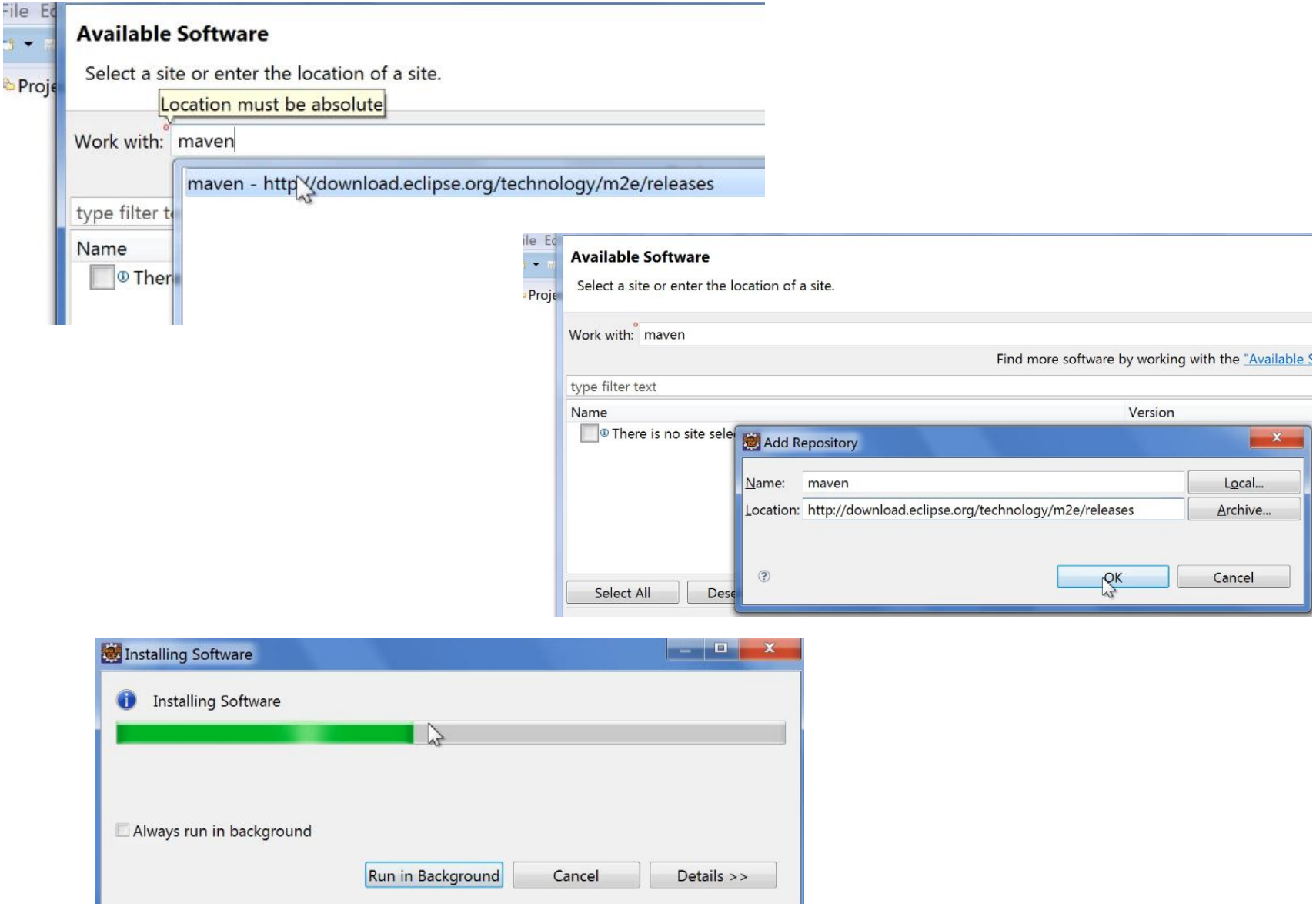

Files Mark Commands Net Show Configuration Start
c [bootcamp] 2 589 476 k of 326 171 644 k free
\ ..
c [bootcamp] 2 590 324 k of 326 171 64
c:\maven*.*
c:\maven*.*
Name Ext Size Date
Name
[..] <DIR> 11.04.
[bin] <DIR> 11.04.
[boot] <DIR> 11.04.
[conf] <DIR> 11.04.
[lib] <DIR> 11.04.
LICENSE 14 865 14.02.
NOTICE 182 14.02.
README txt 2 513 14.02.
[..]
[bin]
[boot]
[conf]
[lib]
LICENSE
NOTICE
README


type filter te:
Genymobi
Help
Install/Upc
Java
Java EE
Java Persis
JavaScript
Maven
Archetyp
Discover
Installati
Lifecycle
Templat
User Inte
User Set
Warning
Mylyn
Plug-in De
Remote Sy
Run/Debu
Server
Team
Terminal
Validation
Web
Web Servi
XML
Installations
Select the installation used to launch Maven:
Embedded (3.0.4/1.4.0.201305
Add...
Edit...
Remove
Note: Embedded runtime is always used for dependency resolution, but does not use global settings when it is used t launch Maven. To learn more, visit the Maven web page.
Global settings for embedded ins
Maven Installation
Select Maven installation directory
maven
bin
boot
conf
lib
mp4_video
music
NVIDIA
Папка:
maven
Создат
type filter te:
Genymobi
Help
Install/Upc
Java
Java EE
Java Persis
JavaScript
Installations
Select the installation used to launch Maven:
Embedded (3.0.4/1.4.0.201305
External C:\maven (3.2.1)
Add...
Edit...
Remove


https://www.eclipse.org/m2e/
Google
ссылки
курсы
eclipse
Visit other Ecl
mp
Home
Download
Documentation
Support
Developers
m2eclipse
Download
Eclipse Distribution,
Update Site, Dropins
Documentation
Tutorials, Examples.
Videos, Online Reference
Maven Integration (m2e)
The goal of the m2e project is to provide a first-class Apache Maven support in the Eclipse IDE, making it easier to edit Maven's pom.xml, run a build from the IDE and much more. For Java developers, the very tight integration with JDT greatly

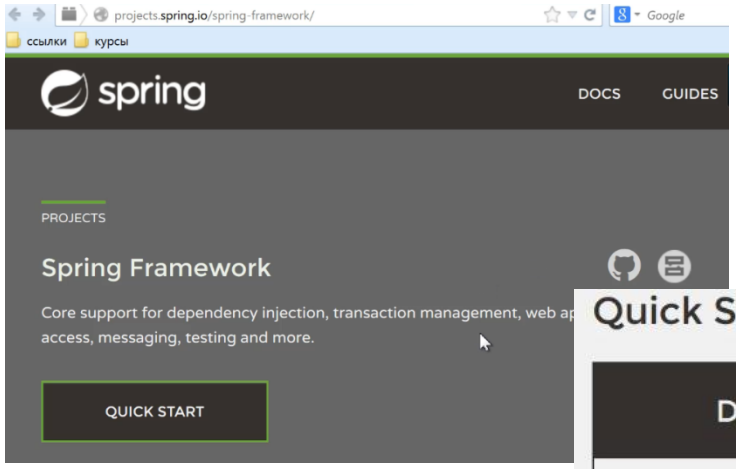

https://www.eclipse.org/m2e/download/
ссылки
курсы
Download
Documentation
Support
Getting Involved
Install
All downloads are provided under the terms and conditions of the Eclipse Foundatio
Software User Agreement unless otherwise specified.
m2e is tested against Eclipse 4.2 (Juno) and 4.3 (Kepler).
See http://wiki.eclipse.org/M2E_updatesite_and_gittags for detailed information ab
available builds and m2e build repository layout.
m2e 1.3 and earlier version have been removed from the main m2e update site.
old releases are still available and can be installed from repositories documented in
http://wiki.eclipse.org/M2E_updatesite_and_gittags
Please note that links below point at Eclipse p2 repositories; you must access them f
Eclipse (see how).
Update Sites
Latest m2e release (recommended)
http://download.eclipse.org/technology/m2e/releases
m2e milestone builds towards version 1.5
http://download.eclipse.org/technology/m2e/milestones/1.5
Latest m2e 1.5 SNAPSHOT build (not tested, not hosted at eclipse.org)

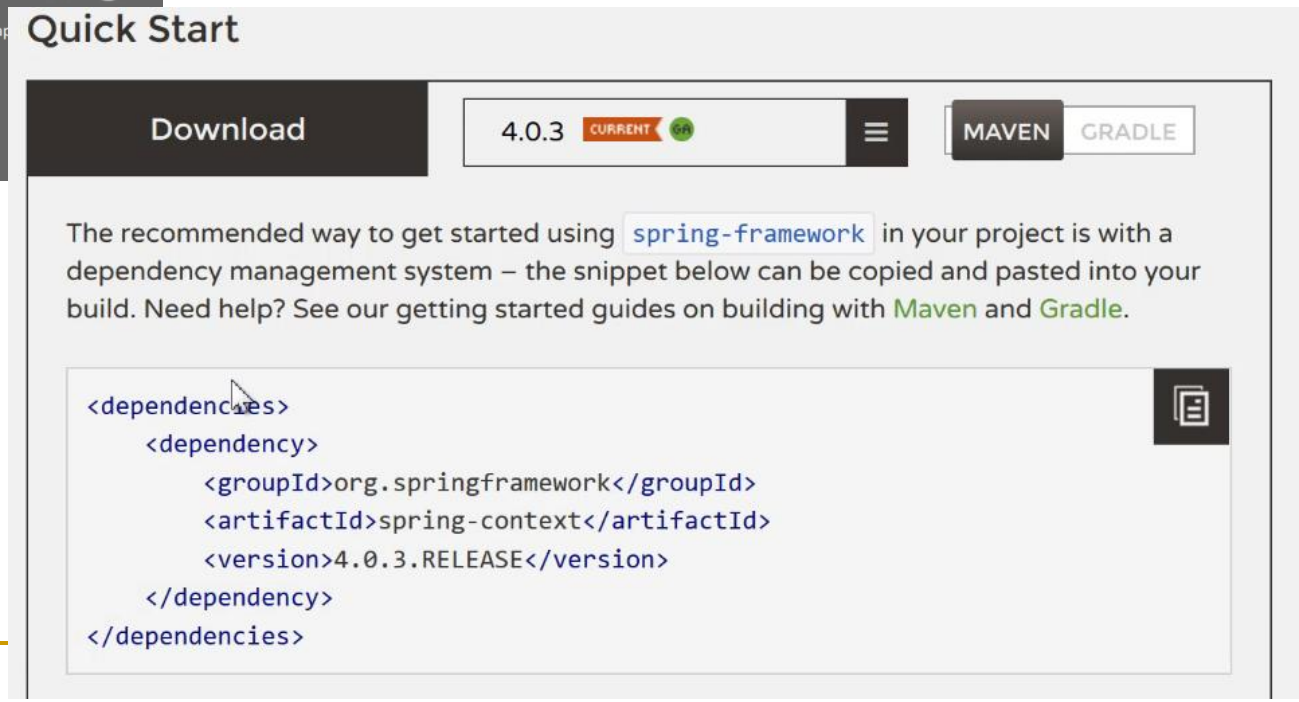

File Edit Navigate Search Project Run Window Help
Project Explorer
Welcome
Help Contents
Search
Dynamic Help
Key Assist... Ctrl+Shift+L
Tips and Tricks...
Report Bug or Enhancement...
Cheat Sheets...
Eclipse Marketplace...
Check for Updates
Install New Software...
About Eclipse

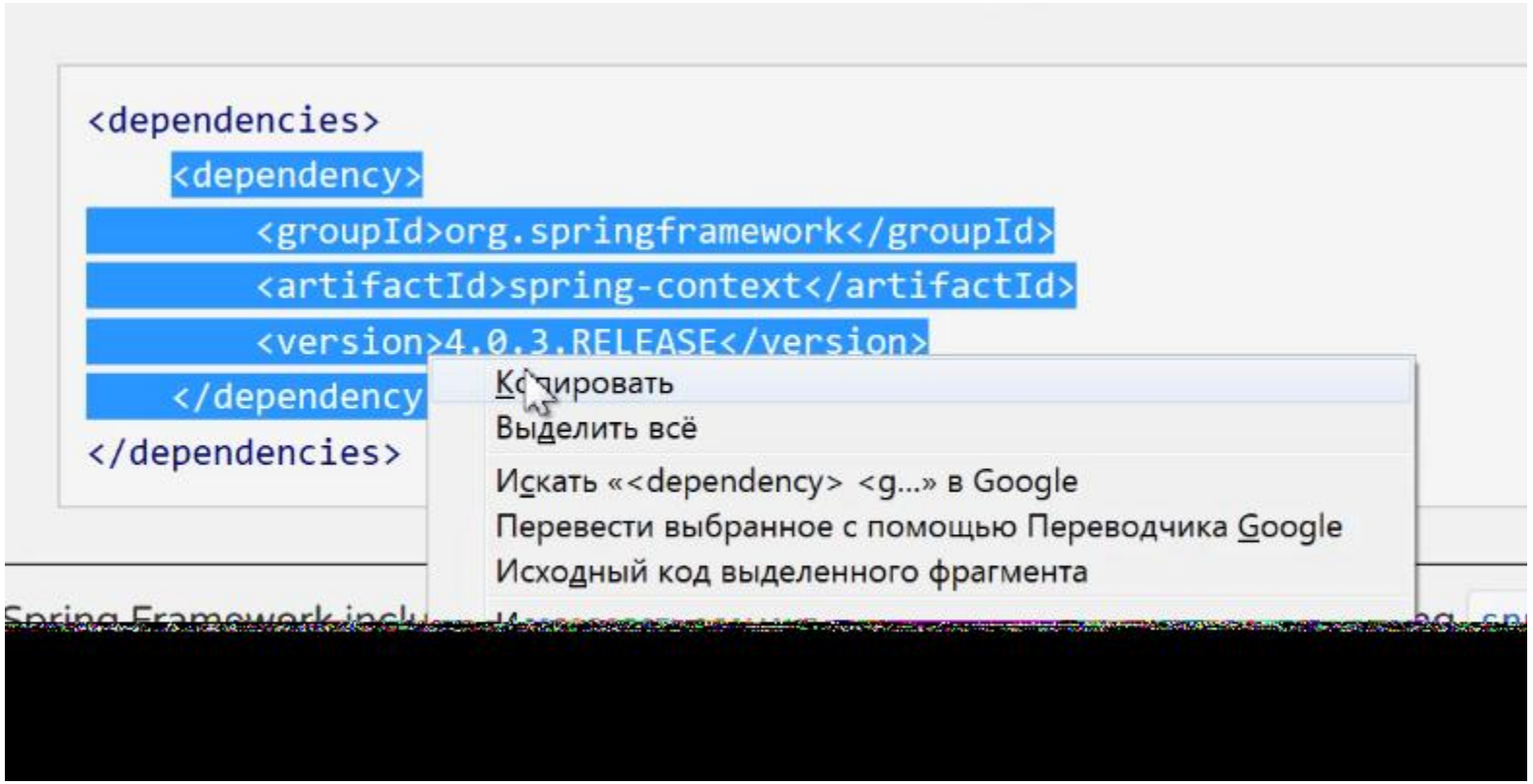

Available Software
Select a site or enter the location of a site.
Location must be absolute
Work with: maven
maven - http://download.eclipse.org/technology/m2e/releases
type filter text
Name
Available Software
Select a site or enter the location of a site.
Work with: maven
Find more software by working with the "Available
type filter text
Name
Version
There is no site sele
Add Repository
Name: maven
Local...
Location: http://download.eclipse.org/technology/m2e/releases
Archive...
OK
Cancel
Select All
Installing Software
Installing Software
Always run in background
Run in Background
Cancel
Details >>

