Java Best Practice

Лучшие практики разработки на Java

PETER-SERVICE

Обзор курса

- Инструменты автоматизированной сборки проектов (Ant, Maven)
- Обзор технологии Unit-тестирования (Junit)
- Технологии логирования (Log4J, Slf4J, LogBack)
- Создание «заглушек» (Mockito)
- Введение в Spring
- MVC Паттерн (Spring MVC)

Обзор курса. Продолжение

- Обзор технологии ORM
- Создание DAO-объекта (JDBC, MyBatis)
- Механизмы безопасности (Spring Security)
- Создание веб-сервисов SOAP и REST (Apache CXF)
- Мониторонг (JMX)


<APACHE ANT>

Apache Ant

- **Apache Ant** (англ. *ant* — муравей и акроним — «Another Neat Tool») – утилита для автоматической сборки проекта
- Императивная сборка проекта
- Платформонезависимость (Использует JRE)
- Управление процессом сборки на основе сценария, записанного в файле build.xml

```xml
<?xml version="1.0"?>
<project default="build" basedir=".">
    <property name="name" value="AntBuildJar"/>
    <property name="src.dir" location="${basedir}/src"/>
    <property name="build" location="${basedir}/build"/>
    <property name="build.classes" location="${build}/classes"/>
    <path id="libs.dir">
        <fileset dir="lib" includes="**/*.jar"/>
    </path>
    <!-- Сборка приложения -->
    <target name="build" depends="clean" description="Builds the application">
        <!-- Создание директорий -->
        <mkdir dir="${build.classes}"/>

        <!-- Компиляция исходных файлов -->
        <javac srcdir="${src.dir}"
               destdir="${build.classes}"
               debug="false"
               deprecation="true"
               optimize="true" >
            <classpath refid="libs.dir"/>
        </javac>

        <!-- Копирование необходимых файлов -->
        <copy todir="${build.classes}">
            <fileset dir="${src.dir}" includes="**/*.*" excludes="**/*.java"/>
        </copy>

        <!-- Создание JAR-файла -->
        <jar jarfile="${build}/${name}.jar">
            <fileset dir="${build.classes}"/>
        </jar>
    </target>

    <!-- Очистка -->
    <target name="clean" description="Removes all temporary files">
        <!-- Удаление файлов -->
        <delete dir="${build.classes}"/>
    </target>
</project>
```

maven

Apache Maven

- Инструмент для автоматизации сборки проектов: компиляции, создания jar, создания дистрибутива программы, генерации документации
- Обеспечивает декларативную сборку. Информация о проекте описывается на языке POM(Project Object Model) и содержится в файле pom.xml
- Maven придерживается принципа «соглашения прежде конфигурации»

Ключевые преимущества

- Автоматическое управление зависимостями
- Огромный, поддерживаемый в актуальном состоянии репозиторий артефактов
- Maven – наиболее широко распространенный инструмент для сборки
- Поддержка большинством современных IDE (Eclipse, IntelliJ IDEA …)

Установка

- Требуется наличие на машине JDK версии >= 1.5
- Дистрибутив можно скачать с сайта http://maven.apache.org
- Прописать переменную окружения M2_HOME
- Добавить путь %M2_HOME%/bin в PATH
- Для запуска используется команда **mvn**

