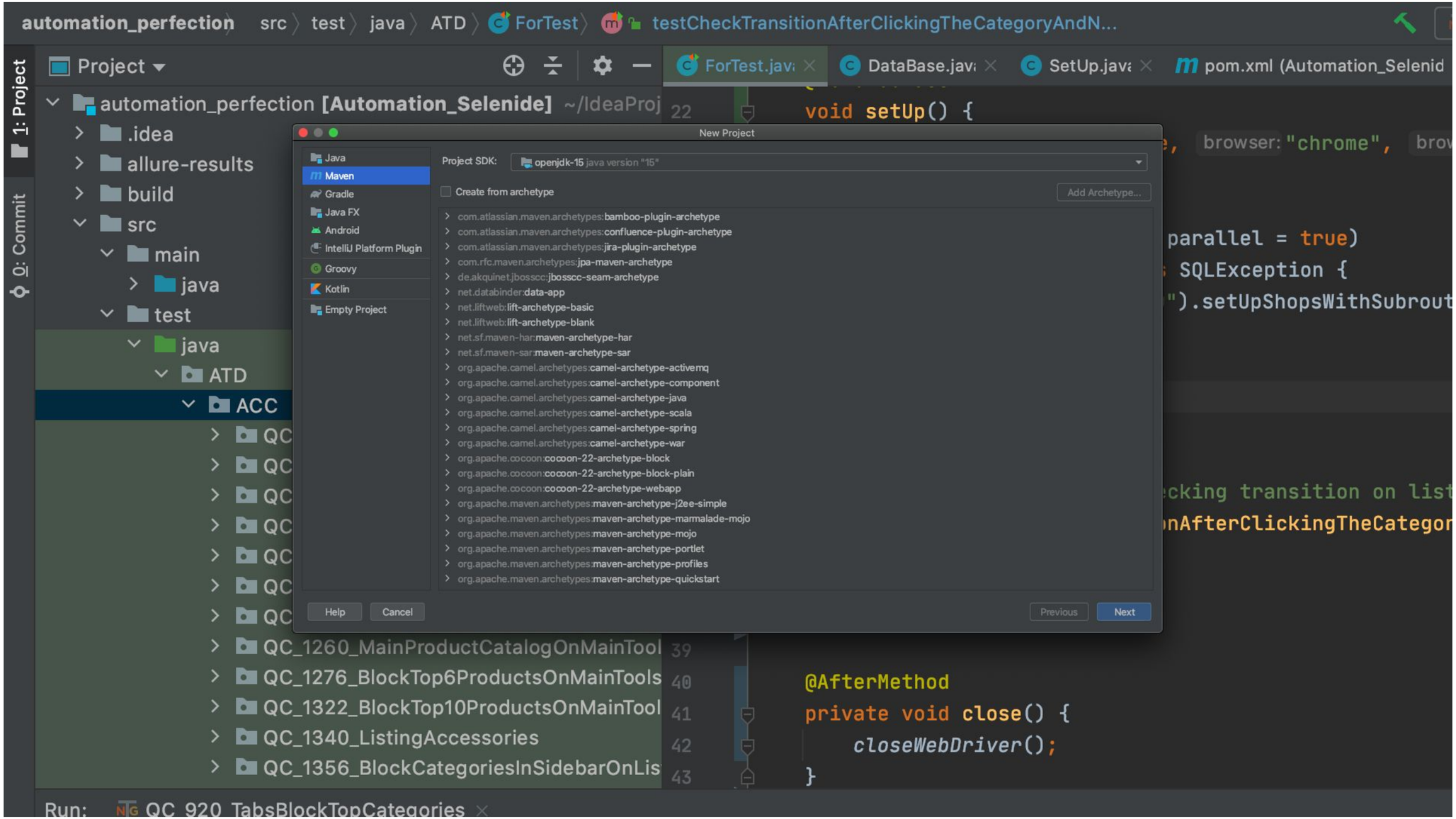
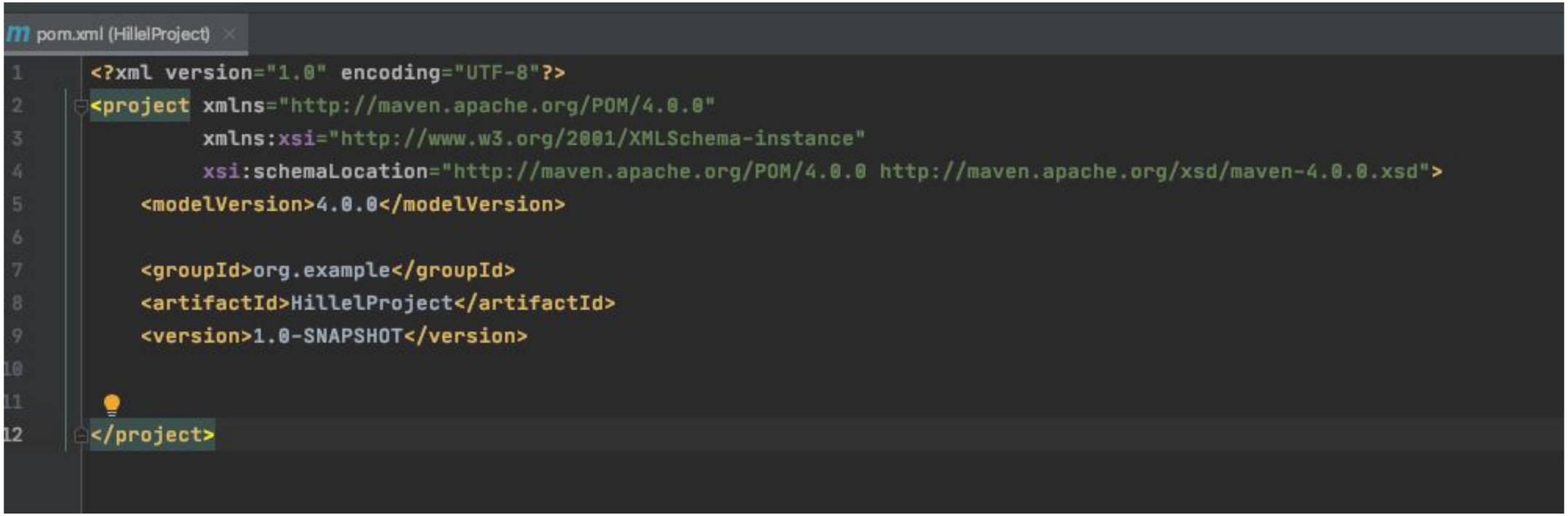