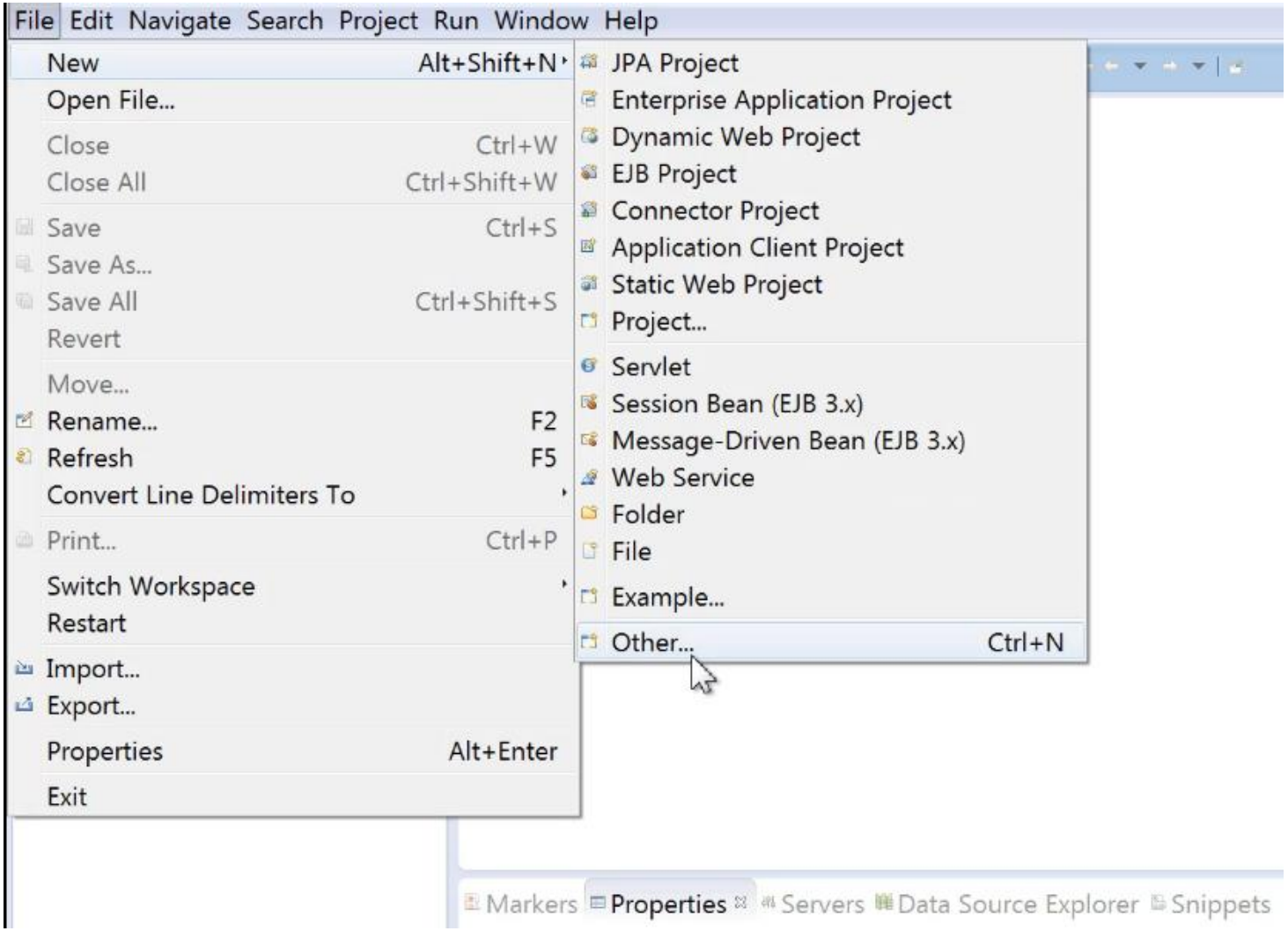
# Инструменты

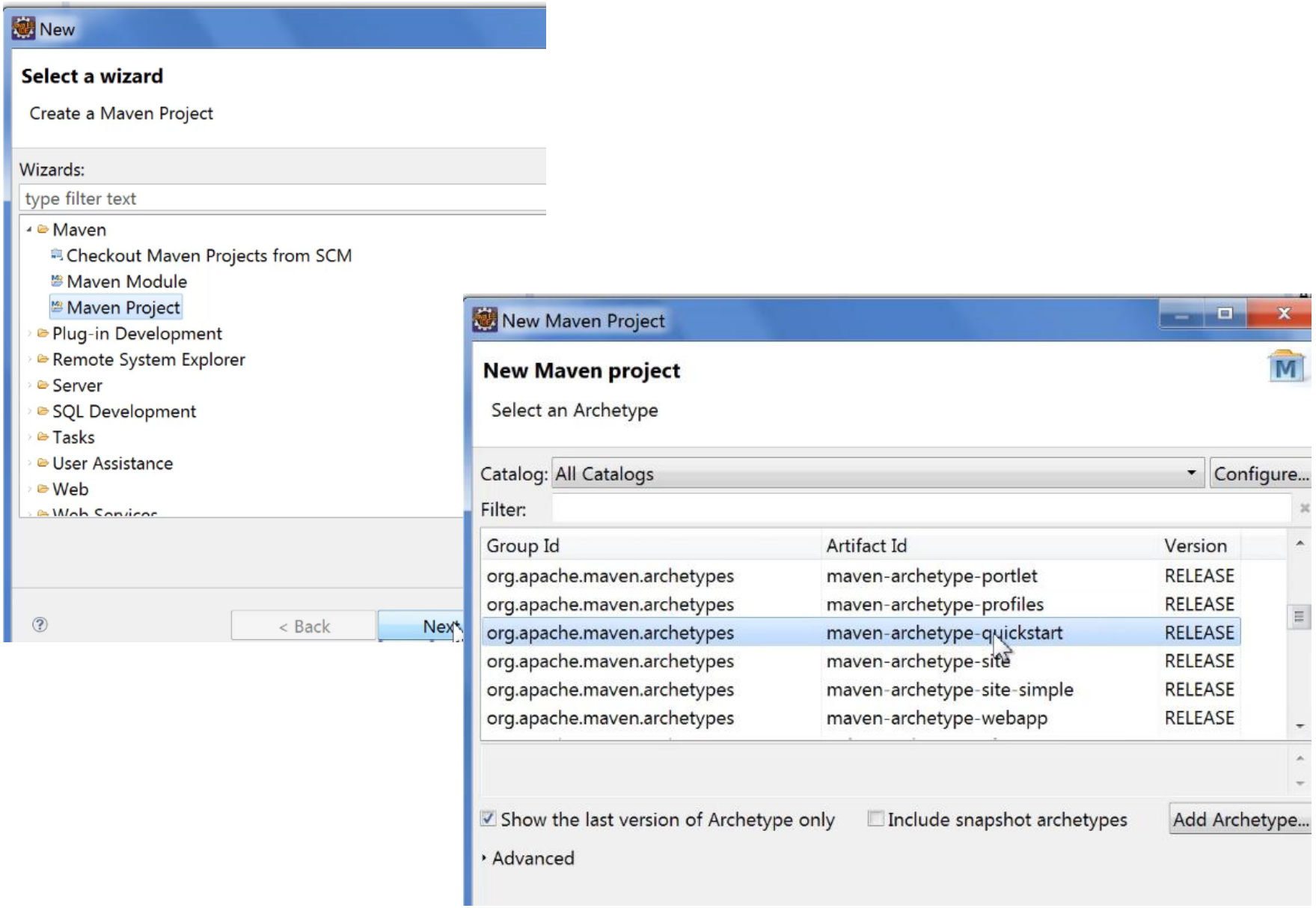
- Spring Framework http://projects.spring.io/spring-framework/

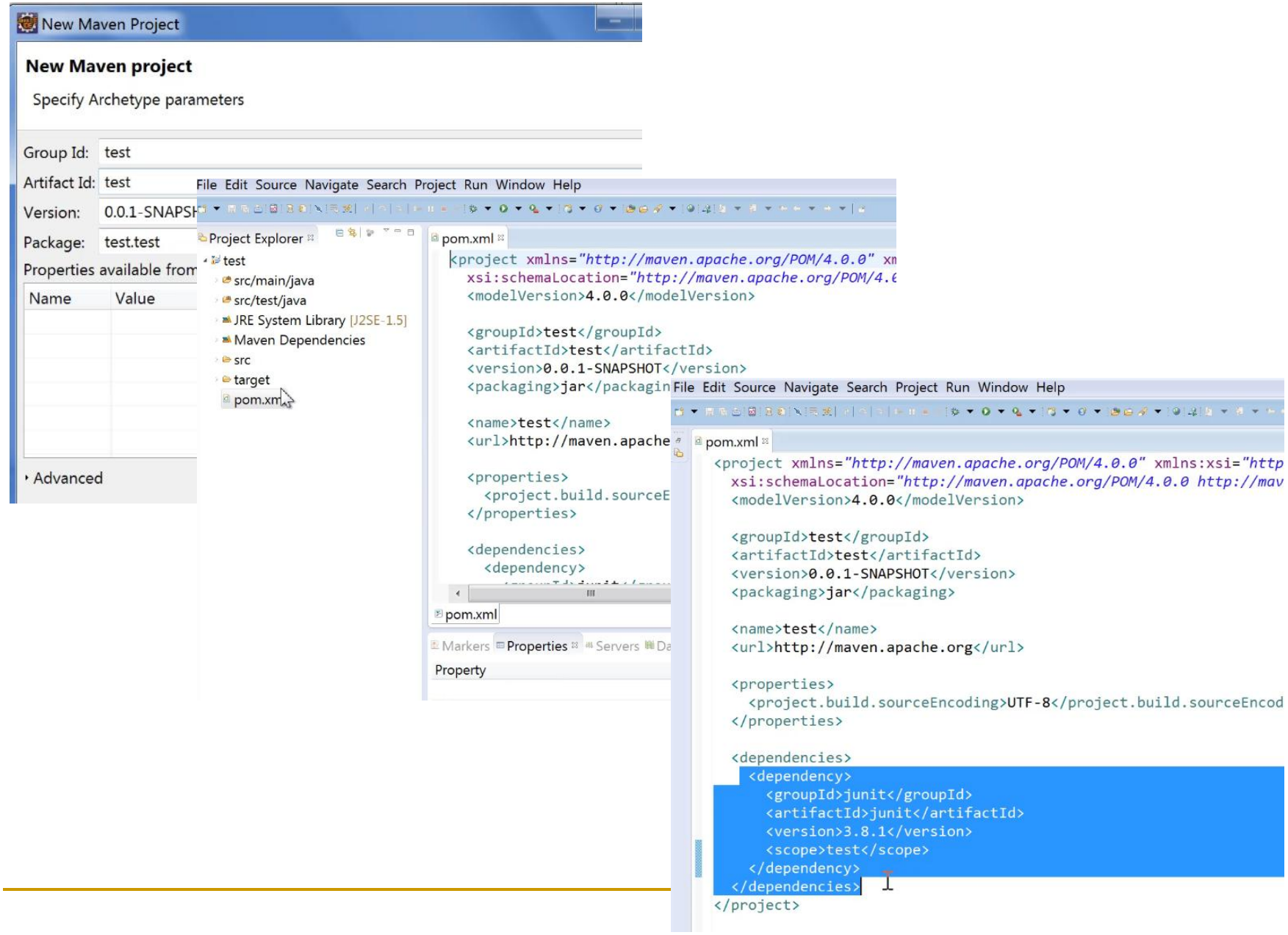


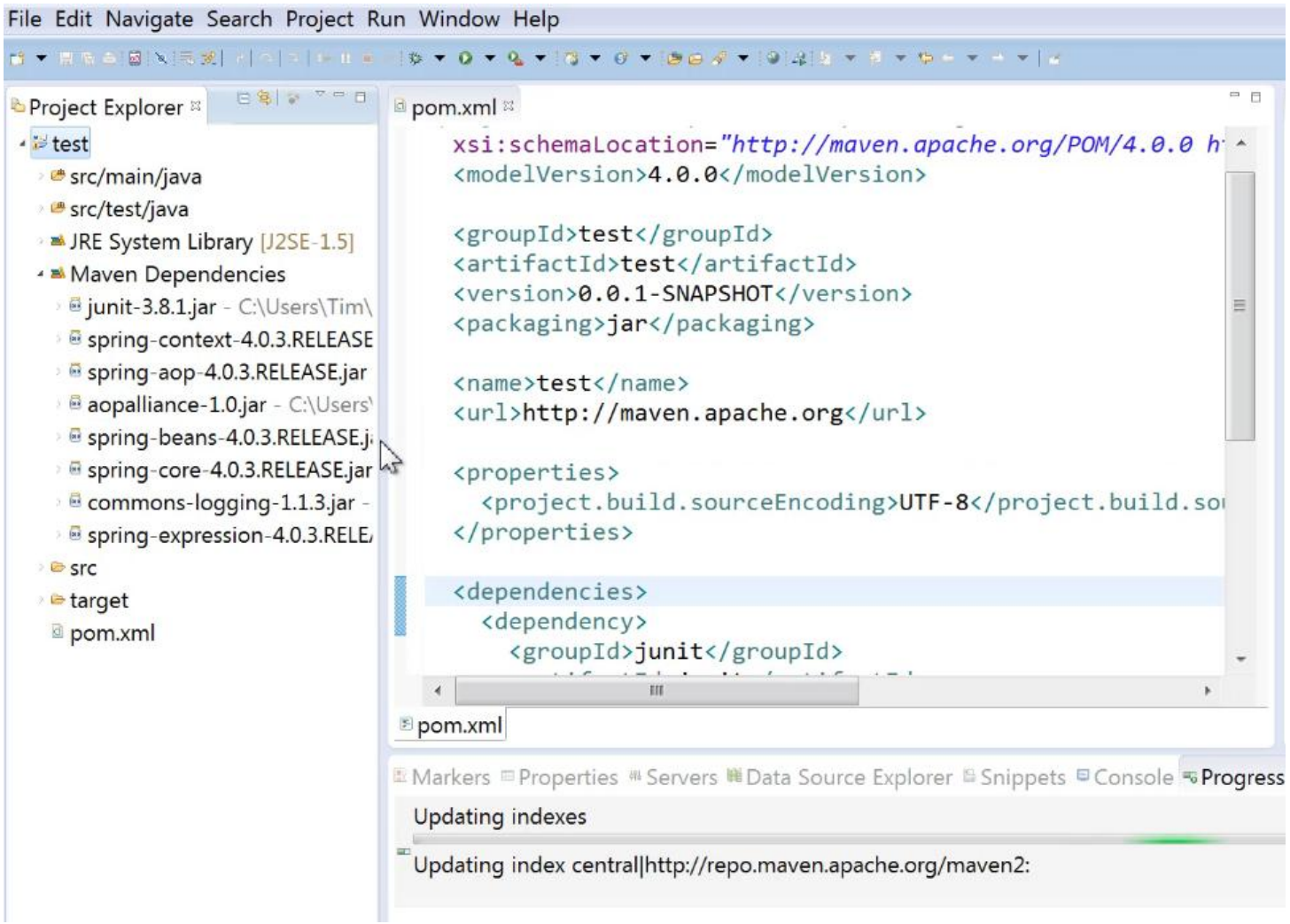

<dependencies>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>4.0.3.RELEASE</version>
</dependency>
</dependencies>
Копировать
Выделить всё
Искать «<dependency> <g...» в Google
Перевести выбранное с помощью Переводчика Google
Исходный код выделенного фрагмента

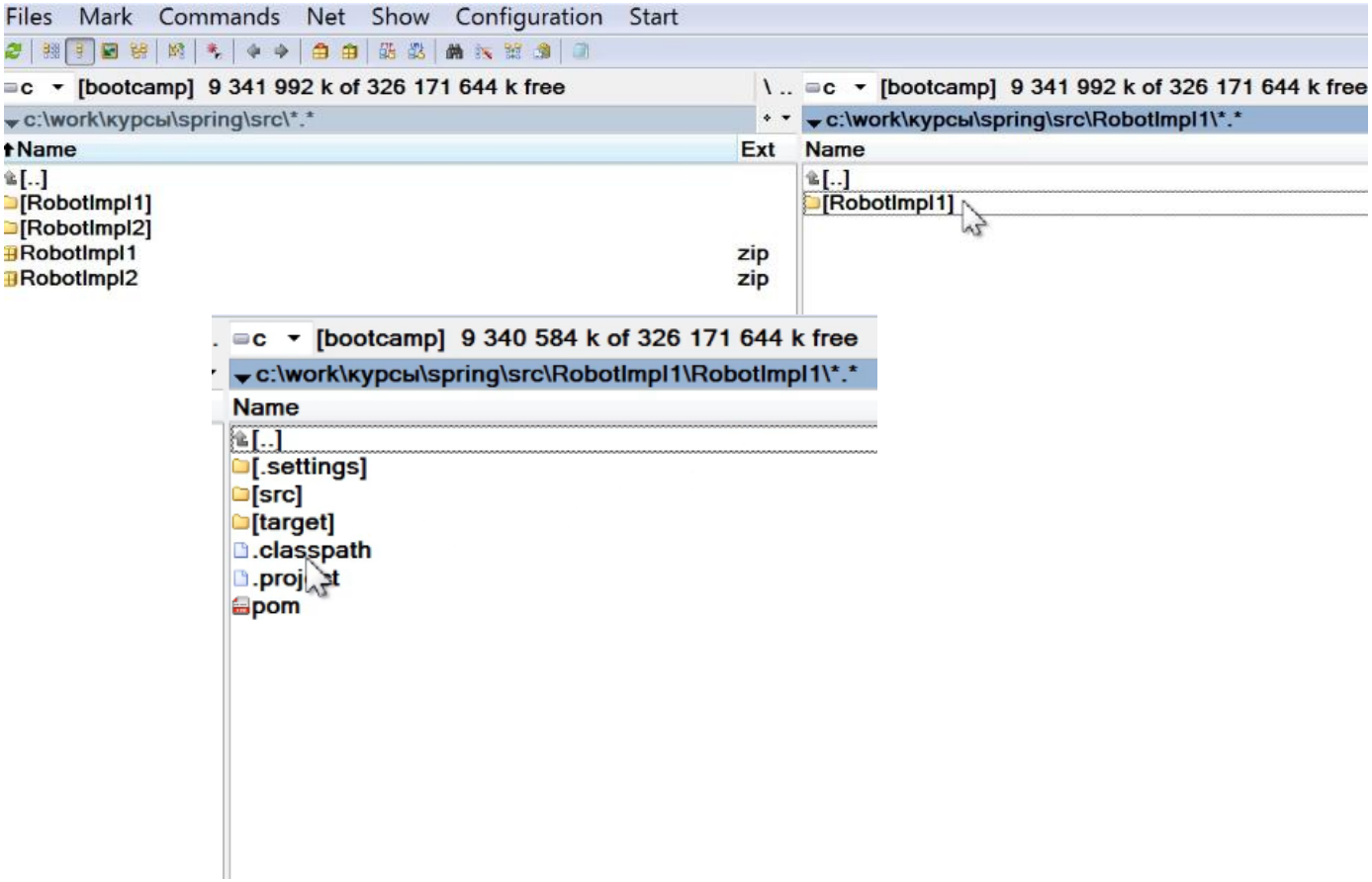

File Edit Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
Exit
JPA Project
Enterprise Application Project
Dynamic Web Project
EJB Project
Connector Project
Application Client Project
Static Web Project
Project...
Servlet
Session Bean (EJB 3.x)
Message-Driven Bean (EJB 3.x)
Web Service
Folder
File
Example...
Other... Ctrl+N
Markers
Properties
Servers
Data Source Explorer
Snippets

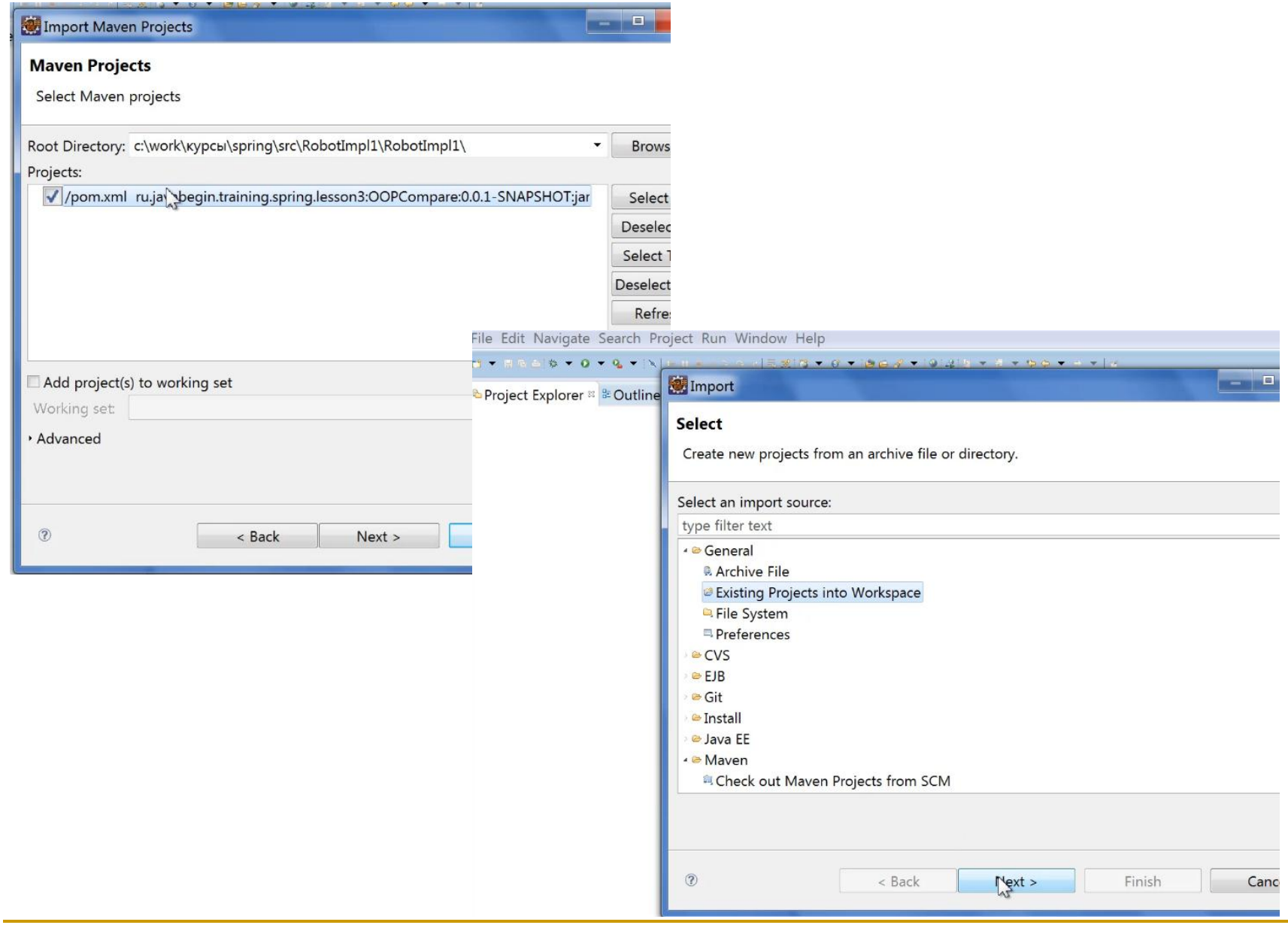

New
Select a wizard
Create a Maven Project
Wizards:
type filter text
Maven
Checkout Maven Projects from SCM
Maven Module
Maven Project
Plug-in Development
Remote System Explorer
Server
SQL Development
Tasks
User Assistance
Web
< Back
New Maven Project
New Maven project
Select an Archetype
Catalog: All Catalogs
Configure...
Filter:
Group Id
Artifact Id
Version
org.apache.maven.archetypes
maven-archetype-portlet
RELEASE
org.apache.maven.archetypes
maven-archetype-profiles
RELEASE
org.apache.maven.archetypes
maven-archetype-quickstart
RELEASE
org.apache.maven.archetypes
maven-archetype-site
RELEASE
org.apache.maven.archetypes
maven-archetype-site-simple
RELEASE
org.apache.maven.archetypes
maven-archetype-webapp
RELEASE
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced

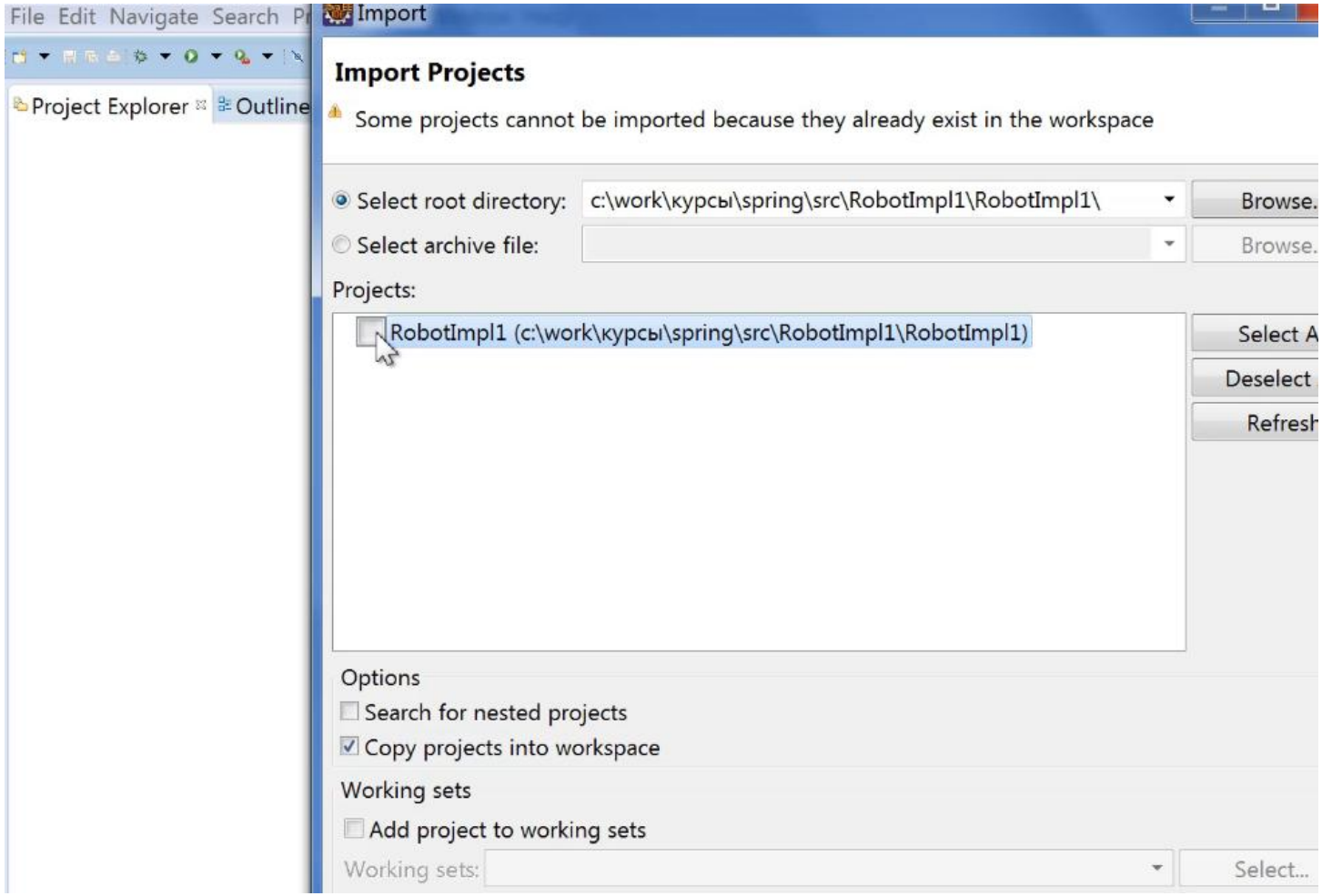

New Maven Project
New Maven project
Specify Archetype parameters
Group Id: test
Artifact Id: test
Version: 0.0.1-SNAPSH
Package: test.test
Properties available from
Name
Value
Advanced
File Edit Source Navigate Search Project Run Window Help
Project Explorer
test
src/main/java
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xm
xsi:schemaLocation="http://maven.apache.org/POM/4.0
<modelVersion>4.0.0</modelVersion>
<groupId>test</groupId>
<artifactId>test</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packagin
<name>test</name>
<url>http://maven.apache
<properties>
<project.build.sourceE
</properties>
<dependencies>
<dependency>
pom.xml
Markers Properties Servers
Property
File Edit Source Navigate Search Project Run Window Help
pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://mav
<modelVersion>4.0.0</modelVersion>
<groupId>test</groupId>
<artifactId>test</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>test</name>
<url>http://maven.apache.org</url>
<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncod
</properties>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>

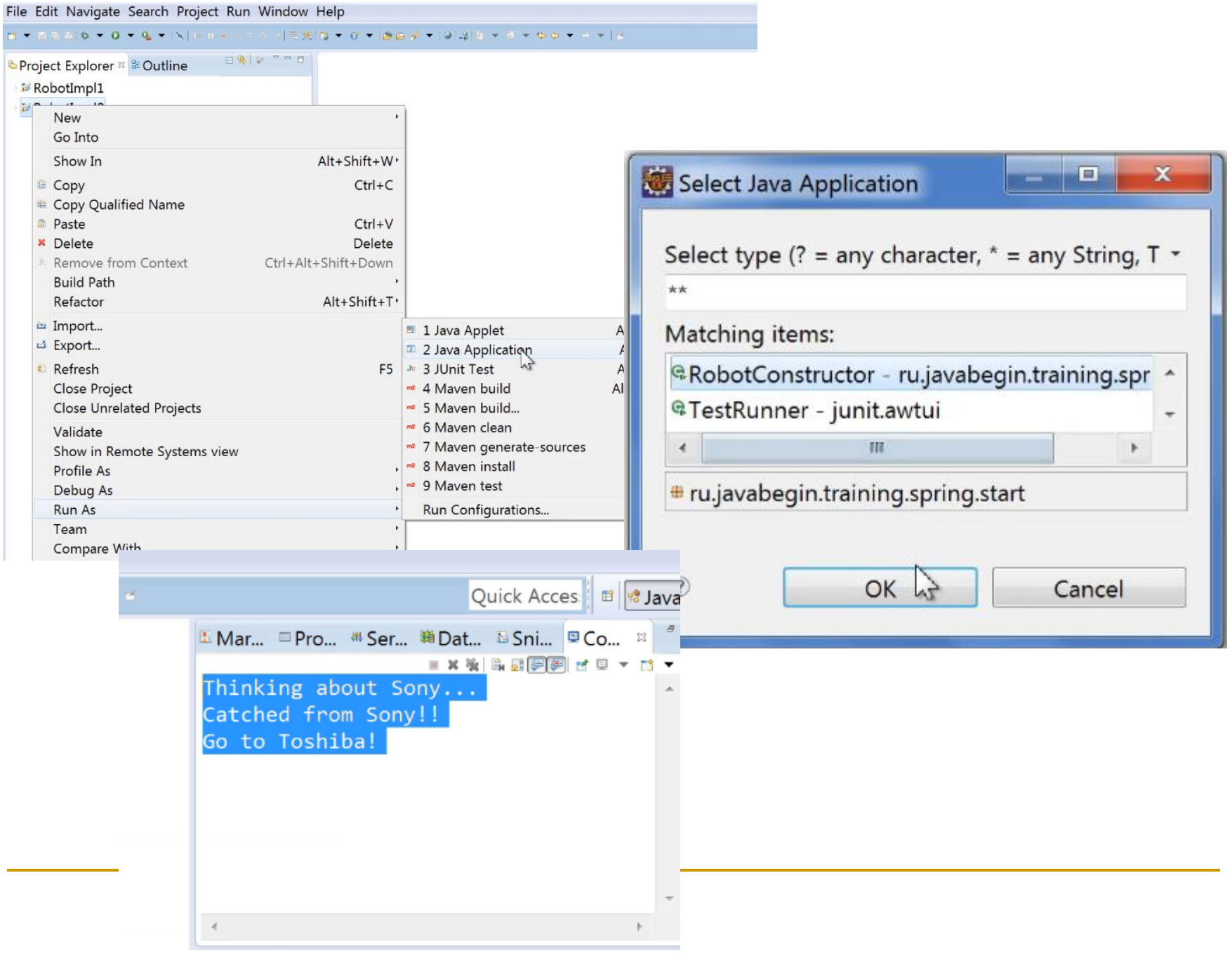

File Edit Source Navigate Search Project Run Window Help
pom.xml
xsi:schemaLocation="http://maven.apache.org/POM
<modelVersion>4.0.0</modelVersion>
<groupId>test</groupId>
<artifactId>test</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>test</name>
<url>http://maven.apache.org</url>
<properties>
<project.build.sourceEncoding>UTF-8</project.
</properties>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>4.0.3.RELEASE</version>


File Edit Navigate Search Project Run Window Help
Project Explorer
test
src/main/java
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
junit-3.8.1.jar - C:\Users\Tim\
spring-context-4.0.3.RELEASE
spring-aop-4.0.3.RELEASE.jar
aopalliance-1.0.jar - C:\Users\
spring-beans-4.0.3.RELEASE.j
spring-core-4.0.3.RELEASE.jar
commons-logging-1.1.3.jar -
spring-expression-4.0.3.RELE
src
target
pom.xml
pom.xml
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 h
<modelVersion>4.0.0</modelVersion>
<groupId>test</groupId>
<artifactId>test</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>test</name>
<url>http://maven.apache.org</url>
<properties>
<project.build.sourceEncoding>UTF-8</project.build.so
</properties>
<dependencies>
<dependency>
<groupId>junit</groupId>
pom.xml
Markers Properties Servers Data Source Explorer Snippets Console Progress
Updating indexes
Updating index central|http://repo.maven.apache.org/maven2:


Files Mark Commands Net Show Configuration Start
c [bootcamp] 9 341 992 k of 326 171 644 k free
c:\work\курсы\spring\src*.*
Name
Ext
[..]
[RobotImpl1]
[RobotImpl2]
RobotImpl1 zip
RobotImpl2 zip
c [bootcamp] 9 341 992 k of 326 171 644 k free
c:\work\курсы\spring\src\RobotImpl1*.*
Name
[..]
[RobotImpl1]
c [bootcamp] 9 340 584 k of 326 171 644 k free
c:\work\курсы\spring\src\RobotImpl1\RobotImpl1*.*
Name
[..]
[.settings]
[src]
[target]
.classpath
.project
pom


Import Maven Projects
Maven Projects
Select Maven projects
Root Directory: c:\work\курсы\spring\src\RobotImpl1\RobotImpl1\
Projects:
/pom.xml ru.ja begin.training.spring.lesson3:OOPCompare:0.0.1-SNAPSHOT:jar
Add project(s) to working set
Working set:
Advanced
< Back
Next >
File Edit Navigate Search Project Run Window Help
Project Explorer
Outline
Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
CVS
EJB
Git
Install
Java EE
Maven
Check out Maven Projects from SCM
< Back
Next >
Finish


File Edit Navigate Search Pr
Project Explorer
Outline
Import
Import Projects
Some projects cannot be imported because they already exist in the workspace
Select root directory:
c:\work\курсы\spring\src\RobotImpl1\RobotImpl1\
Browse.
Select archive file:
Browse.
Projects:
RobotImpl1 (c:\work\курсы\spring\src\RobotImpl1\RobotImpl1)
Select A
Deselect
Refresh
Options
Search for nested projects
Copy projects into workspace
Working sets
Add project to working sets
Working sets:
Select...

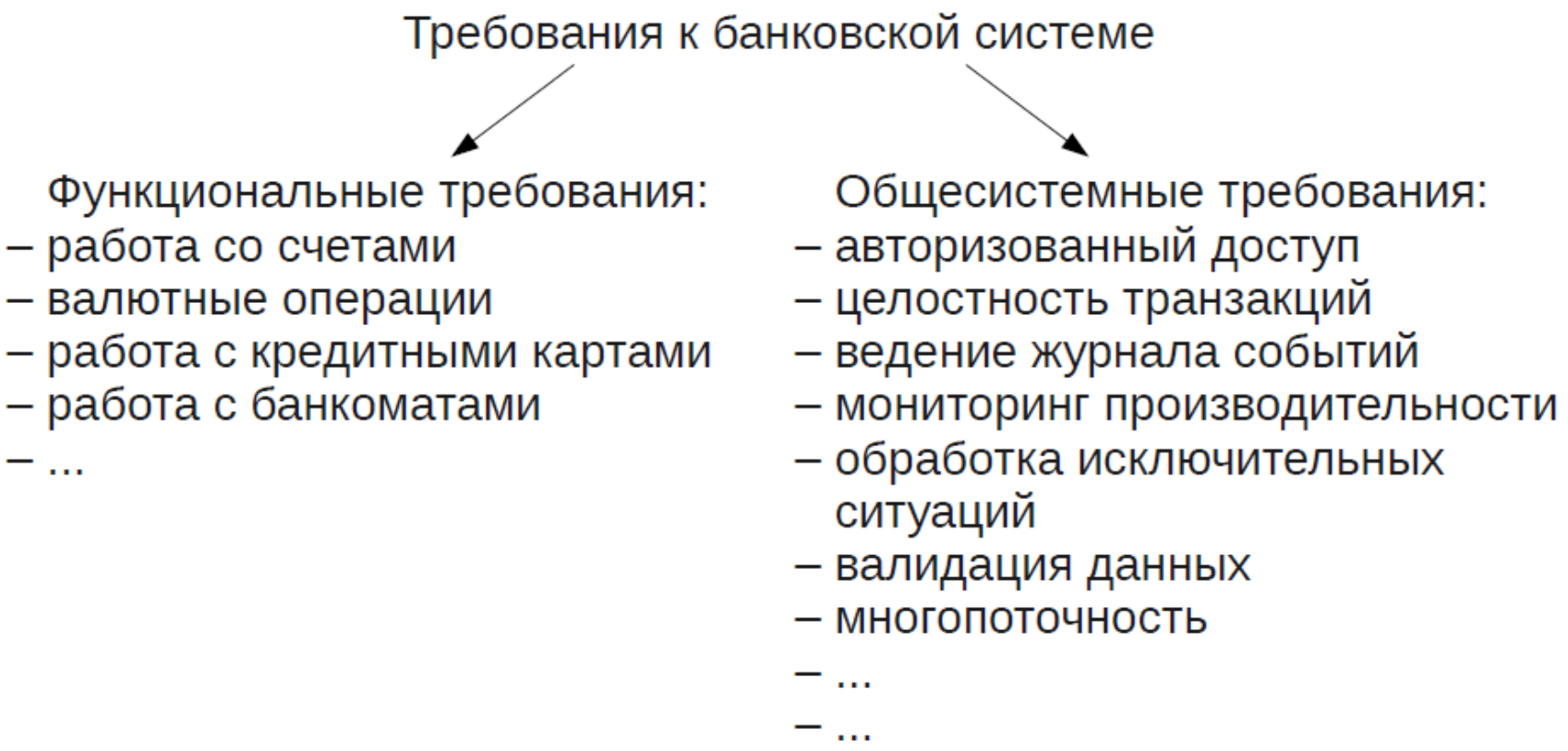

File Edit Navigate Search Project Run Window Help
Project Explorer
Outline
RobotImpl1
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Validate
Show in Remote Systems view
Profile As
Debug As
Run As
Team
Compare With
1 Java Applet
2 Java Application
3 JUnit Test
4 Maven build
5 Maven build...
6 Maven clean
7 Maven generate-sources
8 Maven install
9 Maven test
Run Configurations...
Select Java Application
Select type (? = any character, * = any String, T
**
Matching items:
RobotConstructor - ru.javabegin.training.spr
TestRunner - junit.awtui
ru.javabegin.training.spring.start
OK
Cancel
Quick Acces
Java
Mar...
Pro...
Ser...
Dat...
Sni...
Co...
Thinking about Sony...
Catched from Sony!!
Go to Toshiba!

*Аспектно-ориентированное подход (АОП)* — методология программирования, основанная на идее разделения функциональности для улучшения разбиения программы на модули.

АОП предлагает языковые средства, позволяющие выделять сквозную функциональность в отдельные модули, и таким образом упрощать работу с компонентами программной системы, и снижать сложность системы в целом.

Методология АОП была предложена группой инженеров исследовательского центра Xerox PARC под руководством Грегора Кичалеса (Gregor Kiczales). Ими же в 2001 году было разработано аспектно-ориентированное расширение для языка Java, получившее название AspectJ.