Проверка установки

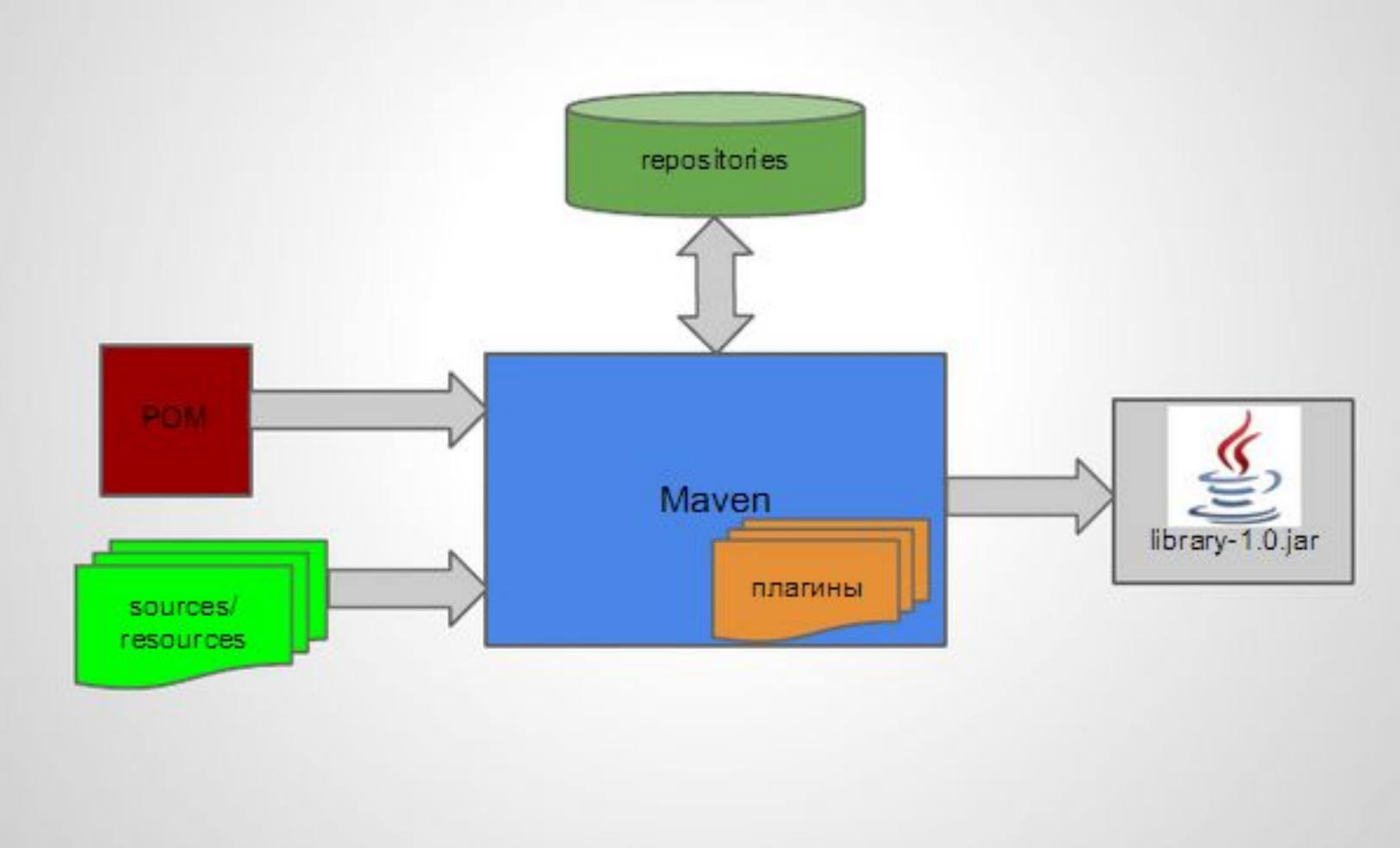
```
D:\Java\mavenTest2>mvn -v
Apache Maven 3.0.4 (r1232337; 2012-01-17 12:44:56+0400)
Maven home: D:\Java\apache-maven-3.0.4\bin\..
Java version: 1.6.0_37, vendor: Sun Microsystems Inc.
Java home: c:\Program Files (x86)\Java\jdk1.6.0_37\jre
Default locale: ru_RU, platform encoding: Cp1251
OS name: "windows 7", version: "6.1", arch: "x86", fami
D:\Java\mavenTest2>
```