automation_perfection
src
test
java
ATD
ForTest
testCheckTransitionAfterClickingTheCategoryAndN...
Project
ForTest.java
DataBase.java
SetUp.java
pom.xml (Automation_Selenid
automation_perfection [Automation_Selenide] ~/IdeaProj
.idea
allure-results
build
src
main
java
test
java
ATD
ACC
QC_1260_MainProductCatalogOnMainTool
QC_1276_BlockTop6ProductsOnMainTools
QC_1322_BlockTop10ProductsOnMainTool
QC_1340_ListingAccessories
QC_1356_BlockCategoriesInSidebarOnLis
Run:
QC_920_TabsBlockTopCategories
1: Project
0: Commit
New Project
Java
Maven
Gradle
Java FX
Android
IntelliJ Platform Plugin
Groovy
Kotlin
Empty Project
Project SDK:
openjdk-15 java version "15"
Create from archetype
Add Archetype...
com.atlassian.maven.archetypes:bamboo-plugin-archetype
com.atlassian.maven.archetypes:confluence-plugin-archetype
com.atlassian.maven.archetypes:jira-plugin-archetype
com.rfc.maven.archetypes:jpa-maven-archetype
de.akquinet.jbosscc:jbosscc-seam-archetype
net.databinder:data-app
net.liftweb:lift-archetype-basic
net.liftweb:lift-archetype-blank
net.sf.maven-har:maven-archetype-har
net.sf.maven-sar:maven-archetype-sar
org.apache.camel.archetypes:camel-archetype-activemq
org.apache.camel.archetypes:camel-archetype-component
org.apache.camel.archetypes:camel-archetype-java
org.apache.camel.archetypes:camel-archetype-scala
org.apache.camel.archetypes:camel-archetype-spring
org.apache.camel.archetypes:camel-archetype-war
org.apache.cocoon:cocoon-22-archetype-block
org.apache.cocoon:cocoon-22-archetype-block-plain
org.apache.cocoon:cocoon-22-archetype-webapp
org.apache.maven.archetypes:maven-archetype-j2ee-simple
org.apache.maven.archetypes:maven-archetype-marmalade-mojo
org.apache.maven.archetypes:maven-archetype-mojo
org.apache.maven.archetypes:maven-archetype-portlet
org.apache.maven.archetypes:maven-archetype-profiles
org.apache.maven.archetypes:maven-archetype-quickstart
Help
Cancel
Previous
Next
void setUp() {
browser:"chrome",
parallel = true)
SQLException {
").setUpShopsWithSubrout
cking transition on list
nAfterClickingTheCategor
@AfterMethod
private void close() {
closeWebDriver();
}