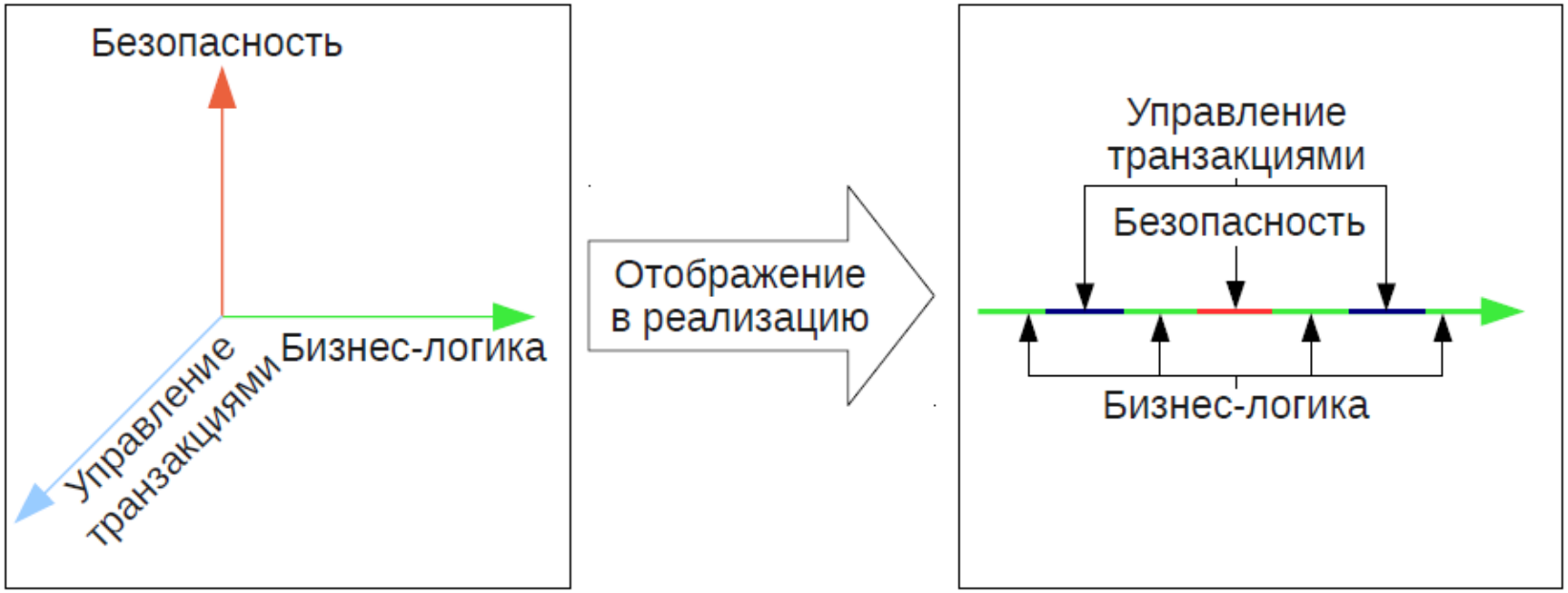
# Система как набор функциональных требований



# Проблема сквозной функциональности

```
public class SomeBusinessClass {
    // Данные, отвечающие за состояние объекта

    // Вспомогательные данные, отвечающие за:
    //   - авторизованный доступ к данным;
    //   - целостность транзакций;
    //   - ведение журнала событий;
    //   - безопасность междупоточного взаимодействия;
    //   - ...

    public void someOperation(Object...params) {
        // проверить уровень доступа к данным
        // запретить доступ к данным другим потокам выполнения
        // занести в журнал отметку о начале операции
            // Реализация логики данного метода
        // занести в журнал отметку о конце операции
        // разрешить доступ к данным другим потокам выполнения
    }

}
```

# Пространство требований и реализации

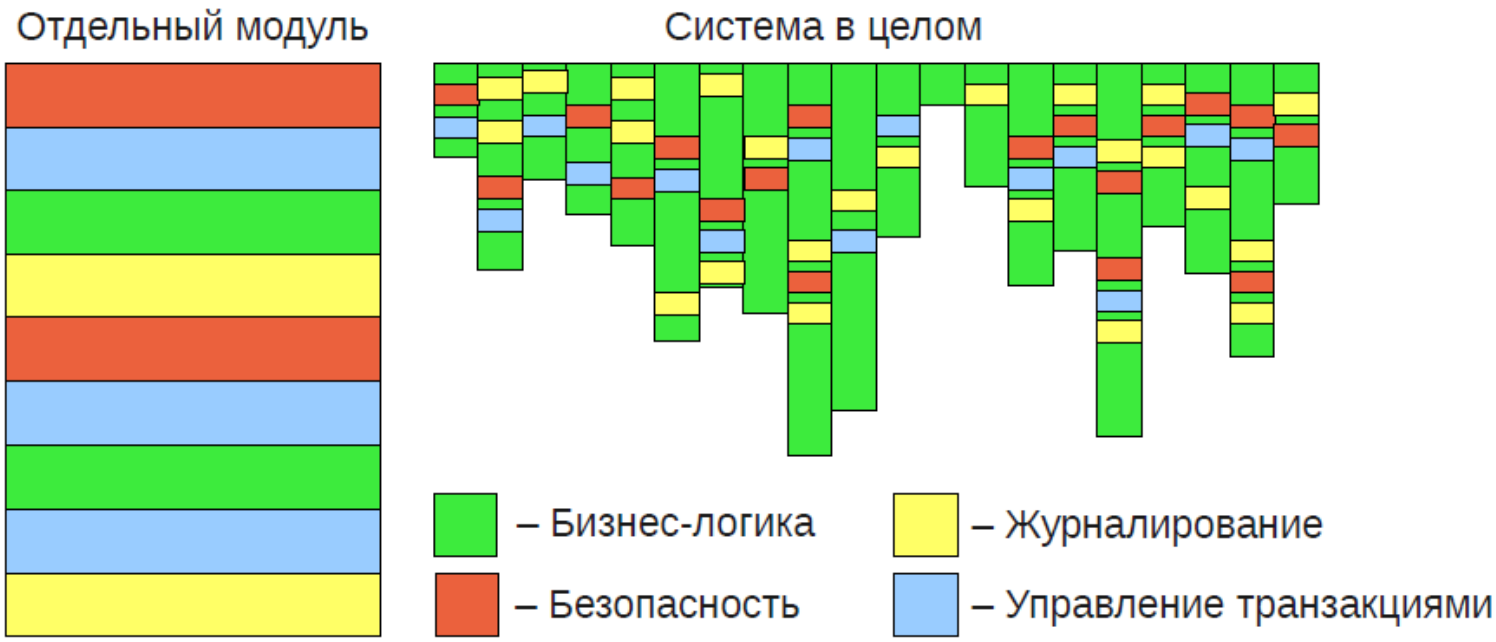



Пространство требований

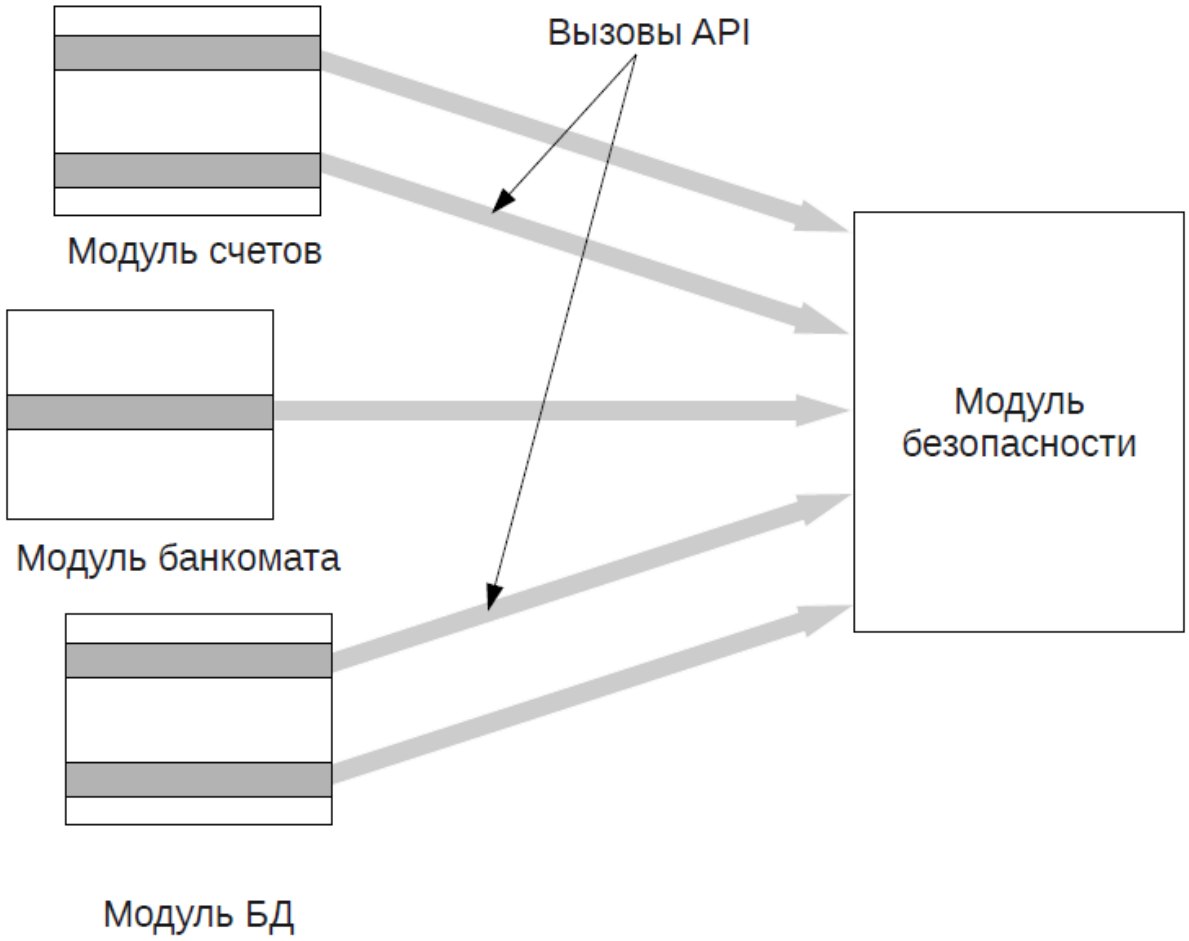
Пространство реализации

Сворачивание многомерного пространства требований
в одномерное пространство реализации

# Спутанный код



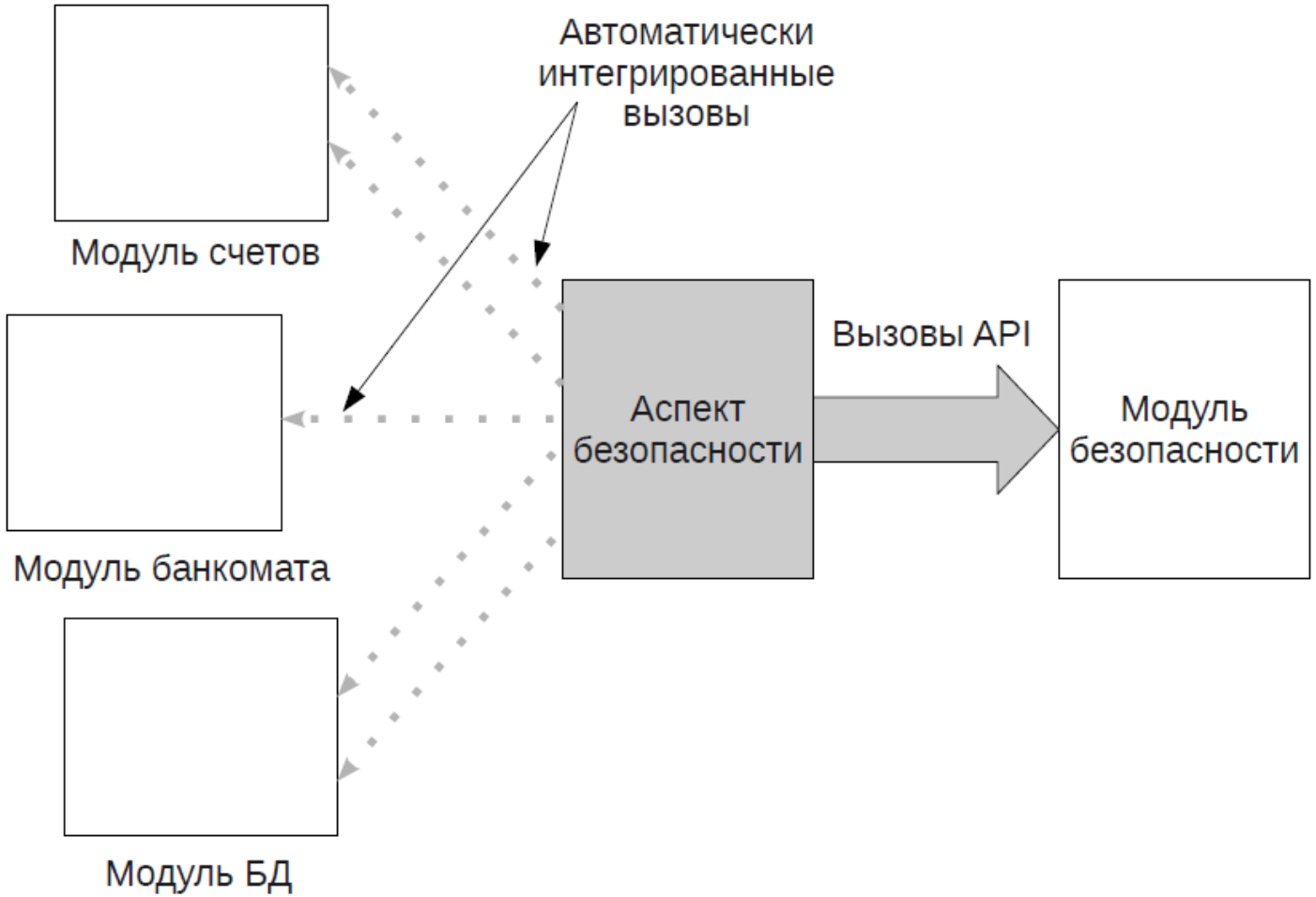
# Рассредоточенный код



# Влияние запутанного и рассредоточенного кода на проектирование и реализацию

– Плохое прослеживание назначения модуля
– Низкая пригодность кода для повторного использования
– Большая вероятность ошибок
– Трудности в сопровождении

# Выделение сквозной функциональности с помощь аспектов



# Фундаментальные понятия АОП

- **JoinPoint** — строго определённая точка выполнения программы, ассоциированная с контекстом выполнения (вызов метода, конструктора, доступ к полю класса, обработчик исключения, и т.д.)
- **Pointcut** — набор (срез) точек JoinPoint удовлетворяющих заданному условию.
- **Advice** — набор инструкций, выполняемых до, после или вместо каждой из точек выполнения (JoinPoint), входящих в заданный срез (Pointcut)
- **Inter-type declaration*** — способность аспекта изменять статическую структуру класса путем добавления как новых полей и методов, так и иерархии класса.
- **Aspect** — основная единица модульности АОП, инкапсулирующая срезы точек выполнения (Pointcut), наборы инструкци (Advice), а также Inter-type определения*.
- **Weaving** – процесс интеграции правил вплетения кода аспектов с кодом, ответственным за бизнес-логику, результатом которого является конечная система.

* – доступно только в AspectJ

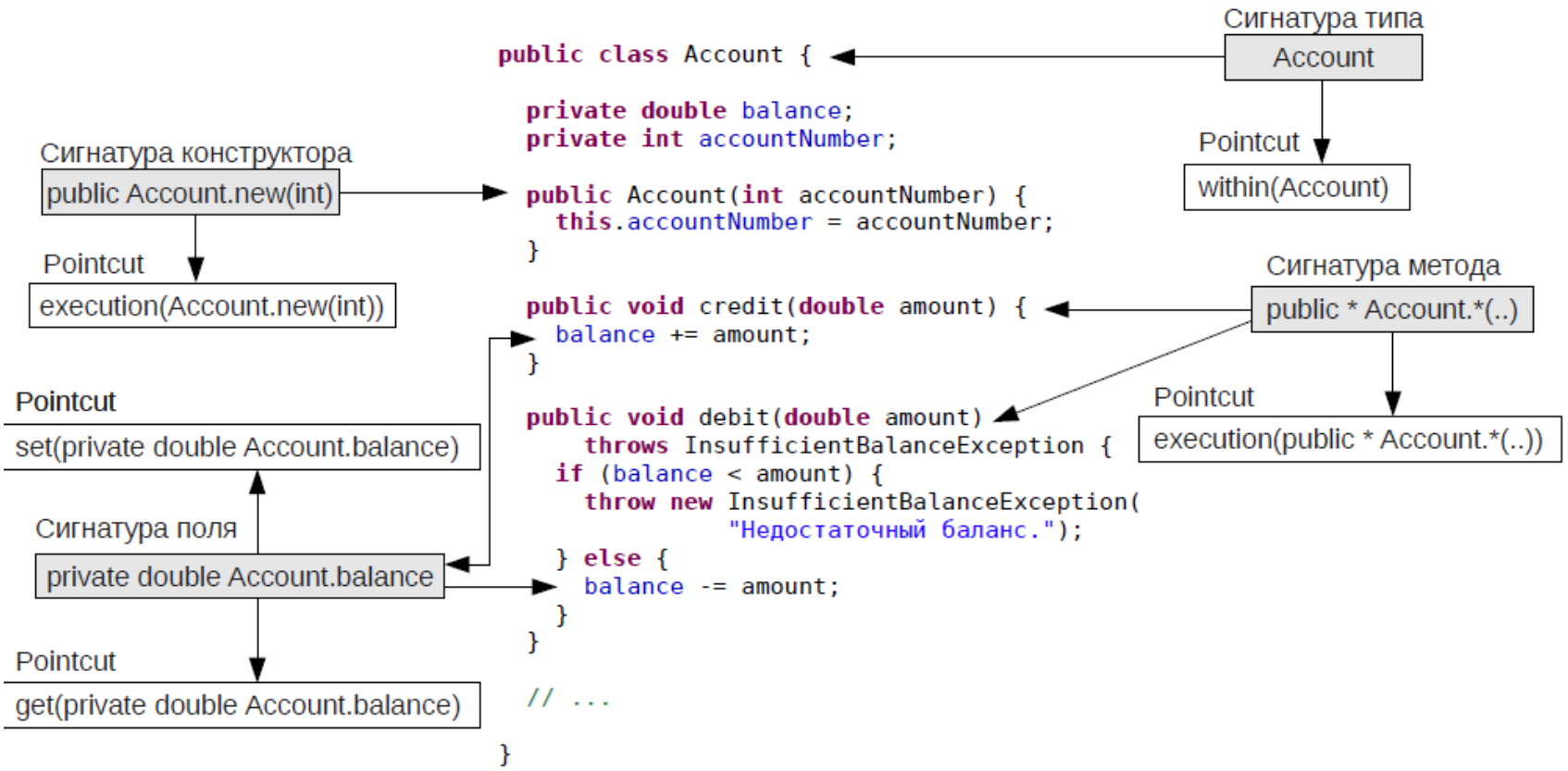
# Aspect. AspectJ

```
public aspect SomeAspect {

    pointcut allMethodCalls() : call(* *.*(..)) && !within(*Aspect);

    before() : allMethodCalls() {
        Logger.log("Before " + thisJoinPoint.toLongString());
    }

    after() : allMethodCalls() {
        Logger.log("Before " + thisJoinPoint.toLongString());
    }

    declare parents : Account implements Serializable;

}
```

# Joinpoints, доступные в AspectJ

| Категория | Преставление |
|---|---|
| Выполнение метода | Тело метода |
| Вызов метода | Вызов метода |
| Выполнение конструктора | Тело конструктора |
| Вызов конструктора | Вызов конструктора |
| Чтение поля класса | Чтение поля |
| Запись в поле класса | Запись поля |
| Обработка исключений | catch-блок обработки исключения |
| Инициализация класса | Блок статической инициализации |
| Инициализация объекта | Инициализация в конструкторе |

# Отношения между Joinpoints, шаблонами сигнатур и Pointcuts



# Pointcut. Мета-символы и логические операторы

| Символ | Пример использования |
|---|---|
| * | call(public *.new(int))<br>execution(public int *.get*()) |
| .. | call(public Account.new(..))<br>call(public int *.*(Account, .., double)) |
| + | execution(public Account+.new(..)) |

Мета-символы

| Операция | Пример использования |
|---|---|
| && | call(* debit(..)) && target(Account+) |
| \|\| | call(* int Account.*(..)) \|\| call(* double Account.*(..)) |
| ! | !whitin(Account) |

Логические операторы

# Pointcut. Лексические, типизированные и зависимые от аргументов

| Pointcut | Пример |
| --- | --- |
| within(<Синатура типа>) | within(Account) |
| withincode(<Синатура метода/конструктора>) | withincode(Account.new(..)) |
| this(<Синатура типа>) | call(* *(..)) && target(Account) |
| target(<Синатура типа>) | execution(* *(..)) && this(Account) |
| args(<Список аргументов>) | execution(* Account.*(double)) && args(amount) |

# Конструкция Advice

**Advice** – набор инструкций сквозной функциональности, добавляемый до, после или вместо JoinPoint, входящих в указанный Pointcut.

