Лабораторная работа №1

«Сборка Java-проекта с использованием Gradle»

Цель работы:

Ознакомиться с основными принципами автоматической сборки проектов с использованием системы Gradle на примере простейшего проекта Java.

Краткие теоретические сведения

Gradle — система автоматической сборки, построенная на принципах Apache Ant и Apache Maven. Gradle была разработана для расширяемых много-проектных сборок, и поддерживает инкрементальные сборки, определяя, какие компоненты дерева сборки не изменились и какие задачи, зависимые от этих частей, не требуют перезапуска. Основные плагины предназначены для разработки и развертывания Java, Groovy и Scala приложений, но готовятся плагины и для других языков программирования.

Рассмотрим пример проекта Java, в котором используется стандартная структура директорий Maven для исходных кодов и ресурсов. Такая структура проекта включает в себя следующие директории:

```
src\main\java
src\main\resources
src\test\java
src\test\resources
```

Простейший файл конфигурации `build.gradle` для проекта без внешних зависимостей будет содержать единственную строку:

```
apply plugin: 'java'
```

Система Gradle позволяет использовать для проектов структуру каталогов, отличающуюся от конвенции Maven. Например, для проекта, в котором исходный код находится в каталоге `src\java`, а не в `src\main\java`, содержимое файла ***`build.gradle`*** будет выглядеть следующим образом:

```
apply plugin: 'java'
sourceSets.main.java.srcDirs = ['src\java']
```

Для сборки Java-кода необходимо использовать команду ***gradle build***. Данная команда компилирует, тестирует и упаковывает код в JAR-файл.

Результат сборки доступен в каталоге `build`. Здесь находится несколько директорий, среди которых три наиболее значимые:

- `Classes` - скомпилированные .class файлы, генерируемые во время сборки Java-кода
- `Reports` - отчеты в течении сборки (такие как отчеты о тестировании)
- `Libs` - библиотеки для сборки проекта (обычно в виде JAR и/или WAR файлов)

Для запуска проекта необходимо использовать плагин `application`, предварительно прописав эту особенность в файле `build.gradle`.

```
apply plugin: 'application'
```

Также необходимо указать класс, который будет вызываться первым (содержит метод main). Например:

```
mainClassName = 'hello.HelloWorld'
```

Для выполнения и запуска проекта используется команда ***gradle run.***

Сборка проекта с Gradle Wrapper

Gradle Wrapper является предпочтительным способом для начала Gradle сборки. Он содержит bat-скрипты для Windows и shell-скрипты для OS X и Linux. Эти скрипты позволяют запускать сборку с Gradle без необходимости установки самого Gradle в систему. Чтобы это стало возможным требуется добавить следующий блок в build.gradle:

```
task wrapper(type: Wrapper) {
    gradleVersion = '1.11'
}
```

Для загрузки и инициализации wrapper-скриптов необходимо запустить команду ***gradle wrapper,*** после чего Gradle Wrapper будет доступен для сборки проекта. Далее можно добавить его в систему контроля версий и каждый, кто клонирует проект, сможет его собрать точно таким же способом без необходимости устанавливать и настраивать Gradle определенной версии. Gradle Wrapper можно использовать наравне с установленным Gradle.

Для выполнения задачи сборки необходимо выполнить команду ***gradlew build***, для выполнения и запуска проекта требуется использование команды ***gradlew run***.

Задание:

Выполнение лабораторной работы должно производиться только средствами командной строки!

1. Протестируйте правильность установки Gradle, запустив в командной строке: ***gradle***. Должно отобразиться сообщение:

```
:help
Welcome to Gradle 3.0Ghd.
To run a build, run gradle <task> ...
To see a list of available tasks, run gradle tasks
To see a list of command-line options, run gradle --help
BUILD SUCCESSFUL
Total time: 2.675 secs
```

2. Проверьте доступность задач Gradle, выполнив команду: ***gradle tasks***. Ознакомьтесь со списком доступных задач.

Примечание: Несмотря на то, что эти задачи доступны, они не представляют большого значения без конфигурации для сборки проекта. С `build.gradle` файлом, некоторые задачи будут более полезны. Список задач будет увеличиваться при добавлении плагинов в `build.gradle`.

3. В выбранном вами каталоге проекта создайте следующую структуру каталогов

```
└── src
    └── main
        └── java
            └── hello
```

4. В директории `hello` создайте два класса: `HelloWorld.java` и `Greeter.java`.

src/main/java/hello/HelloWorld.java

```
package hello;
public class HelloWorld {
public static void main(String[] args) {
    Greeter greeter = new Greeter();
    System.out.println(greeter.sayHello());
    }}
```

src/main/java/hello/Greeter.java

```
package hello;
public class Greeter {
    public String sayHello() {
        return "Hello world!";
    }}
```

5. В корневой папке проекта создайте файл `build.gradle`, в котором пропишите строку: `apply plugin: 'java'`. Снова выполните команду ***gradle tasks*** и ознакомьтесь со списком доступных задач.

6. Выполните команду ***gradle build***. Ознакомьтесь с содержимым каталога `build`.

7. К классу `HelloWorld` добавьте внешнюю библиотеку для получения текущего времени и даты Joda Time:

```
package hello;
import org.joda.time.LocalTime;
public class HelloWorld {
public static void main(String[] args) {
    LocalTime currentTime = new LocalTime();
    System.out.println("The current local time is: " +
currentTime);
    Greeter greeter = new Greeter();
    System.out.println(greeter.sayHello());
}}
```

8. Добавьте Joda Time компилируемую зависимость в сборке, прописав в файле `build.gradle`:

```
dependencies {
compile "joda-time:joda-time:2.2"}
```

Также необходимо добавить источники сторонних библиотек:

```
repositories {
mavenLocal()
mavenCentral()}
```

Примечание: Блок `repositories` означает, что сборка должна разрешать зависимости из Maven Central репозитория. Gradle опирается в основном на многие соглашения и возможности, определенные в инструменте сборки Maven, включая использование Maven Central как источник библиотек зависимостей.

В блоке `dependencies` вы описываете единственную зависимость Joda Time. В частности, вы запрашиваете (справа налево) версию 2.2 библиотеки joda-time в joda-time группе. Ключевое слово `compile` обозначает доступность библиотеки во время компиляции.

9. Назначьте имя для JAR-файла

```
jar {
    baseName = 'gs-gradle'
    version =  '0.1.0'}
```

`Jar` блок определяет, как JAR файл будет назван. В данном случае `gs-gradle-0.1.0.jar`.

10. Запустите ***`gradle build`***. Gradle должен будет загрузить Joda Time зависимость из репозитория Maven Central и успешно собрать проект.

11. Выполните и запустите проект средствами `gradle`.

12. Выполните и запустите проект средствами `gradle wrapper`.

13. Поменяйте структуру каталогов и выполните сборку и выполнение проекта.