Создание тестового проекта

- mvn archetype:generate

Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): **254**:
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
Choose a number: **6**:
Define value for property 'groupId': : **com.peterservice**
Define value for property 'artifactId': : **mavenTest2**
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': com.peterservice: :
Confirm properties configuration:
groupId: com.peterservice
artifactId: mavenTest2
version: 1.0-SNAPSHOT
package: com.peterservice
Y: : **Y**

Схема


repositories
POM
Maven
плагины
sources/
resources
library-1.0.jar

Артефакт

- Что такое Артефакт? Да все что угодно, например (jar,war, и.т.п.)
- 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 (нашихи внешних, плагинов maven)

Координаты артефакта

- groupId
- artifactId
- [packaging] default jar
- Version в формате mmm.nnn.bbb-ssssss-dd , необязательными являются поля ssssss (спецификатор SNAPSHOT,RELEASE и т.п.) и dd (номер сборки)
- [classifier]

groupId:artifactId[:packaging]:version[:classifier]

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.

```
<project xmlns=...>
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.codehaus.mojo</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0</version>
</project>
```

Декларативный стиль в POM

- Основная концепция maven в том, что мы используем **лучшие практики** разработки (best practices) с зафиксированными правилами и настройками **по умолчанию.** Maven использует наследование, агрегирование и управление зависимостями при описании проекта в POM файле.
- Пример стандартного размещения файлов java проекта:

Directory name	Purpose
project home	Contains the pom.xml and all subdirectories.
src/main/java	Contains the deliverable Java sourcecode for the project.
src/main/resources	Contains the deliverable resources for the project, such as property files.
src/test/java	Contains the testing classes (JUnit or TestNG test cases, for example) for the project
src/test/resources	Contains resources necessary for testing.

Build директория по умолчанию - **target**

Репозитории

- Репозиторий maven это файловое хранилище с метаинформацией и быстрым поиском и доступом
  - 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

Жизненный цикл

- ✓ **validate**, initialize
- ✓ generate-sources, process-sources
- ✓ generate-resources, process-resources
- ✓ **compile**, process-classes
- ✓ generate-test-sources, process-test-sources
- ✓ generate-test-resources, process-test-resources
- ✓ test-compile, process-test-classes
- ✓ **test**
- ✓ prepare-package, **package**
- ✓ pre-integration-test, integration-test, post-integration-test
- ✓ **verify, install, deploy**

Выполнение фаз жизненного цикла

- mvn [имя фазы]

- При выполнении определенной фазы автоматически выполняются все предыдущие фазы
- **mvn test (выполняется в папке, где находится pom.xml)**
  - validate -> compile -> test
  - сообщения об ошибках в папке **target\surefire-reports**