- Before – набор инструкций выполняемый перед выполнением инструкций входящих в описываемую Joinpoint.
- after returning – набор инструкций выполняется после успешного возвращения значения из описываемой точки выполнения.
- after throwing – набор инструкций выполняется после возникновения исключительной ситуации в описываемой точке выполнения.
- after – набор инструкций выполняется после возвращения из описываемой точки выполнения в любом случае.
- around – набор инструкций выполняется вместо описываемой точки выполнения.

# Advice. Пример

```
pointcut constructorCall(int accountNumber) :
    call(Account.new(int)) && args(accountNumber);

before(int accountNumber) : constructorCall(accountNumber) {
    System.out.println("Before calling. Caller signature "
            + thisEnclosingJoinPointStaticPart.getSignature());
}

after(int accountNumber) : constructorCall(accountNumber) {
    System.out.println("After calling.");
}
```

# Inter-type declaration

**Inter-type declaration** позволяют добавлять новые методы и переменные в классы, объявлять класс как реализацию интерфейса, устанавливать или отменять проверку исключений, определять предупреждения и ошибки компиляции, использовать примеси.

```
declare parents : Account implements Serializable;

declare @type: Account : @MyAnnotation;

private String Account.name;
private void Account.getName(String name) {
    this.name = name;
}
private String Account.getName() {
    return name;
}

declare warning : get(* System.out) && !within(Logger)
        : "Воспользуйтесь Logger.log()";
```

# Weaving

– модификация исходного кода с дальнейшей компиляцией (source-to-source weaving)
– модификация откомпилированного байт-кода (post-compile weaving)
– модификация байт-кода загружаемого JVM при старте приложения (load-time weaving)
– прямое использование паттерна Proxy

# @AspectJ

```
@Aspect
public class PureJavaAspect {

    @Pointcut(value = "execution(* Content.*(..))")
    public void parents() {}

    @Before("parents()")
    public void before(JoinPoint joinPoint) {
        Logger.log("Before " + joinPoint.toLongString());
    }

    @DeclareWarning("get(* System.*) && !within(cnc.logging.Logger)")
    static final String warning = "Do not use System.out";

}
```

# AspectJ. Средства разработки

Консольные:
– ajc - компилятор AspectJ
– ajdoc - средство документации AspectJ в стиле javadoc

Визуальные:
– плагин для Eclipse
– плагин для NetBeans
– плагин для IntelliJ IDEA

# Преимущества АОП

– Упрощение дизайна
– Более понятная реализация
– Уменьшение объёма исходного кода
– Уменьшение количества ошибок
– Уменьшение связанности между классами
– Улучшение повторного использования кода
– Уменьшение затрат на поддержку ПО
– Снижение стоимости разработки

# Недостатки АОП

– Порог вхождения
– Похоже на магию
– Некоторые недостатки реализации
– Некоторые недостатки средств  разработки
– Изучение существующих практик применения для снижения рисков

# Альтернативы АОП

- Фреймворки
- Кодогенерация
- Паттерны проектирования
  - Observer
  - Chain of Responsibility
  - Decorator
  - Proxy

# Языки программирования, имеющие реализации АОП

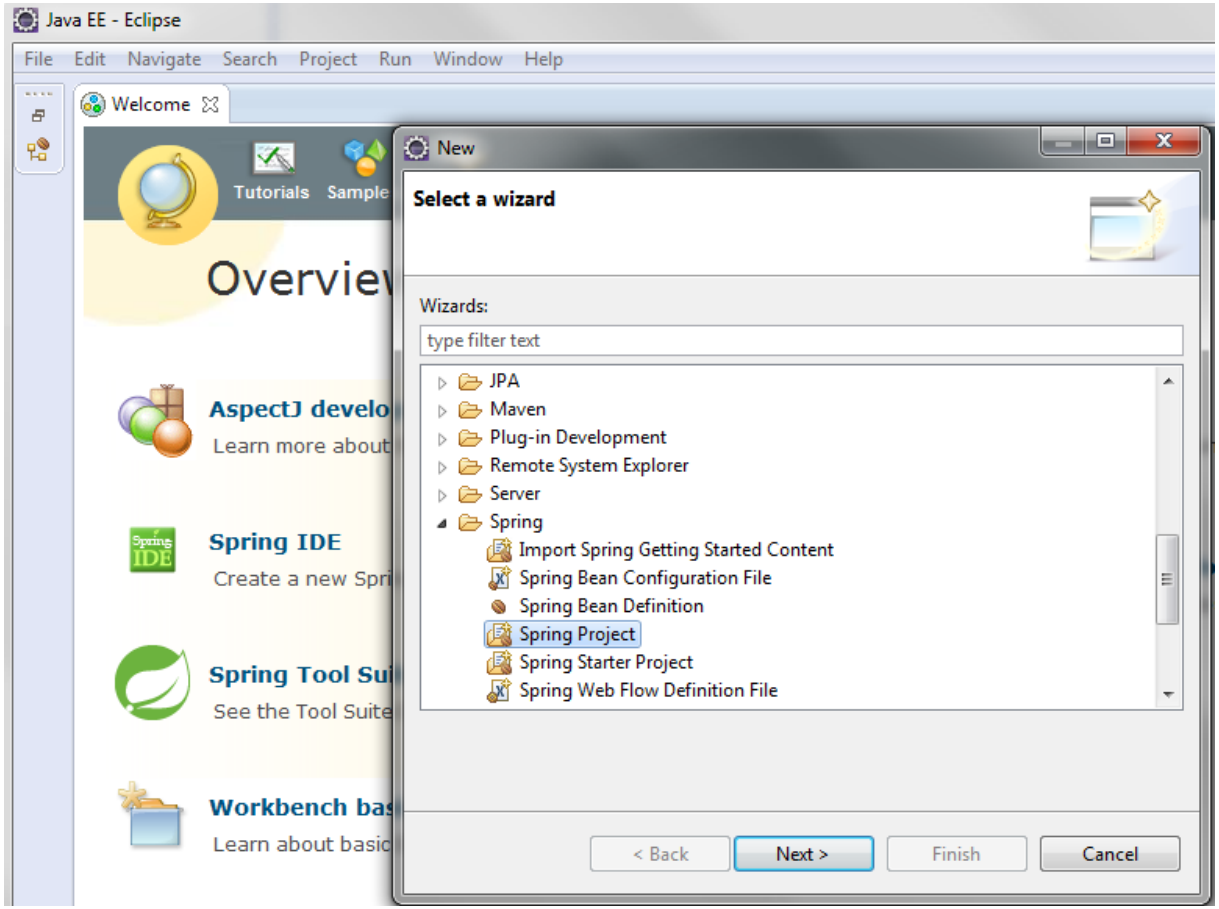
- Java
- .NET
- C/C++
- JavaScript
- Python
- Ruby
- PHP
- Perl
- XML
- ActionScript
- Common Lisp
- Object Pascal
- ColdFusion
- Lua
- Cocoa

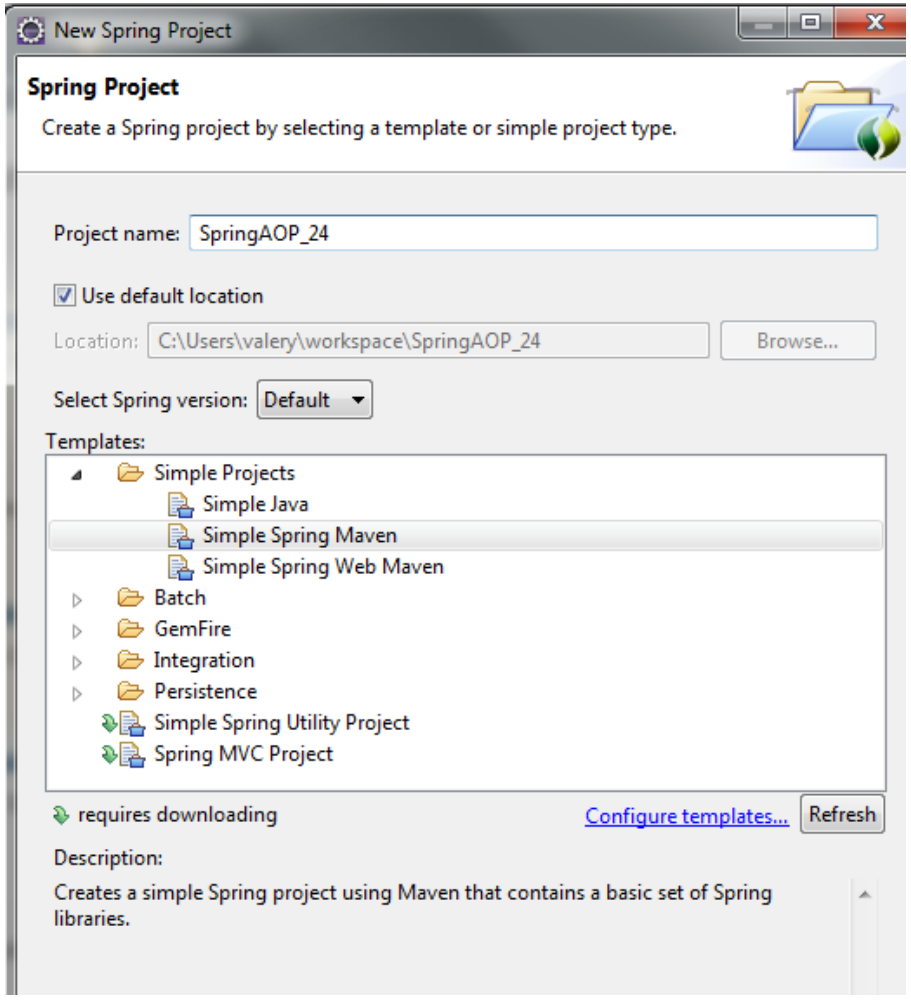
# Ссылки

http://www.javable.com/columns/aop/workshop/02
http://www.eclipse.org/aspectj/
http://www.eclipse.org/aspectj/doc/next/quick5.pdf
http://www.eclipse.org/aspectj/doc/released/progguide/index.html
https://www.ibm.com/developerworks/ru/views/java/libraryview.jsp?search_by=AOP@work

# Создание простого Spring примера

- создадим в **Eclipse** проект **Maven** и назовем его **SpringAOPExample**. В файле **pom.xml** будет содержать следующие зависимости:




New Spring Project
Spring Project
Create a Spring project by selecting a template or simple project type.
Project name: SpringAOP_24
Use default location
Location: C:\Users\valery\workspace\SpringAOP_24
Browse...
Select Spring version: Default
Templates:
Simple Projects
Simple Java
Simple Spring Maven
Simple Spring Web Maven
Batch
GemFire
Integration
Persistence
Simple Spring Utility Project
Spring MVC Project
requires downloading
Configure templates...
Refresh
Description:
Creates a simple Spring project using Maven that contains a basic set of Spring libraries.

Пример из жизни:

- Вы приходите в библиотеку и просите выдать вам книгу. В программе данное действие будет представлено функцией *getBook(String author, String book_name)*.
- Перед тем как выдать вам книгу, необходимо проверить, есть ли такая в наличии: ***checkBook(String author, String book_name)*.**
- Помимо этого было бы неплохо проверить, нет ли у вас задолженностей по книгам, ведь без возврата всех книг нельзя брать новые: ***checkReader(String reader_name)*.**
- Если все необходимые условия выполнены, то вам выдается книга. Однако после этого надо бы пометить, что данная книга находится теперь у вас на руках: ***booked(String author, String book_name, String reader_name)*.**

# Основные понятия:

- *Аспект* (*aspect*) — модуль или класс, реализующий сквозную функциональность. Если в ООП базовым элементом является класс, то в АОП — это аспект;
- *Точка соединения* (*join point*) — определяется как любая логическая точка в процессе выполнения программы, где встречаются основная программа и аспект. В **Spring AOP** точка соединения всегда соответствует вызову метода;

# Напоминалка

Для языка **Java** парадигма АОП реализуется с помощью такого фреймворка, как **Spring AOP** , который заключает всю сквозную функциональность в аспекты. Проще говоря, он способен улавливать выполнение какого-либо метода и добавлять до или после него выполнение других методов. Делается это с помощью **Advice**(совет, рекомендация). В **Spring AOP** есть 4 вида рекомендаций:

- Рекомендация *before* — запускается до выполнения метода;
- Рекомендация *after* — запускается после выполнения метода;
- Рекомендация *throws* — выполняется после того, как метод выбросит исключение;
- Рекомендация *around* — окружает точку соединения. Объединяет в себе три вышеперечисленные рекомендации;

# Добавляем в проект зависимости

```
    <dependency>
        <groupId>cglib</groupId>
        <artifactId>cglib</artifactId>
        <version>2.2.2</version>
    </dependency>
</dependencies>
oject>
```

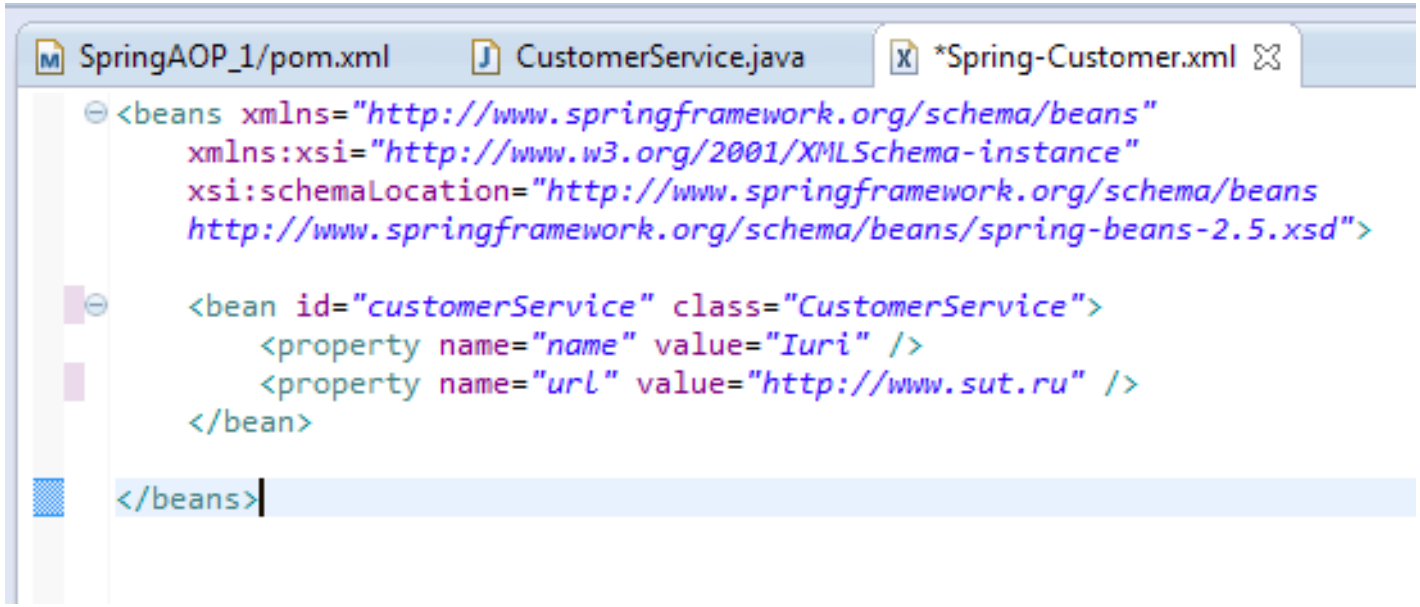
Dependencies | Dependency Hierarchy | Effective POM | pom.xml

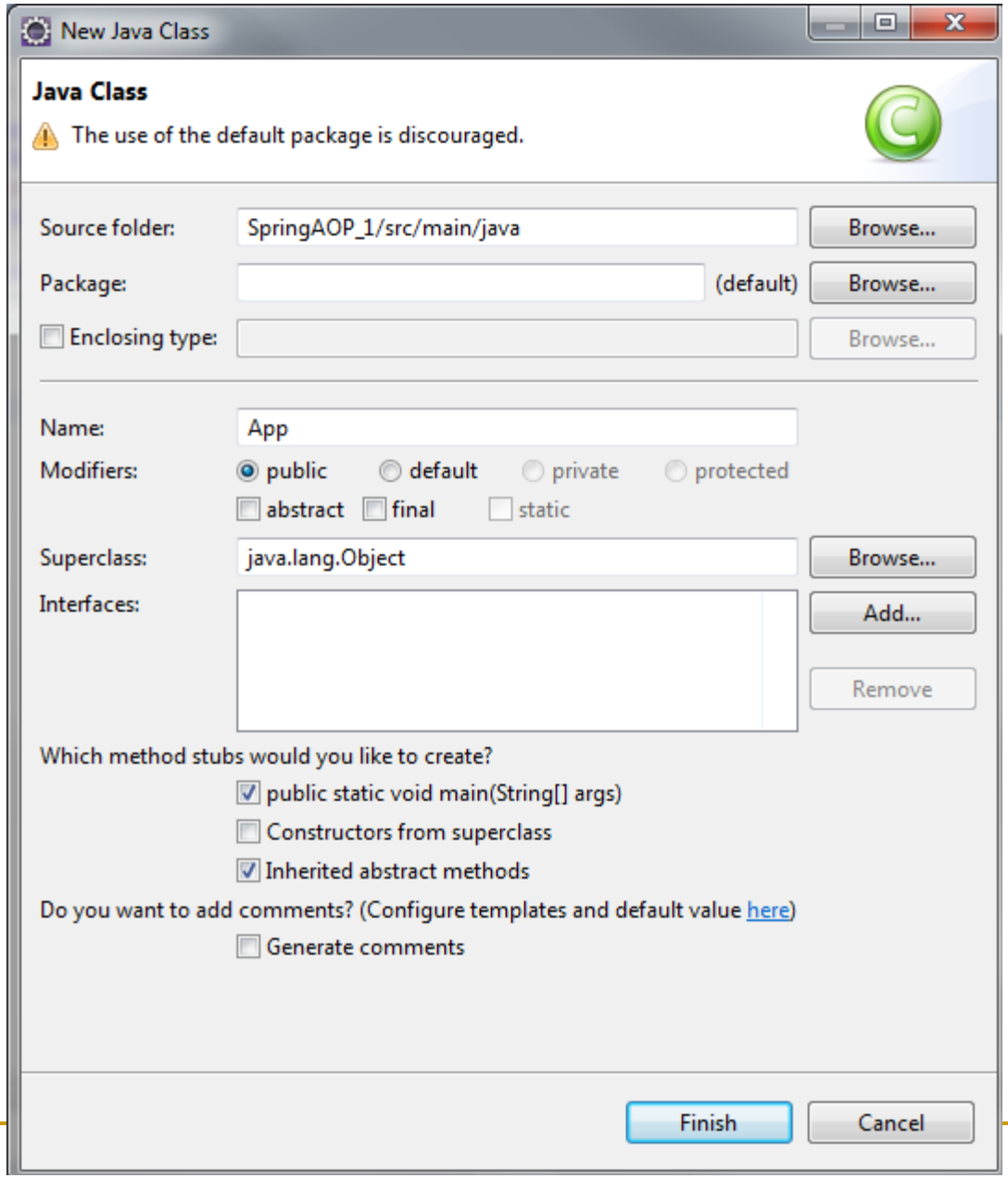
# Создаём класс сервисов пользователя

SpringAOP_1/pom.xml | CustomerService.java

```
public class CustomerService {
    private String name;
    private String url;

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

    public void setUrl(String url) {
        this.url = url;
    }

    public void printName() {
        System.out.println("Customer name : " + this.name);
    }

    public void printURL() {
        System.out.println("Customer website : " + this.url);
    }

    public void printThrowException() {
        throw new IllegalArgumentException();
    }

}
```