Файл
pom.xml

Информация для программного проекта, поддерживаемого Maven, содержится в XML-файле с именем *pom.xml* (от Project Object Model). При исполнении Мавен проверяет прежде всего, содержит ли этот файл все необходимые данные и все ли данные синтаксически правильно записаны.



1.
Корневой элемент

Корневой элемент *<project>*, в котором прописана схема облегчающая редактирование и проверку, и версия POM.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

2. Заголовок

Внутри тэга *project* содержится основная и обязательная информация о проекте:

В *Maven* каждый проект идентифицируется парой *groupId, artifactId*.

Во избежание конфликта имён, *groupId* - наименование организации или подразделения и обычно действуют такие же правила как и при именовании пакетов в Java - записывают доменное имя организации или сайта проекта.

artifactId - название проекта.

Внутри тэга *version* хранится версия проекта.

```xml
<groupId>org.example</groupId>
<artifactId>HillelProject</artifactId>
<version>1.0-SNAPSHOT</version>
```

3. Тэг *packaging*

Тэг <*packaging*> определяет какого типа файл будет создаваться как результат сборки. Возможные варианты *pom, jar, war, ear.*

Тэг является необязательным. Если его нет, используется значение по умолчанию - *jar.*

4. Описание проекта

Также добавляется информация, которая не используется самим *Maven*, но нужна для программиста, чтобы понять, о чём этот проект:

```
<name>hillel project</name> название проекта для человека
<description>test project development</description> Описание проекта
<url>https://odessa.ithillel.ua/</url> сайт проекта
```

5. Зависимости

Зависимости - следующая очень важная часть *pom.xml* - тут хранится список всех библиотек (зависимостей) которые используются в проекте. Каждая библиотека идентифицируется также как и сам проект - тройкой *groupId, artifactId, version* (GAV). Объявление зависимостей заключено в тэг *<dependencies>...</dependencies>*.

Кроме GAV при описании зависимости может присутствовать тэг *<scope>*. Он задаёт, для чего библиотека используется.

6. Тэг <*build*>

Тэг <*build*> не обязательный, так как существуют значения по умолчанию. Этот раздел содержит информацию по самой сборке:

- где находятся исходные файлы,
- где ресурсы,
- какие плагины используются.

TestNG

TestNG — это тестовый фреймворк, он помогает нам удовлетворить многие наши потребности в тестировании. TestNG широко используется вместе с Selenium. Хотите знать, что означает NG? Это значит **“Next Generation”** (“Следующее поколение”). TestNG похож на JUnit, но он более мощный, когда дело касается управления потоком выполнения вашей программы. Архитектура фреймворка помогает нам сделать тесты более структурированными и обеспечить лучшие точки валидации.

Некоторые особенности TestNG, заслуживающие внимания:

- Мощные и разнообразные аннотации для поддержки ваших тест-кейсов.
- Параллельное выполнение тестов, использование зависимостей между тестами.
- Гибкость выполнения ваших тестов на разных наборах данных, через файл TestNG.xml или через концепцию поставщиков данных (data-provider).
- Группировка и приоритизация тест-кейсов.
- Генерация HTML-отчётов, настройка с помощью различных плагинов.
- Генерация логов выполнения тестов.
- Лёгкая интеграция с Eclipse, Maven, Jenkins и др.

Аннотация — это метка, которая предоставляет дополнительную информацию о классе или методе. Для аннотаций используется префикс «@». TestNG использует аннотации, чтобы помочь в создании надёжной структуры тестов. Давайте посмотрим на аннотации TestNG, используемые для автоматизации тестирования с Selenium.