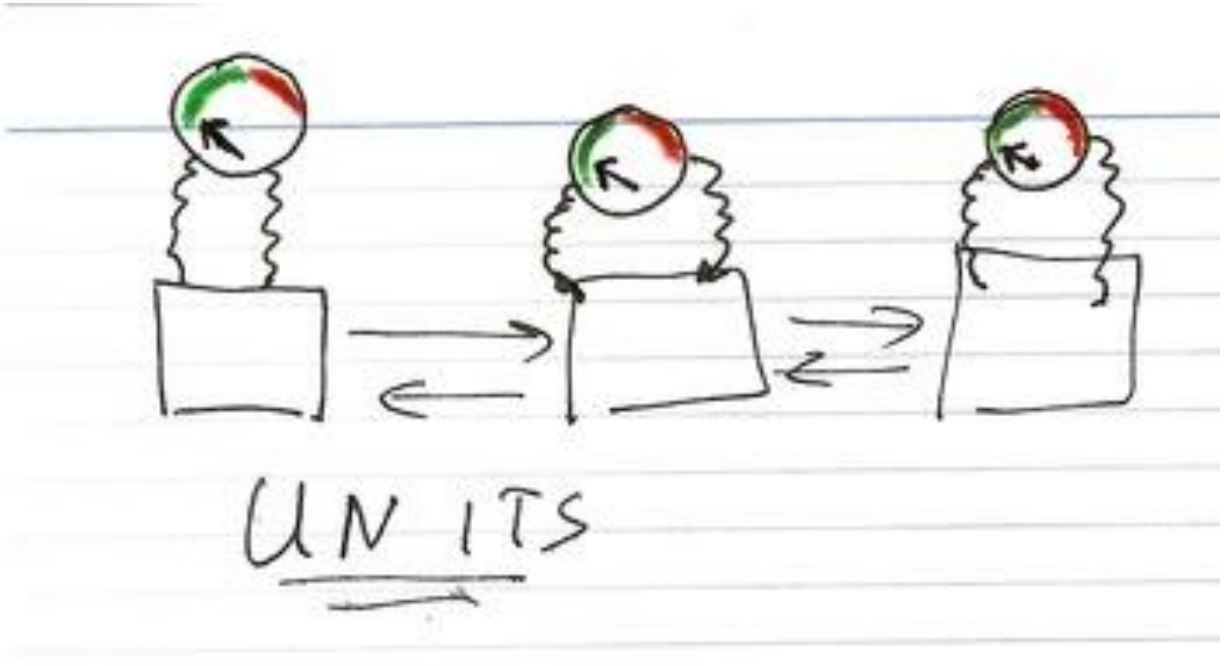
Junit dependency

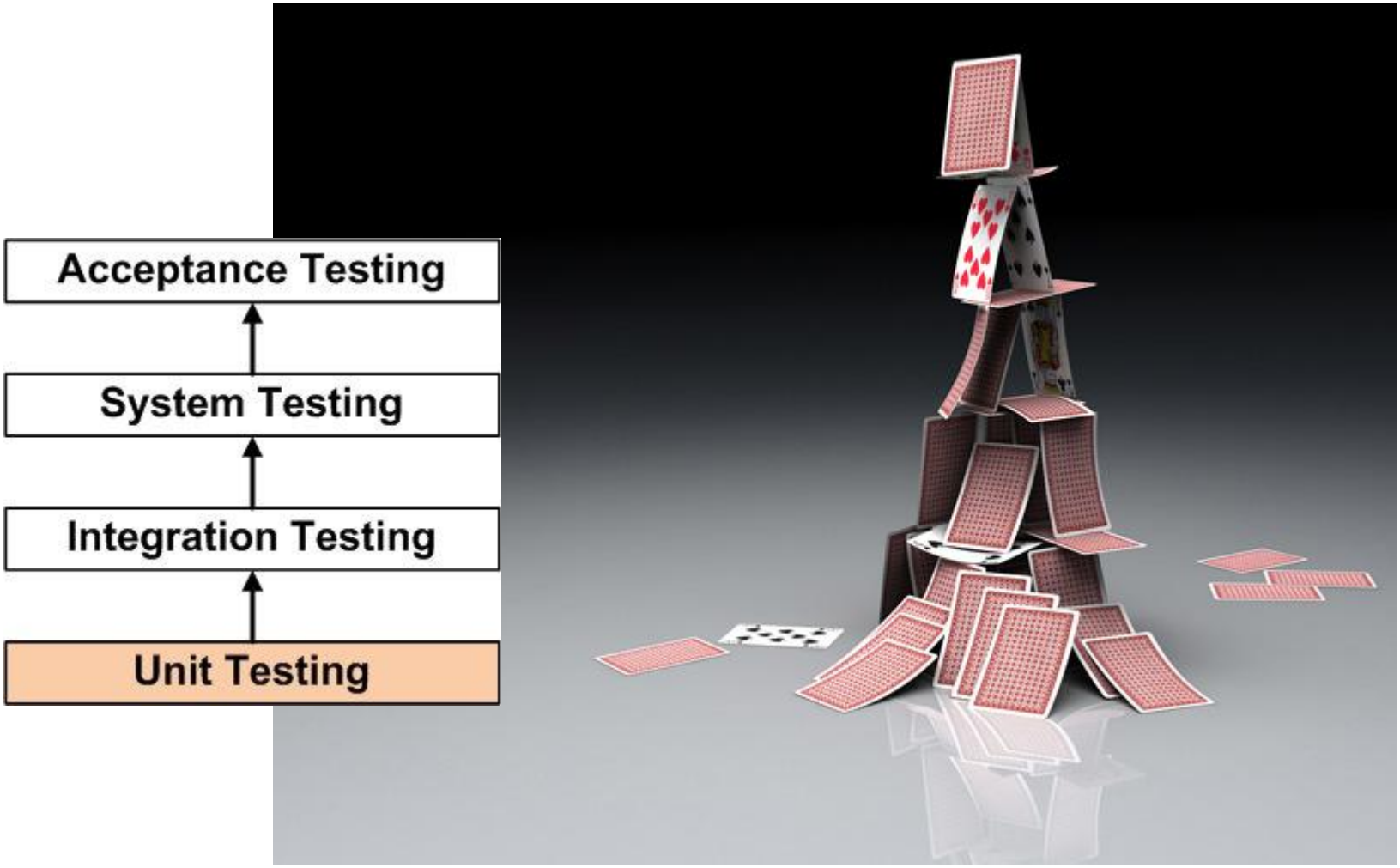
```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
```

