Лабораторная работа №6

«Переменные окружения. Управление Java проектами с помощью ant-скрипт»

Краткие теоретические сведения:

Ant - это инструмент, который призван автоматизировать процесс сборки ПО. В идеале Ant сделает из всех ваших исходных файлов инсталлируемый пакет, готовый к записи на CD. Новички Ant'а иногда спрашивают, что может Ant, чего нельзя сделать другими способами. Например, если я хочу откомпилировать несколько Java файлов и сделать Jar из полученных файлов классов, то я могу просто набрать команды руками:

```
$ javac -d build src/Main.Java src/Utils.src/Java GUI.Java

$ cd build

$ jar -cvf ../mystuff.jar Main.class Utils.class GUI.class
```

Это сработает, но этот метод может порождать ошибки и утомительный, он не станет лучше, если ваша система станет включать все больше и больше Java файлов.

Конечно, вы можете засунуть все это в сценарий оболочки или Perl:

```
#!/bin/sh

javac -d build src/Main.Java src/Utils.Java src/GUI.Java

cd buildre are ten things you need to know about this tool.

jar -cvf ../mystuff.jar Main.class Utils.class GUI.class
```

Проблема в том, что оболочка и Perl являются инструментами общего назначения - наряду со сборкой ПО, вы можете написать web сервер или ПО баз данных на Perl. Ant специализируется на одном - на сборке ПО. Он может, и использует любую возможность чтобы помочь вам собрать ваше ПО. Ant делает это потому что знает, что вы занимаетесь сборкой ПО - вот почему вы должны использовать Ant для сборки и никакие другие средства.

Это отличие является несущественным, когда вы начинаете маленький проект, но с большими системами специализированные средства, такие как Ant могут быть выбором между жизнью и смертью.

Ant немного похож на фабрику. Сырьем для него является то, что создают программисты - исходный код на Java, программы, написанные на других языках, дескрипторы развертывания и т.д. Что должно получится на выходе - скомпилированные программы, готовые к использованию.

В процессе получения из исходного кода готового продукта, Ant'у необходима помощь в виде build файла. Build файл (также известный как "Ant файл", или "build сценарий" ("сценарий сборки") или "файл build.xml") показывает Ant'у что надо делать, чтобы превратить то что есть (как правило, исходный код) в то что вы хотите. Build файл похож на детальный план - он говорит вам как собрать из частей единое целое.

Build файл содержит правила, которые указывают Ant'у делать вещи наподобие таких:

- В соответствии с тем что мне нужно, скопируй Russell.jar в каталог релиза...
- кстати, сначала ты должен собрать Russell.jar;

- кстати, для того, чтобы собрать Russell.jar, примени команду jar к *.class файлам
- кстати, тебе нужно скомпилировать class файлы, если ты этого еще не сделал.
- Для того, чтобы сделать *.class файлы, запусти компилятор Java для *.java файлов.

Хватит теории, давай посмотрим на реальный, хотя и очень маленький Ant файл. Нижеследующее является "Hello world" build файлов.

```
<project name="MyProject" default="all">
<target name="all" description="Do the entire build" >

<echo>I'm building my software</echo>
</target>
</project>
```

Как видишь, это XML файл, содержащий элемент <project>. <project> содержит 2 атрибута:

- Имя;
- Задание по умолчанию, под названием "all".

Имя является просто обозначением, обычно оно нигде не используется. Задание по умолчанию - это имя того задания, которое вы хотите, чтобы Ant выполнял по умолчанию. Это задание Ant выполнит если вы не укажите ему, что собирать. Наполнением build файла являются элементы <target>. По сути, этот элемент <target> значит следующее: чтобы собрать нечто под названием "all", выполни команду <echo>. Для того чтобы заставить Ant что-то сделать, вы должны выполнить команду Ant и указать ей какой файл содержит инструкции сборки:

```
$ Ant -f build.xml
Buildfile: build.xml

all:
     [echo] I'm building my software

BUILD SUCCESSFUL
Total time: 1 second
```

Итак, что здесь произошло?