@Test

Это самая важная аннотация в TestNG, в которой находится основная логика теста. Все автоматизируемые функции находятся в методе с аннотацией @Test. Она имеет различные атрибуты, с помощью которых может быть настроен запуск метода.

@BeforeTest

Метод с этой аннотацией запускается перед запуском первого метода с аннотацией @Test. Вы можете использовать эту аннотацию в TestNG с Selenium для настройки браузера. Например, запустить браузер и развернуть его на весь экран, установить специфичные настройки браузера и т.д.

@AfterTest

Методы, помеченные этой аннотацией, запускаются после всех @Test-методов вашего теста. Это полезная аннотация, которая пригодится для предоставления результатов выполнения тестов. Вы можете использовать эту аннотацию, чтобы создать отчёт о ваших тестах и отправить его заинтересованным сторонам по электронной почте.

@BeforeMethod

Методы с этой аннотацией запускаются перед каждым @Test-методом. Вы можете использовать её, чтобы перед выполнением теста проверить соединение с базой данных. Или, например, при тестировании функциональности, зависимой от логина пользователя, поместить сюда код для входа в систему.

@AfterMethod

Методы с этой аннотацией запускаются после каждого @Test-метода. Эту аннотацию можно использовать для создания скриншотов при каждом выполнении теста.

@BeforeClass

Метод с этой аннотацией выполнится перед первым тестовым методом в текущем классе. Эту аннотацию можно использовать для настройки свойств браузера, инициализации драйвера, открытия браузера с нужным URL-адресом и т.д.

@AfterClass

Метод с этой аннотацией выполнится после последнего тестового метода в текущем классе. Эта аннотация в TestNG может использоваться для выполнения действий по очистке ресурсов после выполнения теста, таких как закрытие драйвера и т.п.

@BeforeSuite

Набор тестов (suite) может состоять из нескольких классов, эта аннотация запускается перед всеми тестовыми методами всех классов. **Эта аннотация помечает точку входа при запуске.** Аннотацию @BeforeSuite в TestNG можно использовать для выполнения общих функций, таких как настройка и запуск Selenium или удалённых веб-драйверов и т.д.

@AfterSuite

Эта аннотация в TestNG запускается после запуска всех методов тестирования во всех классах. Эта аннотация может использоваться для очистки перед завершением тестов, когда у вас используется несколько классов, например, закрытие драйверов и т. д.

@BeforeGroups

TestNG может объединять тесты в группы с помощью атрибута group в аннотации @Test.

Аннотация @BeforeGroups в TestNG помогает запустить определённые действия перед указанной группой тестов.

@AfterGroups

Эта аннотация запускается после выполнения всех тестовых методов указанной группы.

- BeforeClass
- BeforeGroups
- BeforeMethod
- Test
- AfterMethod
- AfterGroups
- AfterClass
- AfterTest
- AfterSuite

```
<BeforeSuite>
    <BeforeTest>
        <BeforeClass>
            <BeforeGroups>
                <BeforeMethod>
                    <Test>
                </AfterMethod>
            </AfterGroups>
        </AfterClass>
    </AfterTest>
</AfterSuite>
```

Атрибуты, используемые с аннотациями в TestNG

У аннотаций в TestNG есть атрибуты, которые можно использовать для настройки. Они помогают настроить порядок выполнения тестовых методов.

- **description**: можно указать описание тестового метода.

  Например, @Test(description=”этот тест проверяет вход в систему”).

- **alwaysRun**: этот атрибут гарантирует, что тестовый метод будет выполнен всегда, даже в случае падения тестов, от которых он зависит. Когда значение атрибута true, этот метод будет запускаться всегда.

  Например, @Test(alwaysRun= true).

- **dataProvider**: задаёт имя поставщика данных (data provider) для тестового метода.

  Предположим, что вы собираетесь запускать свои тесты в нескольких браузерах, тогда в тестовом методе с атрибутом dataProvider, можно добавить параметры для браузера и его версии, которые будут передаваться в метод поставщиком данных. В этом случае тест, содержащий этот атрибут, будет использовать эти входные данные для запуска тестов в нескольких браузерах.

  Например, @Test(dataProvider=”cross-browser-testing”).

- **dependsOnMethods**: предоставляет информацию о порядке выполнения тестов. Тест с этим атрибутом будет выполнен, только если успешно выполниться тест, от которого он зависит. Если тест, от которого зависит метод, падает, то тест не запускается.