Обзор технологии Unit-тестирования

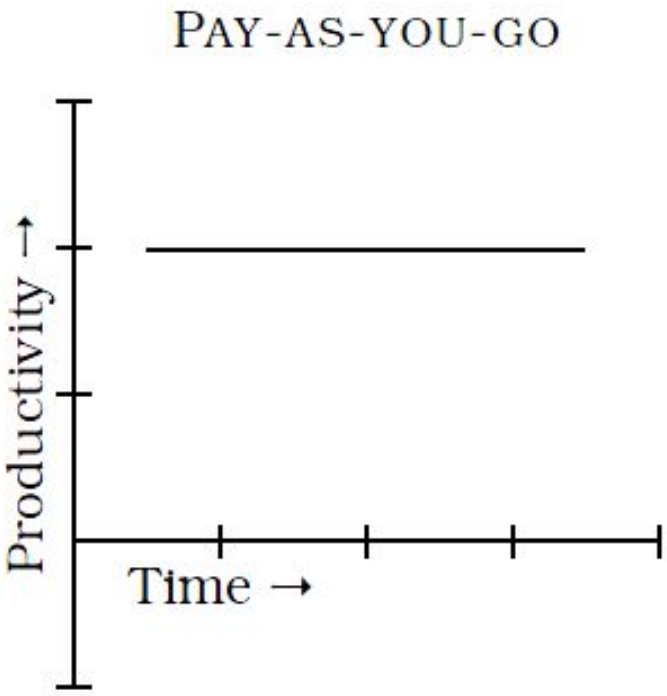
Что такое Unit-тесты

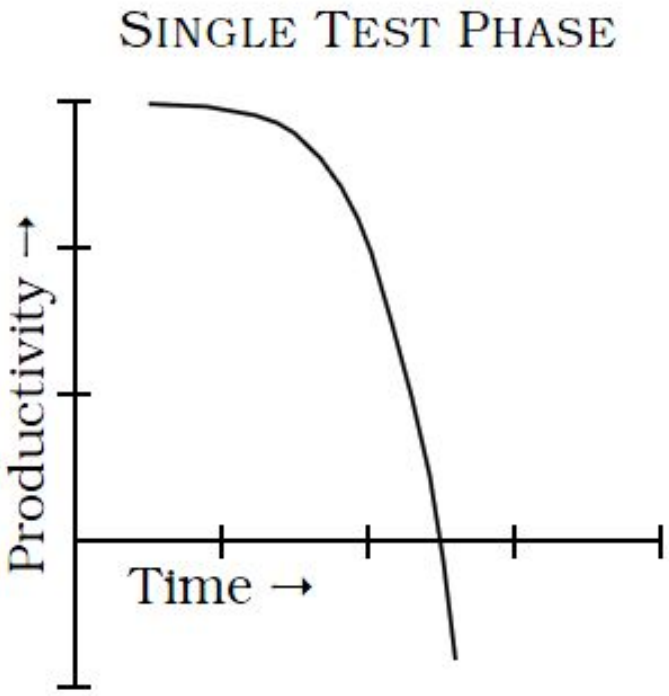
- Unit-тест – код, написанный разработчиком, который проверяет небольшой кусок функциональности тестируемого кода.