# помещаем конфигурационный файл **Spring** с именем **Spring-Customer.xml**




New Java Class
Java Class
The use of the default package is discouraged.
Source folder:
SpringAOP_1/src/main/java
Browse...
Package:
(default)
Browse...
Enclosing type:
Browse...
Name:
App
Modifiers:
public
default
private
protected
abstract
final
static
Superclass:
java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel

```java
public class App {
    public static void main(String[] args) {
        ApplicationContext appContext = new ClassPathXmlApplicationContext(
                new String[] { "Spring-Customer.xml" });

        CustomerService cust = (CustomerService) appContext
                .getBean("customerServiceProxy");

        System.out.println("*************************");
        cust.printName();
        System.out.println("*************************");
        cust.printURL();
        System.out.println("*************************");
        try {
            cust.printThrowException();
        } catch (Exception e) {

        }

    }
}
```

# Вывод программы

```
**************************
Customer name : Iuri
**************************
Customer website : http://www.sut.ru
**************************
```

# Spring AOP Advices (Рекомендации)
# Рекомендация before

```
import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;

public class BeforeMethod implements MethodBeforeAdvice {
    @Override
    public void before(Method method, Object[] args, Object target)
            throws Throwable {
        System.out.println("BeforeMethod : До метода!");
    }
}
```

В конфигурационном файле **Spring** (**Spring-Customer.xml**) создаем бин для класса **BeforeMethod**, а также новый объект с именем **customerServiceProxy**:

```
<bean id="BeforeMethodBean" class="ru.aop.BeforeMethod" />

<bean id="customerServiceProxy"
            class="org.springframework.aop.framework.ProxyFactoryBean">

    <property name="target" ref="customerService" />

    <property name="interceptorNames">
        <list>
            <value>BeforeMethodBean</value>
        </list>
    </property>
</bean>
```

Свойство с именем *target* определяет бин класса, с которым мы будем работать. Свойство с именем *interceptorNames* определяет какие классы (рекомендации) будут работать с классом, находящемся в свойстве *target*. Теперь при запуске программы вы увидите следующее:

# Вывод программы

```
**************************
BeforeMethod : До метода!
Customer name : Iuri
**************************
BeforeMethod : До метода!
Customer website : http://www.sut.ru
**************************
BeforeMethod : До метода!
```

до выполнения каждого метода класса **CustomerService** выполняется метод *before* рекомендации **BeforeMethod**

# Рекомендация after

```
import java.lang.reflect.Method;
import org.springframework.aop.AfterReturningAdvice;

public class AfterMethod implements AfterReturningAdvice {
    @Override
    public void afterReturning(Object returnValue, Method method,
            Object[] args, Object target) throws Throwable {
        System.out.println("AfterMethod : После метода!");
    }
}
```

# Конфигурационный файл **Spring-Customer.xml**:

```
<bean id="AfterMethodBean" class="ru.aop.AfterMethod" />

<bean id="customerServiceProxy"
          class="org.springframework.aop.framework.ProxyFactoryBean">

    <property name="target" ref="customerService" />

    <property name="interceptorNames">
        <list>
            <value>AfterMethodBean</value>
        </list>
    </property>
</bean>
```

```
*************************
Customer name : Iuri
AfterMethod : После метода!
*************************
Customer website : http://www.sut.ru
AfterMethod : После метода!
*************************
```

# Рекомендация throws

Выполняется после того, как метод *выбросит исключение*.

```java
import org.springframework.aop.ThrowsAdvice;

public class ThrowException implements ThrowsAdvice {
    public void afterThrowing(IllegalArgumentException e) throws Throwable {
        System.out.println("ThrowException : После выброса исключения!");
    }
}
```

```xml
<bean id="ThrowExceptionBean" class="ru.aop.ThrowException" />

<bean id="customerServiceProxy"
            class="org.springframework.aop.framework.ProxyFactoryBean">

    <property name="target" ref="customerService" />

    <property name="interceptorNames">
        <list>
            <value>ThrowExceptionBean</value>
        </list>
    </property>
</bean>
```

```
*************************
Customer name : Iuri
*************************
Customer website : http://www.sut.ru
*************************
ThrowException : После выброса исключения!
```

# Рекомендация around

Сочетает в себе три вышеприведенных рекомендации и выполняется во время выполнения метода.

```java
import java.util.Arrays;

import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class AroundMethod implements MethodInterceptor {
    @Override
    public Object invoke(MethodInvocation methodInvocation) throws Throwable {

        System.out.println("Method name : "
                + methodInvocation.getMethod().getName());
        System.out.println("Method arguments : "
                + Arrays.toString(methodInvocation.getArguments()));

        // до метода
        System.out.println("AroundMethod : Вместо BeforeMethod!");

        try {
            // выполняем оригинальный метод
            Object result = methodInvocation.proceed();

            // после метода
            System.out.println("AroundMethod : Вместо AfterMethod!");

            return result;

        } catch (IllegalArgumentException e) {
            // если был выброс исключения
            System.out
                    .println("AroundMethod : Вместо ThrowMethod!");
            throw e;
        }
    }
}
```

# Конфигурационный файл **Spring-Customer.xml**

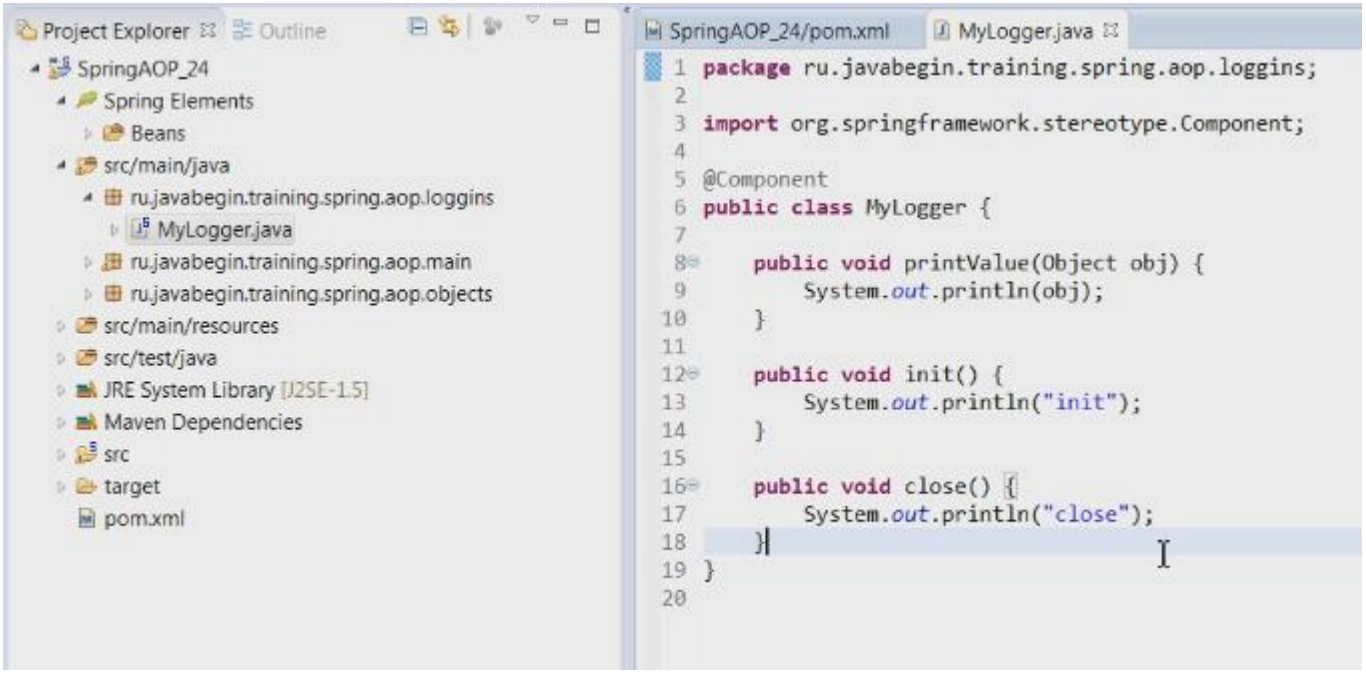
```
<bean id="AroundMethodBean" class="ru.aop.AroundMethod" />

<bean id="customerServiceProxy" class="org.springframework.aop.framework

    <property name="target" ref="customerService" />

    <property name="interceptorNames">
        <list>
            <value>AroundMethodBean</value>
        </list>
    </property>
</bean>
```

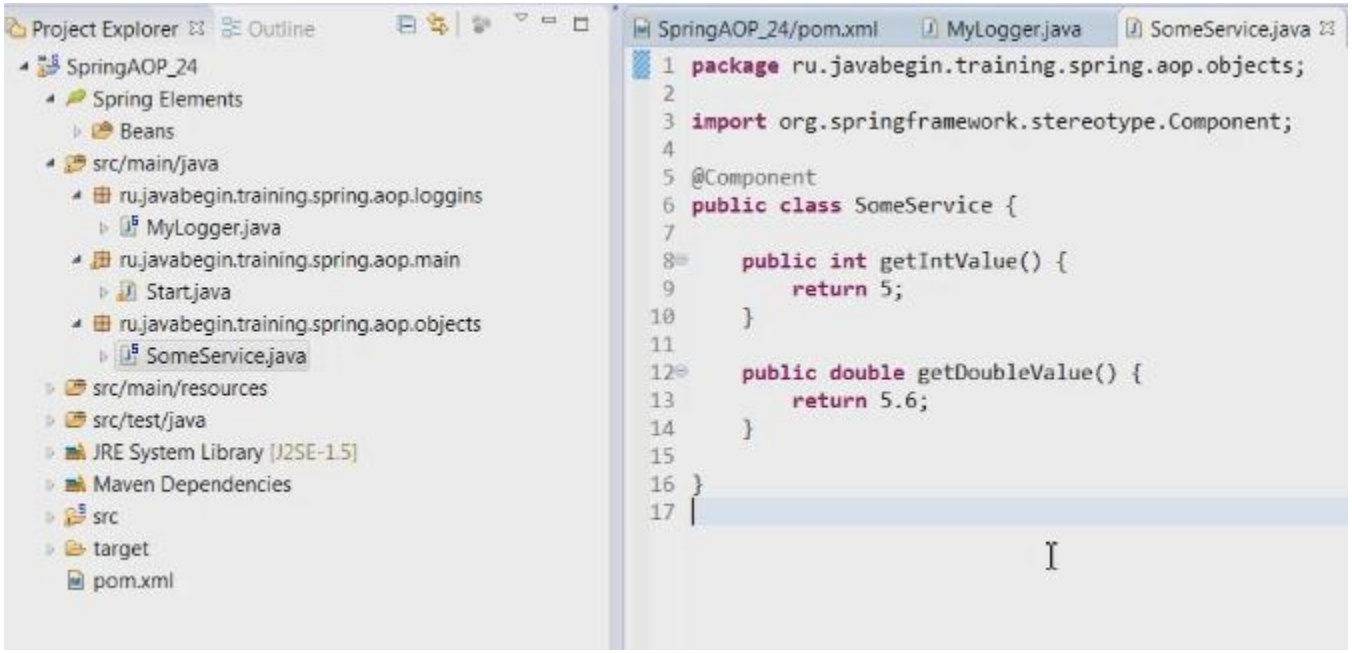
```
*************************
Method name : printName
Method arguments : []
AroundMethod : Вместо BeforeMethod!
Customer name : Iuri
AroundMethod : Вместо AfterMethod!
*************************
Method name : printURL
Method arguments : []
AroundMethod : Вместо BeforeMethod!
Customer website : http://www.sut.ru
AroundMethod : Вместо AfterMethod!
*************************
Method name : printThrowException
Method arguments : []
AroundMethod : Вместо BeforeMethod!
AroundMethod : Вместо ThrowMethod!
```

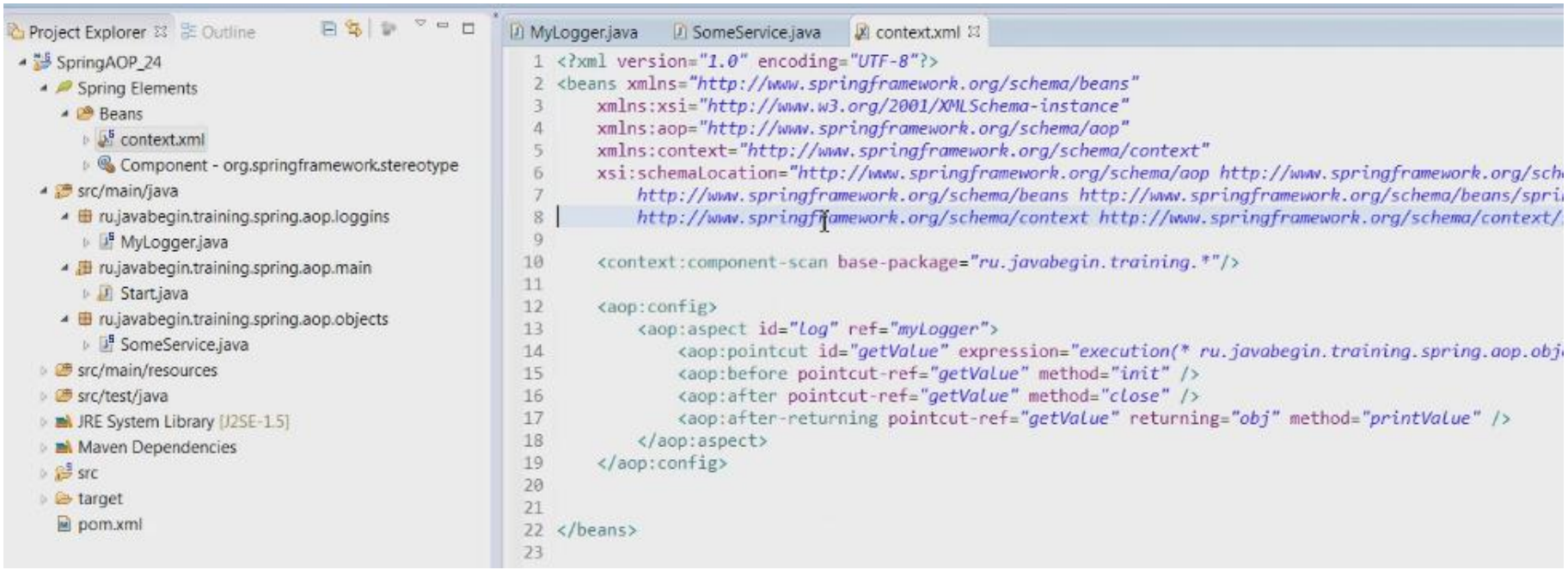
# Добавление зависимостей в проект

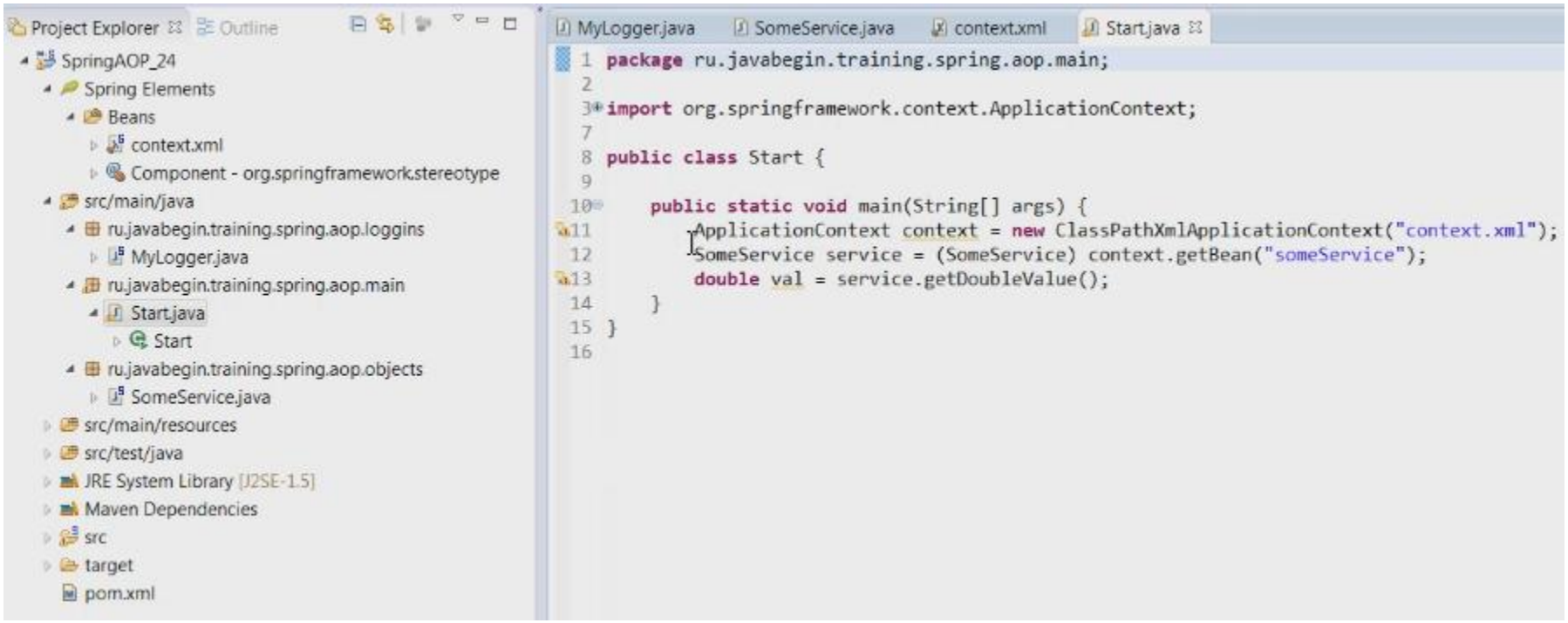
SpringAOP_24/pom.xml

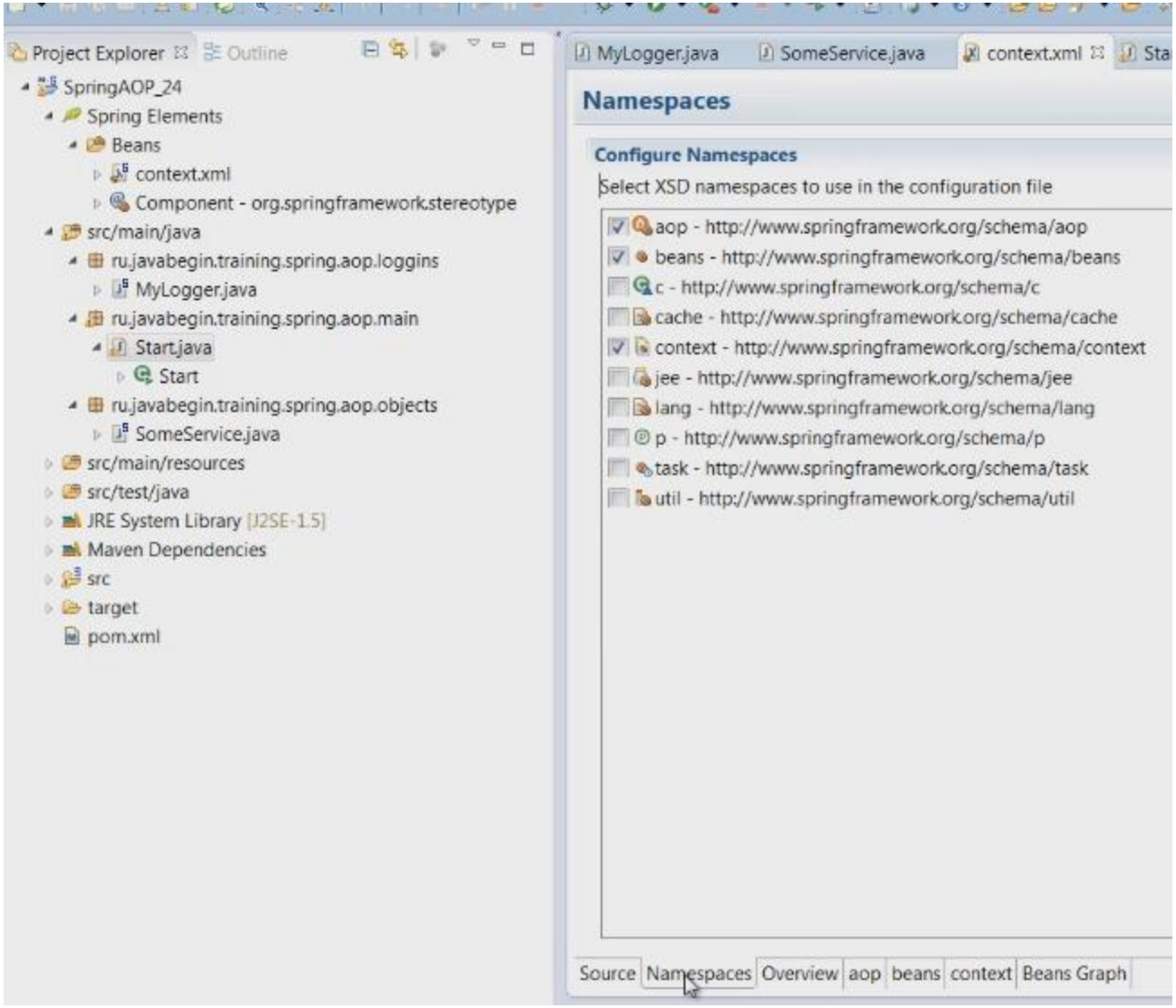
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.springframework.samples</groupId>
    <artifactId>SpringAOP_24</artifactId>
    <version>0.0.1-SNAPSHOT</version>