  Например, @Test (depenOnmethod = “login”).

- **groups**: помогает сгруппировать ваши тестовые методы, ориентированные на одну функциональность, в одну группу.

  Например, @Test(groups=”Payment_Module”).

  Этот атрибут также позволяет управлять тем, какие тесты запускать. При запуске тестов можно игнорировать какие-то группы или, наоборот, запустить только некоторые группы. Всё, что нужно сделать, это указать нужные группы в файле TestNG.xml. В теге include указать группы, которые необходимо запустить, а в теге exclude, которые надо игнорировать.

- **dependsOnGroups**: выполняет функции двух, вышеупомянутых атрибутов, то есть определяет зависимость тестового метода от указанной группы. Этот тестовый метод будет запущен только после того, как указанная группа тестов будет выполнена.

  Например, @Test (depenOnMethods = «Payment_Module»).

- **priority**: помогает нам определить приоритет тестовых методов. Когда TestNG выполняет тестовые методы, он может делать это в произвольном порядке. В сценарии, где вы хотите, чтобы ваши тесты выполнялись в нужном порядке, вы можете использовать атрибут priority. Приоритет по умолчанию для всех тестовых методов равен 0. Сначала выполняются тесты с меньшим значением priority.

  Например, @Test (priority = 1), @Test (priority = 2). В этом случае сначала будет выполнен тест с приоритетом, равным единице, а потом тест с приоритетом два.

- **enabled**: этот атрибут используется, когда вам нужно игнорировать и не запускать определённый тест. Всё, что вам нужно сделать, это установить его в false. Например, @Test(enabled= false).

- **timeout**: определяет время, за которое должен выполниться тест. Если выполнение теста превышает время, определённое атрибутом, то тест завершится с ошибкой с выбросом исключения org.testng.internal.thread.ThreadTimeoutException Например, @Test(timeOut= 500). **Обратите внимание, что время указывается в миллисекундах**.

- **invocationCount**: работает точно так же, как цикл. Тест будет запущен столько раз, сколько указано в invocationCount.

  Например, @Test(invocationCount = 5), будет запущен 5 раз.

- **invocationTimeOut**: используется вместе с вышеуказанным атрибутом invocationCount. Значение этого атрибута вместе с invocationCount указывает на то, что тест будет запущен столько раз, сколько указано в invocationCount, и в течение времени, указанного в атрибуте invocationTimeOut.

  Например, @Test(invocationCount = 5, invocationTimeOut = 20).

- **expectedExceptions**: помогает обрабатывать исключения, выброс которых ожидается в тестовом методе. Если исключение, указанное в атрибуте, выброшено тестовым методом, то тест прошёл успешно. В противном случае, отсутствие исключения или выброс другого исключения, не указанного в атрибуте, провалит тест.

  Например, @Test(expectedExceptions = {ArithmeticException.class }).

@DataProvider

Метод с этой аннотацией используется для предоставления данных тестовому методу, в котором задан атрибут dataProvider. Этот метод помогает в создании тестов, управляемых данными, в которые может передаваться несколько наборов входных значений. Метод должен возвращать двумерный массив или объект.

У аннотации @DataProvider есть два атрибута:

- **name** — этот атрибут используется для указания имени поставщика данных. Если не указано, то по умолчанию используется название метода.
- **parallel** — этот атрибут позволяет запускать тесты параллельно с разными данными. Наличие этого атрибута является одним из преимуществ TestNG перед Junit. По умолчанию false.

@Parameters

Эта аннотация позволяет вам передавать параметры в ваши тесты через файл TestNG.xml. Это удобно, когда нужно передать ограниченное количество данных в ваши тесты. В случае сложных и больших наборов данных лучше использовать аннотацию @DataProvider или Excel.

```java
public static void main(String[] args) {
    String str = "Love Java";
    int n = str.length();
    System.out.println(n % 6);
}
```

```java
public static void main(String[] args) {
    int x =10;
    switch (x){
        case 5:
            ++x;
            break;
        case 10:
            x+=10;
            break;
        case 20:
            x=0;
    }
    System.out.println(x);
}
```