Acceptance Testing
System Testing
Integration Testing
Unit Testing

Paying-as-you-go vs. Having a Single Testing Phase





Библиотека Junit **(Версия 4)**

- Тестовый класс может иметь любое название. Частая практика заканчивать имя класса на Test (CalculatorTest)
- В Junit 4 не нужно наследовать от TestCase
- Тестовый метод должен быть помечен аннотацией @Test и может иметь произвольное имя
- Хорошей практикой считается называть методы, начиная с test (testAdd)

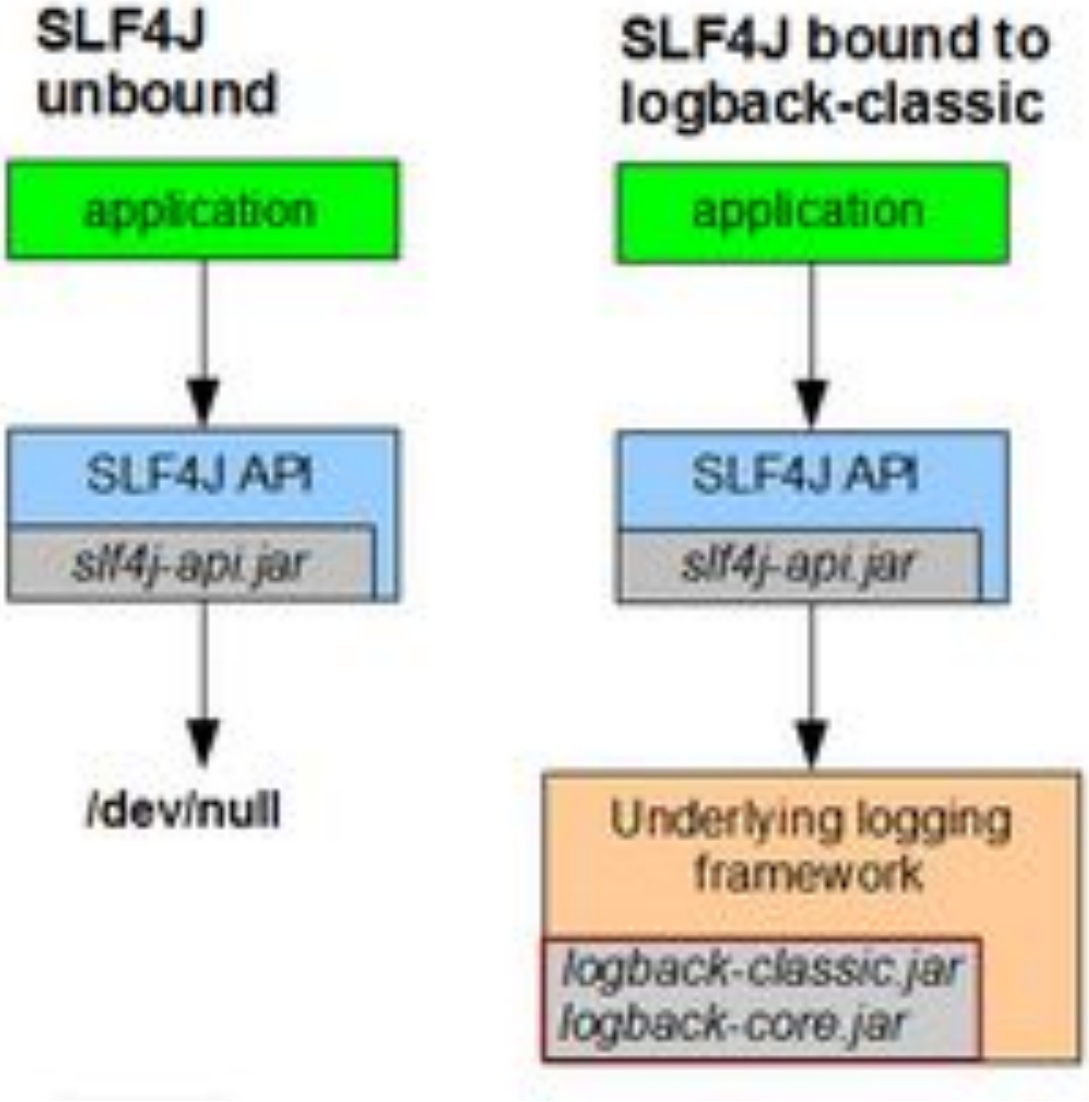
```java
package com.peterservice;

import org.junit.BeforeClass;
import org.junit.Test;
import static junit.framework.Assert.*;

public class AppTest
{
    @BeforeClass
    public static void setUp() {
        System.out.println("SetUp Before Test!!");
    }
    @Test
    public void testDumb()
    {
        System.out.println("testDumb");
        assertTrue(true);
    }
    @Test
    public void testAdd2Integers() {
        System.out.println("testAdd2Integers");
        assertEquals(5, new App().add2Integers(2,3));
    }
}
```