    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>4.0.3.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>cglib</groupId>
            <artifactId>cglib</artifactId>
            <version>3.1</version>
        </dependency>


        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>4.0.3.RELEASE</version>
        </dependency>


    </dependencies>
</project>
```

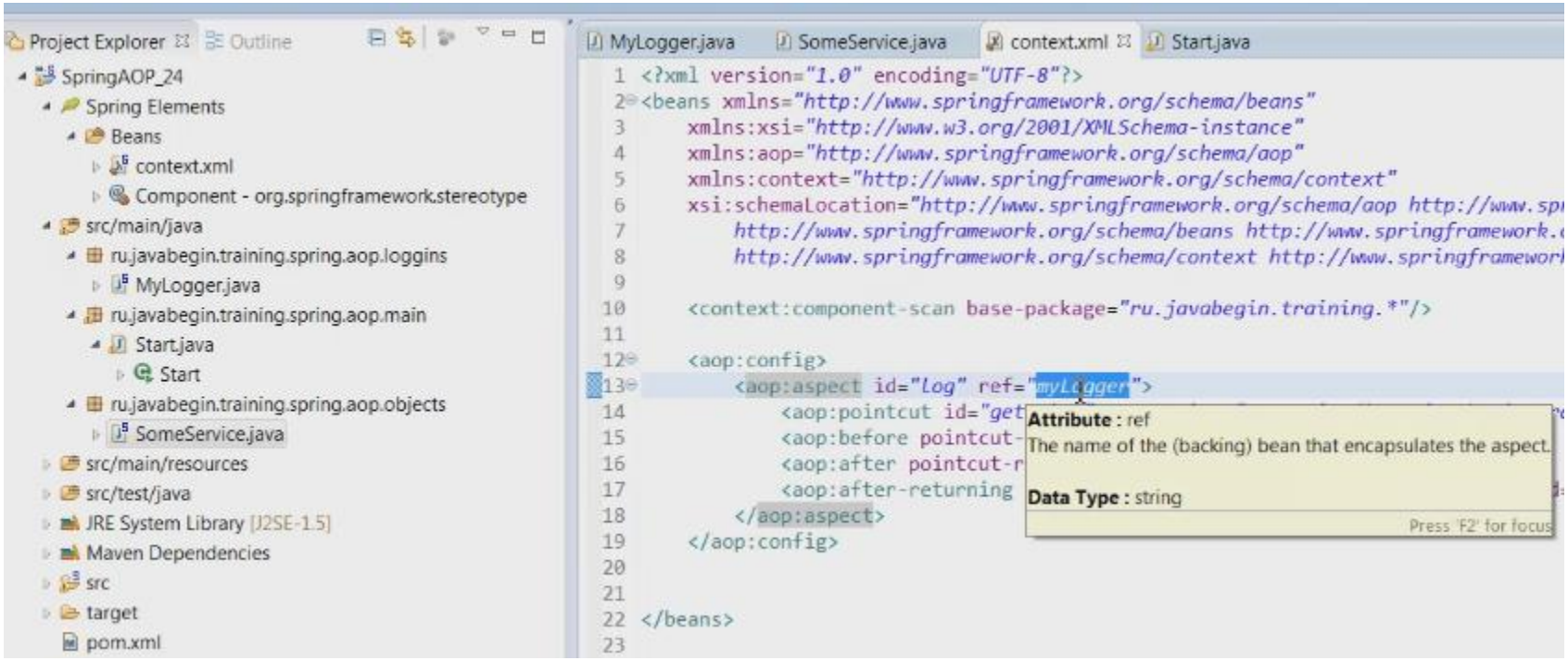

Project Explorer
Outline
SpringAOP_24
Spring Elements
Beans
src/main/java
ru.javabegin.training.spring.aop.loggins
MyLogger.java
ru.javabegin.training.spring.aop.main
ru.javabegin.training.spring.aop.objects
src/main/resources
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
SpringAOP_24/pom.xml
MyLogger.java
1 package ru.javabegin.training.spring.aop.loggins;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class MyLogger {
7
8 public void printValue(Object obj) {
9 System.out.println(obj);
10 }
11
12 public void init() {
13 System.out.println("init");
14 }
15
16 public void close() {
17 System.out.println("close");
18 }
19 }
20


Project Explorer
Outline
SpringAOP_24
Spring Elements
Beans
src/main/java
ru.javabegin.training.spring.aop.loggins
MyLogger.java
ru.javabegin.training.spring.aop.main
Start.java
ru.javabegin.training.spring.aop.objects
SomeService.java
src/main/resources
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
SpringAOP_24/pom.xml
MyLogger.java
SomeService.java
1 package ru.javabegin.training.spring.aop.objects;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class SomeService {
7
8 public int getIntValue() {
9 return 5;
10 }
11
12 public double getDoubleValue() {
13 return 5.6;
14 }
15
16 }
17


Project Explorer
Outline
SpringAOP_24
Spring Elements
Beans
context.xml
Component - org.springframework.stereotype
src/main/java
ru.javabegin.training.spring.aop.loggins
MyLogger.java
ru.javabegin.training.spring.aop.main
Start.java
ru.javabegin.training.spring.aop.objects
SomeService.java
src/main/resources
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
MyLogger.java
SomeService.java
context.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:aop="http://www.springframework.org/schema/aop"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/aop http://www.springframework.org/sch
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/
<context:component-scan base-package="ru.javabegin.training.*"/>
<aop:config>
<aop:aspect id="log" ref="myLogger">
<aop:pointcut id="getValue" expression="execution(* ru.javabegin.training.spring.aop.obj
<aop:before pointcut-ref="getValue" method="init" />
<aop:after pointcut-ref="getValue" method="close" />
<aop:after-returning pointcut-ref="getValue" returning="obj" method="printValue" />
</aop:aspect>
</aop:config>
</beans>
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23


Project Explorer
Outline
SpringAOP_24
Spring Elements
Beans
context.xml
Component - org.springframework.stereotype
src/main/java
ru.javabegin.training.spring.aop.loggins
MyLogger.java
ru.javabegin.training.spring.aop.main
Start.java
Start
ru.javabegin.training.spring.aop.objects
SomeService.java
src/main/resources
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
MyLogger.java
SomeService.java
context.xml
Start.java
1 package ru.javabegin.training.spring.aop.main;
2
3 import org.springframework.context.ApplicationContext;
7
8 public class Start {
9
10 public static void main(String[] args) {
11 ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
12 SomeService service = (SomeService) context.getBean("someService");
13 double val = service.getDoubleValue();
14 }
15 }
16


Project Explorer
Outline
SpringAOP_24
Spring Elements
Beans
context.xml
Component - org.springframework.stereotype
src/main/java
ru.javabegin.training.spring.aop.loggins
MyLogger.java
ru.javabegin.training.spring.aop.main
Start.java
Start
ru.javabegin.training.spring.aop.objects
SomeService.java
src/main/resources
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
MyLogger.java
SomeService.java
context.xml
Namespaces
Configure Namespaces
Select XSD namespaces to use in the configuration file
aop - http://www.springframework.org/schema/aop
beans - http://www.springframework.org/schema/beans
c - http://www.springframework.org/schema/c
cache - http://www.springframework.org/schema/cache
context - http://www.springframework.org/schema/context
jee - http://www.springframework.org/schema/jee
lang - http://www.springframework.org/schema/lang
p - http://www.springframework.org/schema/p
task - http://www.springframework.org/schema/task
util - http://www.springframework.org/schema/util
Source
Namespaces
Overview
aop
beans
context
Beans Graph

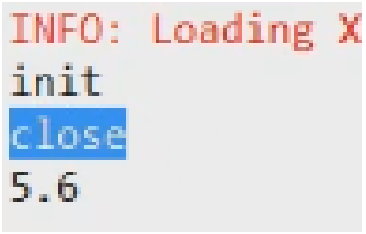

Project Explorer
Outline
SpringAOP_24
Spring Elements
Beans
context.xml
Component - org.springframework.stereotype
src/main/java
ru.javabegin.training.spring.aop.loggins
MyLogger.java
ru.javabegin.training.spring.aop.main
Start.java
Start
ru.javabegin.training.spring.aop.objects
SomeService.java
src/main/resources
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
MyLogger.java
SomeService.java
context.xml
Start.java
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4 xmlns:aop="http://www.springframework.org/schema/aop"
5 xmlns:context="http://www.springframework.org/schema/context"
6 xsi:schemaLocation="http://www.springframework.org/schema/aop http://www.sp
7 http://www.springframework.org/schema/beans http://www.springframework.
8 http://www.springframework.org/schema/context http://www.springframewor
9
10 <context:component-scan base-package="ru.javabegin.training.*"/>
11
12 <aop:config>
13 <aop:aspect id="log" ref="myLogger">
14 <aop:pointcut id="get
15 <aop:before pointcut-
16 <aop:after pointcut-r
17 <aop:after-returning
18 </aop:aspect>
19 </aop:config>
20
21
22 </beans>
23
Attribute : ref
The name of the (backing) bean that encapsulates the aspect.
Data Type : string
Press 'F2' for focus

# Маска аспекта

MyLogger.java | SomeService.java | context.xml | Start.java

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:aop="http://www.springframework.org/schema/aop"
5      xmlns:context="http://www.springframework.org/schema/context"
6      xsi:schemaLocation="http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
7          http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
8          http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd">
9
10     <context:component-scan base-package="ru.javabegin.training.*"/>
11
12     <aop:config>
13         <aop:aspect id="log" ref="myLogger">
14             <aop:pointcut id="getValue" expression="execution(* ru.javabegin.training.spring.aop.objects.SomeService.*(..))" />
15             <aop:before pointcut-ref="getValue" method="init" />
16             <aop:after pointcut-ref="getValue" method="close" />
17             <aop:after-returning pointcut-ref="getValue" returning="obj" method="printValue" />
18         </aop:aspect>
19     </aop:config>
20
21
22 </beans>
23
```

# Вывод результатов



# Проектирование – определение зависимостей

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>3.2.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>3.2.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.7.3</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

# Интерфейс Performer и его реализацию

```java
public interface Performer {
    void doSmth();
    String print();
}
```

```java
public class PerformerImpl implements Performer {
    @Override
    public void doSmth() {
        System.out.println(print());
    }
    @Override
    public String print(){
        return "Performer is working...";
    }
}
```

# Класс с «советами»

```
public class Aspect {
    private Logger logger;

    public void addAppender(){
        logger = Logger.getRootLogger();
        logger.setLevel(Level.INFO);
        PatternLayout layout = new PatternLayout("%d{ISO8601} [%t] %-5p %c %x - %m%n")
;
        logger.addAppender(new ConsoleAppender(layout));
    }
    public void before(){
        logger.info("Before method...");
    }
    public void after(){
        logger.info("After method...");
    }
    public void afterReturning(){
        logger.info("After returning...");
    }
    public void afterThrowing(){
        logger.info("After throwing...");
    }
}
```

# Xml настройки нашего приложения

```
<?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:aop="http://www.spr
ingframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop.xsd">
    <bean id="performerBean" class="net.quizful.aop.PerformerImpl"/>
    <bean id="aspectBean" class="net.quizful.aop.Aspect" init-method="addAppender"/>
    <aop:config>
        <aop:aspect ref="aspectBean">
            <aop:pointcut id="performerPointcut" expression="execution (* net.quizful.
aop.PerformerImpl.doSmth(..))"/>
            <aop:before method="before" pointcut-ref="performerPointcut"/>
            <aop:after method="after" pointcut-ref="performerPointcut"/>
        </aop:aspect>
    </aop:config>
</beans>
```

# параметре тега pointcut - expression.

```
expression="execution (* net.quizful.aop.PerformerImpl.doSmth(..))"
```

- execution означает, что аспект выполняется только при запуске соответствующего метода doSmth.
- Звездочка перед путем означает, что возвращаемое значение может быть любое
- две точки в скобочках, что аргументы могут быть любые.
- можно указывать путь к интерфейсу, а не класса. Тогда аспект будет работать для всех классов, которые имплементируют данный интерфейс

# Главный класс

```
public class App {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationCo
ntext.xml");
        Performer performer = (Performer)context.getBean("performerBean");
        performer.doSmth();
    }
}
```

```
2013-08-31 13:57:53,179 [main] INFO  root  - Before method...
Performer is working...
2013-08-31 13:57:53,181 [main] INFO  root  - After method...
```

# Добавим новые советы

```
public class Worker implements Performer {
    @Override
    public void doSmth() {
        System.out.println(print());
    }
    @Override
    public String print() {
        if(Math.random() < 0.5)
            throw new RuntimeException("Worker exception");
        return "Worker is working...";
    }
}
```

# Добавим новые бины

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.or
g/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop" xsi:s
chemalocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop.xsd">
    <bean id="aspectBean" class="net.quizful.aop.Aspect" init-method="addAppender">
    <bean id="performerBean" class="net.quizful.aop.PerformerImpl">
    <bean id="workerBean" class="net.quizful.aop.Worker">
    <aop:config>
        <aop:aspect ref="aspectBean">
            <aop:pointcut id="performerPointcut" expression="execution (* net.quizful.
aop.PerformerImpl.doSmth(..))">
            <aop:before method="before" pointcut-ref="performerPointcut">
            <aop:after method="after" pointcut-ref="performerPointcut">
        </aop:after></aop:before></aop:pointcut></aop:aspect>
        <aop:aspect ref="aspectBean">
            <aop:pointcut id="workerPointcut" expression="execution(* net.quizful.aop.
Worker.doSmth(..))">
            <aop:after-throwing method="afterThrowing" pointcut-ref="workerPointcut">
            <aop:after-returning method="afterReturning" pointcut-ref="workerPointcut"
>
        </aop:after-returning></aop:after-throwing></aop:pointcut></aop:aspect>
    </aop:config>
</bean></bean></bean></beans>
```

```
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationCo
ntext.xml");
        Performer worker = (Performer)context.getBean("workerBean");
        worker.doSmth();
    }
```

```
Worker is working...
2013-08-31 14:15:06,322 [main] INFO  root  - After returning...
```

```
Exception in thread "main" java.lang.RuntimeException: Worker exception
2013-08-31 14:22:48,084 [main] INFO  root  - After throwing...
```

# Фреймворк PostSharp

- это реализация аспектно-ориентированного подхода для .NET. PostSharp в отличие от многих своих аналогов работает как пост-компилятор, то есть он вносит изменения в MSIL (Microsoft Intermediate Language).
- PostSharp позволяет легко создавать атрибуты, которые меняют поведение методов, полей и типов. Для этого нужно унаследовать класс атрибута от одного из предоставляемых библиотекой базовых классов, реализовать его виртуальные методы и применить этот атрибут.

```
using System;
using PostSharp.Aspects;

namespace HelloAspects
{
    class Program
    {
        private static void Main()
        {
            hello();
        }

        [SayGoodbye]
        private static void hello()
        {
            Console.WriteLine("Hello!");
        }
    }

    [Serializable]
    class SayGoodbyeAttribute : OnMethodBoundaryAspect
    {
        public override void OnExit(MethodExecutionArgs args)
        {
            Console.WriteLine("Goodbye.");
        }
    }
}
```

- Метод OnExit называют советом (advice), он всегда выполняется (даже если выпадет исключение, так как OnExit вызывается из блока finally) после тела метода, к которому применяется атрибут. Помимо него класс OnMethodBoundaryAspect предоставляет ещё три совета:
- **OnEntry** — выполняется перед телом метода;
- **OnSuccess** — выполняется после успешного выполнения тела метода;
- **OnException** — выполнятся после тела метода в случае, если в методе выпало необработанное исключение.