- Ant нашел build файл и прочитал его;
- Так как ты не указал ему, что делать, он решил выполнить задание по умолчанию "all";
- Ant собрал цель путем выполнения команды <echo>.
- Давай рассмотрим файл с более чем одним заданием:

```
<project name="02.depend" default="all" >

      <description>This is a project which uses dependancies</description>

      <target name="make.dirs" description="Make a dir">

     <mkdir dir="build"/>
      </target>

      <target name="all" description="Do the build"  depends="make.dirs">
     <echo>I'm building my software</echo>
      </target>


</project>
```

Так как имя по умолчанию для build файла - это build.xml, то мы можем опустить аргумент -f и просто набрать Ant без параметров:

```
$ Ant
Buildfile: build.xml

make.dirs:
    [ mkdir] Created dir: /home/russ/Ant_example/02.depend/build

all:
     [echo] I'm building my software

BUILD SUCCESSFUL
Total time: 1 second
```

Этот build файл содержит два задания, уже знакомое "all" и новое, "make.dirs". "make.dirs" вызывает встроенную в Ant команду mkdir - Ant называет его встроенные команды задачами - для создания новой директории. Обратите внимание на тот факт, что задание "all" указывает на то, что оно зависит от задания "make.dirs". Эта зависимость означает, что для того чтобы собрать "all", мы должны сначала собрать "make.dirs". Идея зависимости является ключевой в работе Ant'a - большая часть работы по написанию Ant файлов состоит в построении сложных паутин зависимости.

Наш следующий пример содержит 3-е задание, под названием "clean":

```
<project name="nodefault" default="all" >

      <target name="make.dirs" description="Make some dirs">
     <mkdir dir="build"/>

      </target>

      <target name="all" description="Do the build"  depends="make.dirs">
     <echo>I'm building my software</echo>
      </target>

      <target name="clean" description="Clean up">

     <echo>I'm cleaning up.</echo>
     <delete dir="build"/>
      </target>

</project>
```

"clean" просто удаляет каталог сборки, который создается заданием "make.dirs". Ты обычно включаешь "clean" для уборки после сборки всех сгенерированных файлов. Но замечу, что "clean" не является заданием по умолчанию и задание по умолчанию не зависит от него. Так как же ты сможешь выполнить "clean"? Путем набора следующего:

```
$ Ant clean
Buildfile: build.xml

clean:
     [echo] I'm cleaning up.

BUILD SUCCESSFUL
Total time: 1 second
```

Итак, давай попытаемся сделать что-нибудь полезное с Ant, например, откомпилируем что-то из Java кода:

```
<project name="MyProject" default="all" >

      <target name="make.dirs" description="Make some dirs">
     <mkdir dir="build"/>
     <mkdir dir="build/class"/>
```

```
      </target>

      <target name="compile" description="compile java"  depends="make.dirs">
     <javac destdir="build/class">
    <src path="src"/>
     </javac>
      </target>

      <target name="all" description="Do the build"  depends="compile"/>

      <target name="clean" description="clean up">
     <echo>I'm cleaning up.</echo>
     <delete dir="build"/>
</target>

  </project>
```

Похоже, что мы имеем задание по умолчанию "all", зависящее от "compile", которое вызывает задачу javac, встроенную в Ant. Вот что произойдет, когда мы запустим этот сценарий:

```
Ant
Buildfile: build.xml

make.dirs:
    [mkdir] Created dir: /home/rolsen/Documents/Ant/04.compile/build
    [mkdir] Created dir:
/home/rolsen/Documents/Ant/04.compile/build/class

compile:
    [javac] Compiling 1 source file to /home/russ/build/class

all:

BUILD SUCCESSFUL
Total time: 2 seconds
```