Аннотации JUnit

- **@BeforeClass/@AfterClass (метод должен быть public и static):** Метод помеченный такой аннотацией выполняется один раз перед выполнением всех тестовых методов
- **@Before/@After (метод должен быть public):** Метод помеченный такой аннотацией выполняется до/после каждого тестового метода
- **@Test (expected = RuntimeException.class):** Аннотация говорит о том, что в тесте ожидается exception типа Runtime, если в тесте exception не будет выброшен, то тест будет провален
- **@Test (timeout = 100):** Тест будет провален, если он будет выполняться боллее 100 миллисекунд
- **@Ignore (value = "Ignore"):** пропустить тест (не выполнять его)


SLF4J
Simple Logging
Facade for Java

Подключение библиотеки логирования



Уровни логирования

The six logging levels used by Log are (in order):

1. trace (the least serious)
2. debug
3. info
4. warn
5. error
6. fatal (the most serious)


LOGBack™
The Generic, Reliable
Fast & Flexible
Logging Framework

Настройки

- Конфигурация logback находится в файлах:
  - *logback-test.xml*
  - *logback.xml*
- Если найти данные файлы не удается, то используется настройка по-умолчанию (вывод информации в консоль)

Использование логирования

```
10  public class AppTest
11  {
12      private static final Logger logger = getLogger(AppTest.class);
13      @Test
14      public void testLogbackLogging() {
15          logger.info("info message");
16          logger.debug("test started");
17          assertEquals(1,1);
18          logger.debug("test finished");
19      }
20  }
```