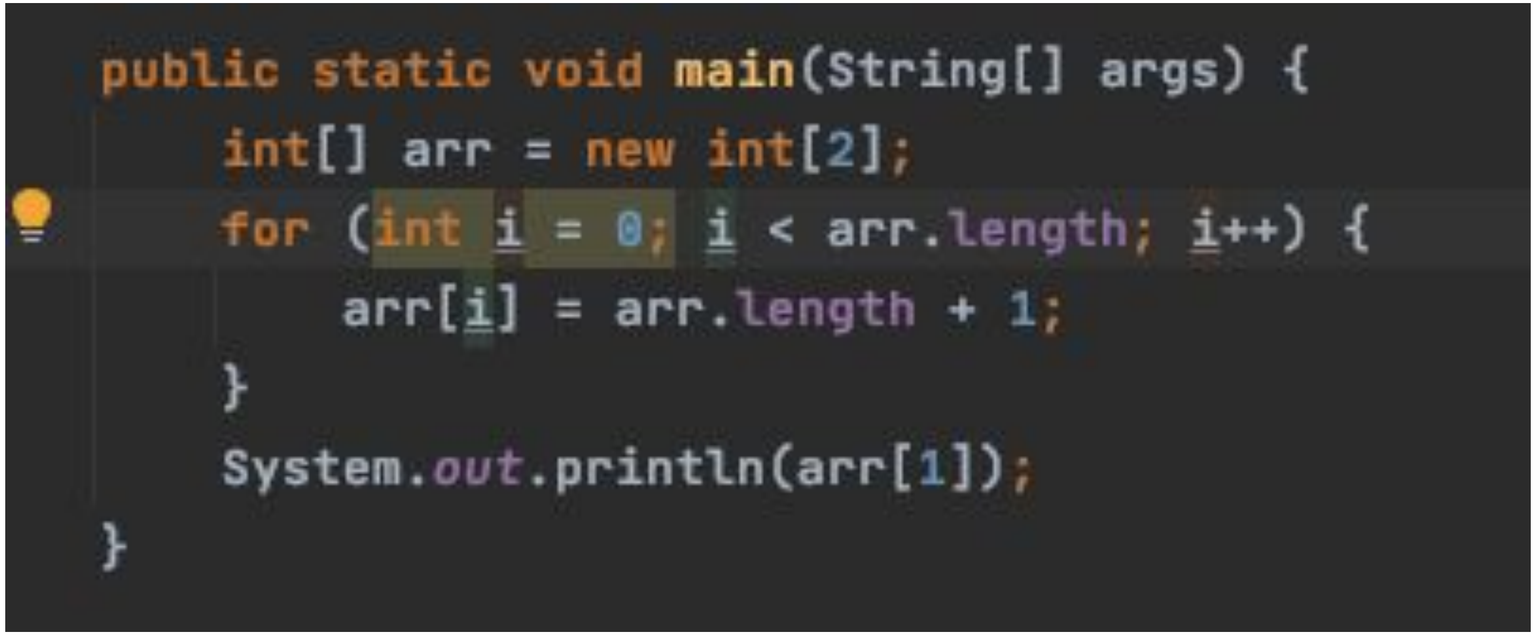
```java
public static void main(String[] args) {
    int x =4;
    int y =0;
    switch (--x){
        case 3:
            y+=x;
        case 4:
            y+=x;
        default:
            y++;
            break;
    }
    System.out.println(y);
}
```

```java
public static void main(String[] args) {
    int a = 20;
    int b = 8;
    System.out.println(a / --b);
}
```

```java
public static void main(String[] args) {
    int x = 1;
    for (int i = 3; i < 5; i++) {
        x += 1;
    }
    System.out.println(x);
}
```

```java
public static void main(String[] args) {
    int x = 0;
    switch (x) {
        case 1:
            System.out.println(1);
        case 0:
            System.out.print(1);
        case 2:
            System.out.println(3);
    }
}
```

```java
public static void main(String[] args) {
    System.out.println(4/0);
}
```


public static void main(String[] args) {
int[] arr = new int[2];
for (int i = 0; i < arr.length; i++) {
arr[i] = arr.length + 1;
}
System.out.println(arr[1]);
}

```java
public static void main(String[] args) {
    String str = "i love java";
    System.out.println(str.length() + 9);
}
```

```java
public static void main(String[] args) {
    int count = 0;
    while (count != 8) {
        ++count;
    }
    System.out.println(count++);
}
```