```
using System;
using System.Diagnostics;
using PostSharp.Aspects;

namespace HelloAspects
{
    class Program
    {
        private static void Main()
        {
            Trace.Listeners.Add(new TextWriterTraceListener(Console.Out));
            hello();
        }

        [Trace]
        private static void hello()
        {
            Console.WriteLine("Hello!");
        }
    }

    [Serializable]
    public class TraceAttribute : OnMethodBoundaryAspect
    {
        public override void OnEntry(MethodExecutionArgs args)
        {
            Trace.WriteLine(string.Format("Entering {0}.{1}.", args.Method.DeclaringType.Name, args.Method.Name));
        }

        public override void OnExit(MethodExecutionArgs args)
        {
            Trace.WriteLine(string.Format("Leaving {0}.{1}.", args.Method.DeclaringType.Name, args.Method.Name));
        }
    }
}
```

- В чём же преимущество использования АОП в данном примере? Представим, что у нас есть несколько классов, в каждом из которых много методов и нам необходимо реализовать трассировку. Если не использовать АОП, то придётся в теле каждого метода прописывать **Trace.WriteLine**… Используя же АОП мы выделяем эту сквозную функциональность в отдельную сущность (аспект) и применяем её к методам при помощи атрибута.

**PostSharp есть и другие аспекты**

- **OnMethodBoundaryAspect:**
- **EventInterceptionAspect**
- **LocationInterceptionAspect**
- **OnExceptionAspect**
- **…**

PostSharp — это удобный инструмент для внедрения АОП в программы, написанные с использованием среды .NET. АОП дополняет ООП, выделяя сквозную функциональсть в отдельные аспекты, избавляется от дублирования кода (принцип DRY – Don't Repeat Yourself) и упрощает архитектуру приложения.

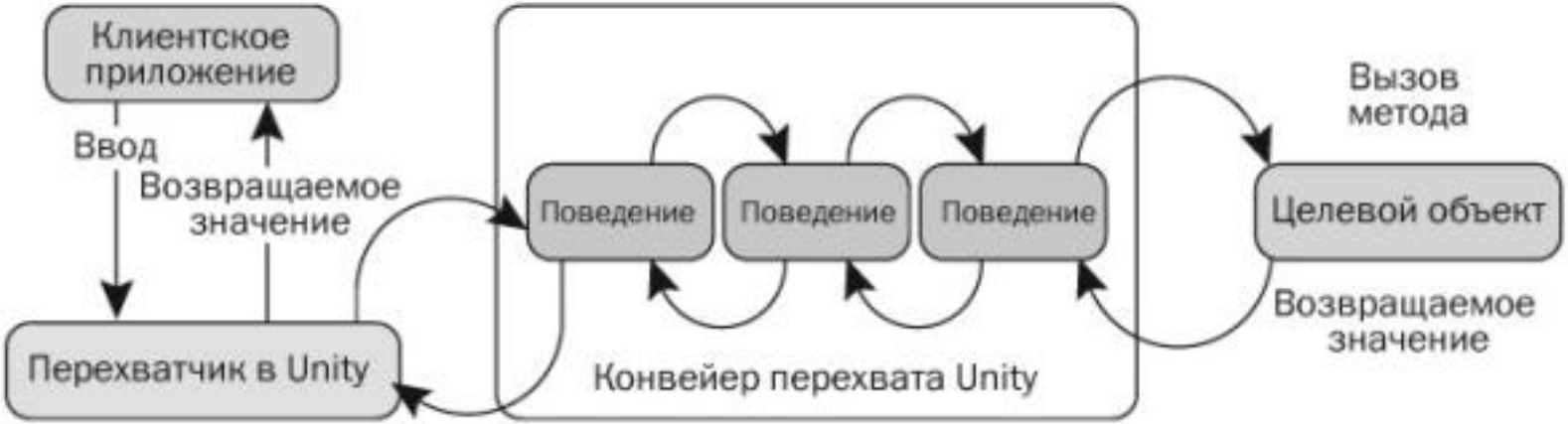
# Краткий обзор Unity 2.0

Unity — это блок приложения, доступный как часть проекта Microsoft Enterprise Library, а также в отдельном виде. Microsoft Enterprise Library — это набор блоков приложения, которые снимают часть проблем, связанных с горизонтальным пересечением иерархии и характерных в разработке .NET-приложений (протоколирование, кеширование, шифрование, обработка исключений и др.).

# Перехват в Unity 2.0

- Основная концепция перехвата в Unity позволяет разработчикам настраивать цепочку вызовов, необходимых для запуска какого-либо метода некоего объекта.
- Иначе говоря, механизм перехвата Unity захватывает вызовы, выдаваемые для настройки объектов, и изменяет поведение целевых объектов, добавляя дополнительный код до, после и вокруг обычного кода методов.
- Перехват — фактически очень гибкий подход к добавлению нового поведения для объекта в период выполнения, не затрагивающий его исходный код и не влияющий на поведение классов в той же цепочке наследования.
- Перехват в Unity — способ реализации популярного проектировочного шаблона Decorator, разработанного для расширения функциональности объекта в период выполнения и в момент его использования.
- Декоратор (decorator) — это объект-контейнер, который принимает (и поддерживает ссылку на) экземпляр целевого объекта и дополняет его возможности.

# Перехват объекта в действии (в Unity 2.0)



# Настройка перехвата

```
1. var container = new UnityContainer();
2. container.AddNewExtension<Interception>();
```

перехват реализуется простым добавлением нового расширения к контейнеру, чтобы описать, как будет разрешаться объект.

# добавить в конфигурационный файл

- Цель этого сценарного кода — расширение схемы конфигурации новыми элементами и псевдонимами, специфичными для подсистемы перехвата

```
<sectionExtension type="Microsoft.Practices.Unity.InterceptionExtension.
  Configuration.InterceptionConfigurationExtension,
  Microsoft.Practices.Unity.Interception.Configuration"/>
```

# Определение контейнера

- Перехватчик интерфейса (interface interceptor) — это перехватчик экземпляра, ограниченный в своих действиях до прокси только одного интерфейса объекта. Такой перехватчик создает класс прокси с помощью генерации динамического кода. Элемент поведения interception в конфигурации указывает внешний код, который должен выполняться вокруг перехватываемого экземпляра объекта.

```
<container>
 <extension type="Interception" />
 <register type="IBankAccount" mapTo="BankAccount">
   <interceptor type="InterfaceInterceptor" />
   <interceptionBehavior type="TraceBehavior" />
 </register>
</container>
```

- Класс TraceBehavior нужно конфигурировать декларативно, чтобы контейнер мог разрешать его и любые его зависимости. Чтобы сообщить контейнеру о классе TraceBehavior и его конструкторе вы используете элемент <register>:

```
<register type="TraceBehavior">
    <constructor>
        <param name="source" dependencyName="interception" />
    </constructor>
</register>
```

```
class TraceBehavior : IInterceptionBehavior, IDisposable    {
  private TraceSource source;
  public TraceBehavior(TraceSource source)      {
    if (source == null)
      throw new ArgumentNullException("source");
    this.source = source;
  }
  public IEnumerable<Type> GetRequiredInterfaces()      {
    return Type.EmptyTypes;
  }
  public IMethodReturn Invoke(IMethodInvocation input,
    GetNextInterceptionBehaviorDelegate getNext)      {
    // BEFORE the target method execution
    this.source.TraceInformation("Invoking {0}",
      input.MethodBase.ToString());
    // Yield to the next module in the pipeline
    var methodReturn = getNext().Invoke(input, getNext);
    // AFTER the target method execution
    if (methodReturn.Exception == null)         {
      this.source.TraceInformation("Successfully finished {0}",
        input.MethodBase.ToString());
    } else {
      this.source.TraceInformation(
        "Finished {0} with exception {1}: {2}",
        input.MethodBase.ToString(),
        methodReturn.Exception.GetType().Name,
        methodReturn.Exception.Message);
    }
    this.source.Flush();
    return methodReturn;
  }
  public bool WillExecute {
    get { return true; }
  }
  public void Dispose(){
    this.source.Close();
  }
}
```

- Класс поведения реализует IInterceptionBehavior, который в основном состоит из метода Invoke. Этот метод содержит всю логику, нужную для любого метода, который находится под контролем перехватчика. Если вы хотите сделать что-то до вызова целевого метода, то делаете это в начале метода. Когда вам требуется перейти к целевому объекту (или, точнее, к следующему поведению, зарегистрированному в конвейере), вы вызываете делегат getNext, предоставляемый инфраструктурой.

# Гибкость конфигурирования

- Перехват и AOP в целом открывают целый ряд интересных возможностей. Например, перехват позволяет добавлять обязанности в индивидуальные объекты без модификации всего класса, благодаря чему решение получается гораздо более гибким, чем при использовании шаблона Decorator.

# Для чего нужен АОП?

- ООП отлично себя зарекомендовала для решения доменных задач. Но есть сопутствующие задачи, которые прошивают все слои и с ними ООП справляется плохо.
- АОП позволяет:
  - Уменьшить стоимость разработки и время доставки приложения
  - Сократить количество ошибок в приложении
  - Увеличить поддерживаемость ПО

Мир Java имеет AspectJ,
в .NET сравнимым функционалом обладает PostSharp

# Состав типичного бизнес кода

```
public void ProcessOrder(Period period, List<Item> items) {
#if DEBUG
    Log.Write(period);
    Log.Write(items);
#endif
    if(accessService.Check(Credentials))
        return;
    if(!items.Any())
        return;
    if(!period.IsValid())
        return;

     foreach (var item in items) {
         SomeService.Process(item, period.From);
     }
     AntherService.CreateInvoice(period, items);
#if DEBUG
     Log.Write("ProcessOrder " + executionTime);
#endif
}
```

# Применимость АОП

- Логирование
- Трассировка
- Кэширование

- Но:
  - Применимость гораздо шире!
  - Например, шаблоны проектирования
  - Обеспечение уровня доступа к данным

# Применимость АОП – не банальные случаи

- Автоматизация инструментирования
  - Например, расставить другие атрибуты для классов
- Реализация GoF шаблонов с помощью АОП
  - Многие шаблоны можно отделить и применять декларативно
- Безопасная работа с многопоточностью
- Обработка исключений
- Контракты на использование типов данных
- Undo\Redo

- Много примеров использования есть на postsharp.net

# Пример на инструментирование кода - **до**

- Код размещается в сборке SomeLibraryExample, в неймспейсе DTO

```
namespace SomeLibraryExample.DTO {
    public class SuperHero {
        public string Name { get; set; }
        public bool CanFly { get; set; }
        public bool IsVillian { get; set; }
    }
}
```

- Точка внедрения объявляется в AssemblyInfo.cs

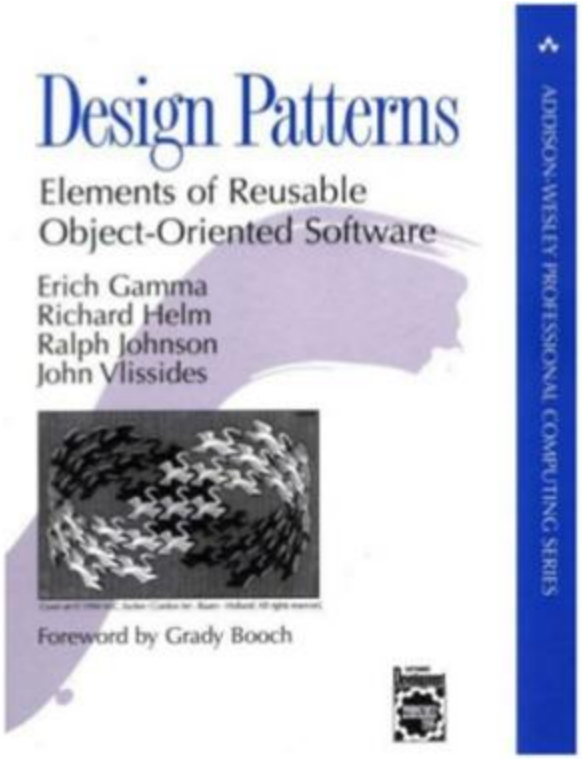
```
[assembly: AutoDataContract(AttributeTargetTypes = "SomeLibraryExample.DTO.*")]
```

# Возможности PostSharp

- Явное указание точек внедрения
  - сборка, класс/объект, метод, свойство, поле
- Внедрение по шаблону имен
  - использование wildcards
- Инструментирование аспектами на этапе компиляции
- Проверка требований на этапе компиляции и в RunTime
- Управление порядком применения и наследуемостью аспектов
- Дебаг с точками остановки для CompileTime и RunTime
- Подсветка точек внедрения в Visual Studio

## Шаблоны «Банды четырех»

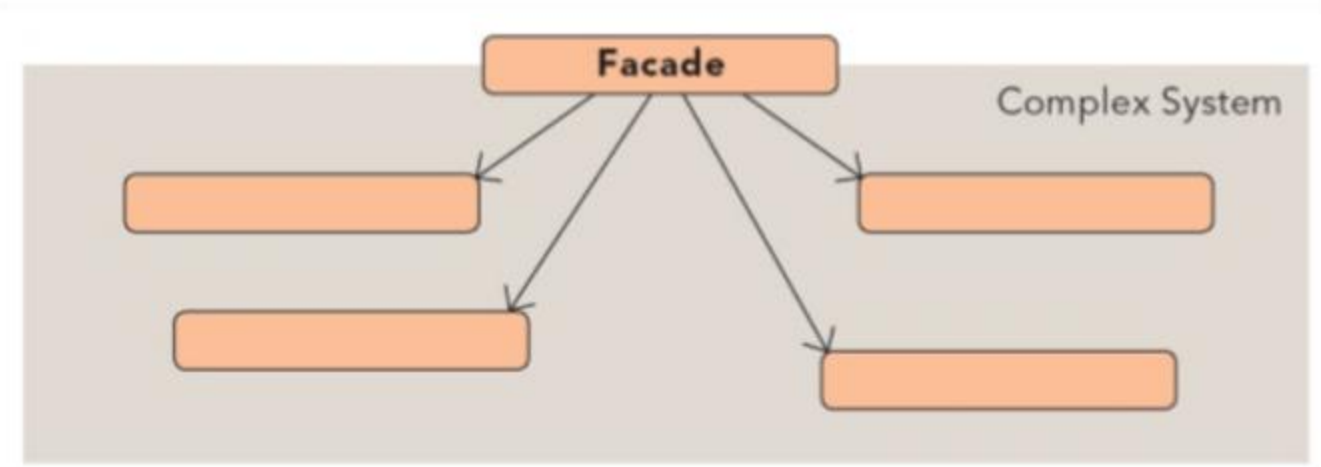
- Мировой бестселлер вышел в 1994 году
- 23 шаблона были разделены на 3 группы:
  - Создание объектов
  - Композиция объектов
  - Поведение объектов




Облегчают понимание и модификацию, но усложняют код. Инфраструктура внедряется в бизнес-логику.

# Аспекты не применимы

- Façade – не выигрывает от применения аспектов.
  - Цель шаблона предоставить более удобное API для подсистемы



# Аспекты не играют особенной роли

- Неочевидный выигрыш для шаблонов:
  - State
  - Interpreter
- Аспекты позволяют чуть сильнее локализировать код шаблона.

## Аспекты могут помочь при определенных условиях

- Следующие шаблоны используют мощь наследования для достижения своих целей и реализации себя:
  - Abstract Factory
  - Factory Method
  - Template Method
  - Builder
  - Bridge
- Аспекты могут нести в себе код из абстрактных классов, освобождая место для явного наследования

# Аспекты могут облегчить жизнь

- Большая часть поведения, может быть выделена в аспект для следующих шаблонов:
  - Composite
  - Command
  - Mediator
  - Chain of Responsibilities

```
[Node]
public class TopLevel : IComposite {
    private readonly List<IComponent> items = new List<IComponent>();

    public void Add(IComponent component) {
        items.Add(component);
    }

    public int Count() {
        return items.Count;
    }
}

[Node(Parent = typeof (TopLevel))]
public class LowLevel2 : IComponent {}
```

# Аспекты кардинально меняют картину

- Для следующих шаблонов аспекты могут полностью изменить процесс использования шаблона:
  - Adapter
  - Decorator
  - Strategy
  - Visitor
  - Proxy
- Код шаблона выносится в аспект и применяется для всего семейства участвующих классов

# Аспекты полностью реализуют шаблон

- Для следующих шаблонов реализация в участниках полностью исчезает
  - Observer
  - Memento
  - Singleton
  - Prototype
- Реализация шаблона становится универсальной

```
[SingletonClass]
public class MyClass {
    public int Counter;

    public static MyClass Instance { get; private
set; }

    private MyClass() {}
   }
}
```

# Проверка структуры кода на этапе компиляции

- Проверка структуры на разных уровнях
  - Сборка, класс, методы
- Управление уровнем критичности нарушения структуры
  - Info, Error, Warning и другие

# Основные выводы по рефакторингу GoF

- Для 17 из 23 шаблонов возможно применить АОП с локализацией кода.
- Для 12 из 17 можно ядро шаблона выделить в отдельные повторно используемые классы.
- Для 14 из 17 появляется более прозрачная компоновка участников шаблона.
- Знания о применении шаблона не теряются благодаря подсветке синтаксиса

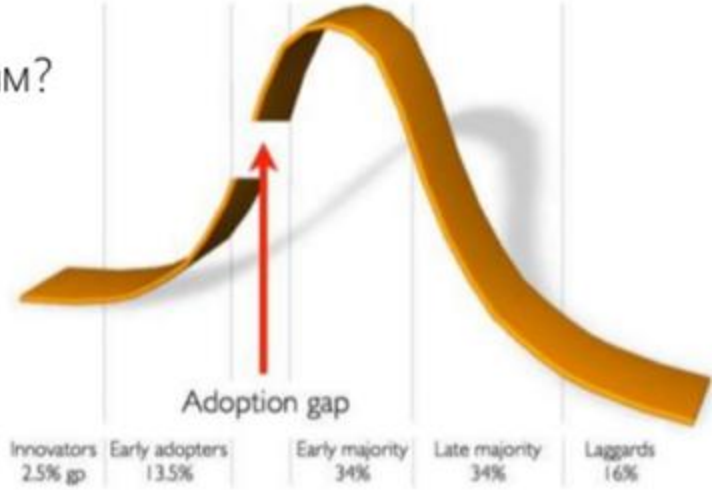
```
[SingletonClass]
4 references
public class MyClass {
    public int Counter;

    2 references
    public static MyClass Instance { get; private set; }

    0 references
    private MyClass() {}

    1 reference
    public int Foo() {
        return ++Counter;
    }
}
```

Accessor is enhanced by Single (provided by aspect SingletonClass applied on type MyClass).
Intercepted by advice OnGetValue.

| Шаблон | Локальность | Повторное использование | Прозрачность компоновки | Отключаемость |
|---|---|---|---|---|
| Façade | - | - | - | - |
| Abstract Factory | Нет | Нет | Нет | Нет |
| Bridge | Нет | Нет | Нет | Нет |
| Builder | Нет | Нет | Нет | Нет |
| Factory Method | Нет | Нет | Нет | Нет |
| Interpreter | Нет | Нет | н/о | Нет |
| Template Method | (Да) | Нет | Нет | (Да) |
| Adapter | Да | Нет | Да | Да |
| State | (Да) | Нет | н/о | (Да) |
| Decorator | Да | Нет | Да | Да |
| Proxy | (Да) | Нет | (Да) | (Да) |
| Visitor | (Да) | Да | Да | (Да) |
| Command | (Да) | Да | Да | Да |
| Composite | Да | Да | Да | (Да) |
| Iterator | Да | Да | Да | Да |
| Flyweight | Да | Да | Да | Да |
| Memento | Да | Да | Да | Да |
| Strategy | Да | Да | Да | Да |
| Mediator | Да | Да | Да | Да |
| Chain of Responsibility | Да | Да | Да | Да |
| Prototype | Да | Да | (Да) | Да |
| Singleton | Да | Да | н/о | Да |
| Observer | Да | Да | Да | Да |

# Вопросы

- Приходилось ли использовать в продакшене?
  - Да, приходилось
- Для чего?
  - Как раз таки для детального трейса фин.операций в системе для этого не приспособленной изначально.
- Тяжело ли его продавить руководству и коллегам?
  - Да, очень тяжело
- Почему?
  - Сопротивление всему новому в природе человека



# Ссылки для самостоятельного изучения

- Примеры АОП решений - http://www.bodden.de/tools/aop-dot-net/
- Пишем АОП сами http://habrahabr.ru/post/199378/ - для уверенных и смелых
- Пользуемся готовым – http://www.postsharp.net/ – есть бесплатная версия, которая хороша
- Теория
  - https://www.cs.ubc.ca/labs/spl/papers/2002/oopsla02-patterns.pdf
  - ftp://cs.joensuu.fi/pub/Theses/2008_MSc_Oprisan_Andrei.pdf
- Мифы и реальность АОП (ажно 15 штук)
  - https://www.ibm.com/developerworks/ru/library/j-aopwork15/
- Примеры кода https://github.com/VioletTape/GoF_PostSharp