Пример конфигурации

View as .groovy

```
<configuration debug="true">

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!-- encoders are  by default assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder -->
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <root level="debug">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

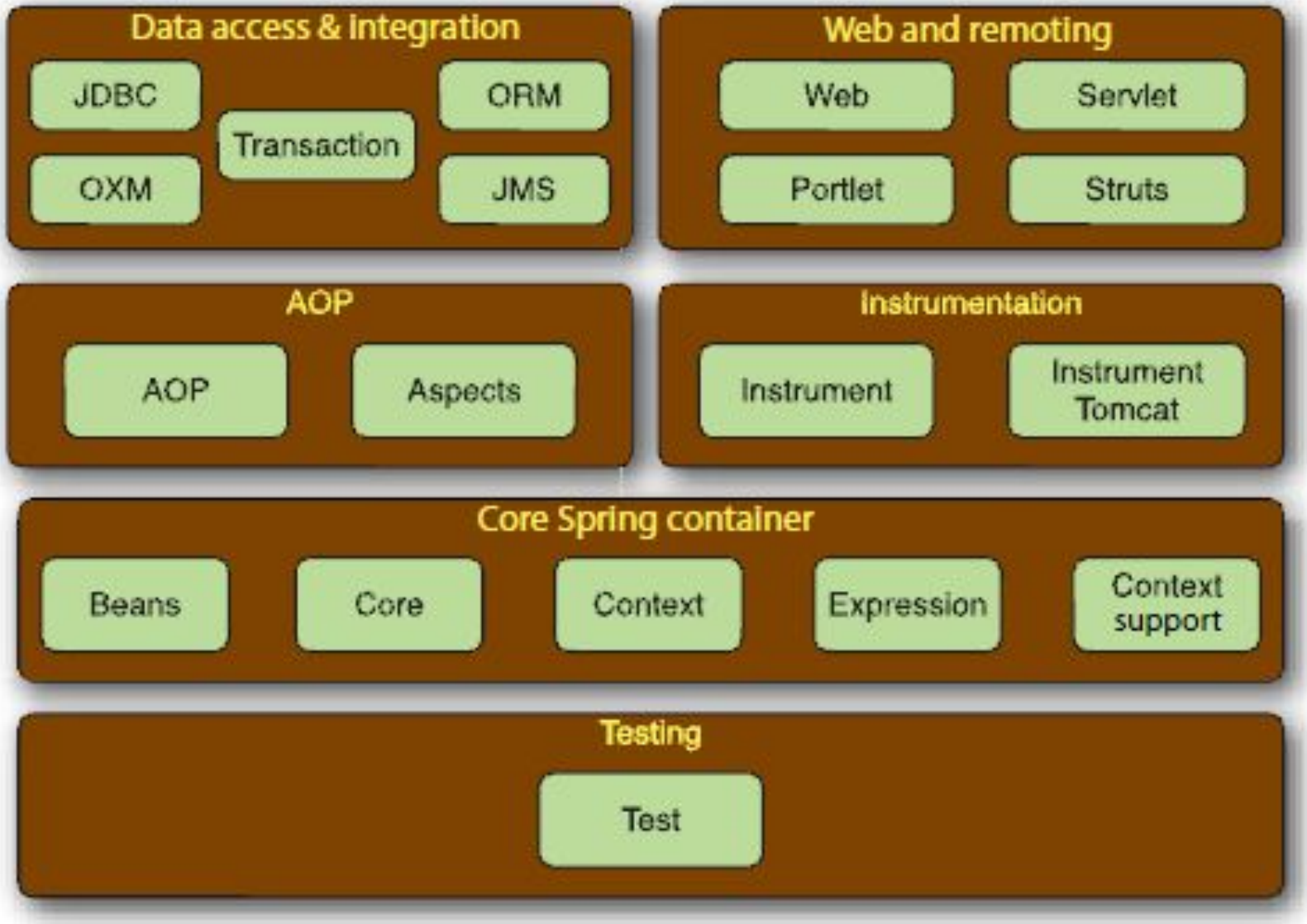

mockito

```java
@Test
public void getByAccountIdTest(  )
{
    logger.debug("test 1 DAO started");

    IClient cl = new ClientImpl();
    cl.setAccount(123);

    IClientDAO dao = mock(IClientDAO.class);
    when(dao.getByAccountId(123)).thenReturn(cl);

    IClient clnt = dao.getByAccountId(123);
    assertEquals(123, clnt.getAccount());

    logger.debug("test 1 DAO passed");
}
```


spring

Принципы Spring Framework

- Dependency Injection
- Aspect-Oriented programming


Data access & integration
JDBC
Transaction
ORM
OXM
JMS
Web and remoting
Web
Servlet
Portlet
Struts
AOP
AOP
Aspects
Instrumentation
Instrument
Instrument Tomcat
Core Spring container
Beans
Core
Context
Expression
Context support
Testing
Test