Существует 2 интересные особенности задачи javac. Во-первых, ты можешь не указывать непосредственно имена java файлов, которые хочешь скомпилировать - достаточно просто указать каталог. Во-вторых, javac может определить, когда class файл существует и новее, чем java файл, в этом случае, она ничего не откомпилирует. Ты можешь проверить это, выполнив строку дважды - во втором случае javac ничего не откомпилирует. Эти 2 особенности являются примерами приемов сборки ПО, упоминавшихся ранее. Так как Ant знает, что то, что вы делаете, это попытка собрать ПО, он пропускает все, что уже сделано. Сейчас давай обратимся к яду, отравляющему жизнь Java программиста - classpath. Как ты знаешь, classpath в Java это список Jar файлов и/или каталогов, который сообщает Java где искать файлы библиотек. Проблема состоит в том, что в сложных системах classpath может становиться очень длинным и запутанным. Ant предоставляет тебе 2 метода для определения classpath, или прямо в самой javac:

```
<project name="04b.classpath" default="all" >

      <target name="make.dirs" description="Make some dirs">
     <mkdir dir="build"/>

     <mkdir dir="build/class"/>
      </target>

      <target name="all" description="compile java" depends="make.dirs">
     <javac destdir="build/class" classpath="extra.jar">
    <src path="src"/>
     </javac>

      </target>
```

```
</project>
```

...или ты можешь указать classpath отдельно, что удобно есть путь длинный или используется в нескольких заданиях.

```
<project name="04c.classpath" default="all" >

      <target name="make.dirs" description="Make some dirs">

     <mkdir dir="build"/>
     <mkdir dir="build/class"/>
      </target>

 <path id="myclasspath" >
     <fileset dir="lib">
    <include name="**/*.jar"/>

     </fileset>
      </path>

      <target name="all" description="compile java" depends="make.dirs">
     <javac destdir="build/class" classpathref="myclasspath">
    <src path="src"/>
</javac>

      </target>

</project>
```

Недостаток этого в том, что Ant принимает это значение без дополнительной проверки, существует ли jar файл, который ты указал в сценарии. Поэтому, если ты ошибся в имени jar файла, Ant не покажет тебе место ошибки - проверяй то, что вводишь. Java приложения как правило поставляются упакованными в jar файлы. Jar файлы похожи на привычные нам zip файлы - один jar файл может содержать любое количество файлов собранных вместе и сжатых. Ant предоставляет специальную задачу для создания jar файлов:

```
<project name="05.jar" default="all" >

      <target name="make.dirs" description="Make some dirs">
     <mkdir dir="build"/>
     <mkdir dir="build/class"/>
      </target>

      <target name="compile" description="compile java" depends="make.dirs">

<javac destdir="build/class">
    <src path="src"/>
     </javac>
      </target>

      <target name="stuff.jar" description="make jar file" depends="compile">
     <jar destfile="build/stuff.jar">
    <fileset dir="build/class">
  <include name="**/*.class"/>
  <exclude name="**/CVS"/>
    </fileset>

     </jar>
      </target>

      <target name="all" description="Do the build" depends="stuff.jar"/>
```

```
</project>
```

Заметьте, что атрибут "distfile" сообщает Ant'у имя создаваемого jar файла, в то время как файлы, помещаемые в jar файл, указываются через элемент fileset. Filese'ыt могут быть использованы во многих задачах, требующих списков файлов и являются очень мощным методом указания файлов по маске. Jar задача имеет те же особенности, что и javac: если jar файл существует и имеет более новую дату и время, то задача jar сэкономит время и ничего не сделает.

Задание(выполняется без использования IDE):

1. Разработать консольное приложение выводящее произвольный текст на экран(текст поступает как параметр командной строки). Класс содержащий метод main находится в пакете «org.bsuir.gr######.фамилия.laba4.proj1». Класс содержащий метод выводящий строки на экран находится в пакете «org.bsuir.gr######.фамилия.laba4.proj1.util».
2. Проект собирается, упаковывается в jar архив и запускается ANT-скриптом написанным студентом самостоятельно!.
3. Разработать консольное приложение использующее проект 1 для вывода строк на экран(пользователь вводит строки с клавиатуры, они выводятся методом размещенным в классе проекта 1). Класс, содержащий метод main находится в пакете «org.bsuir.gr######.фамилия.laba4.proj2». Класс, содержащий метод, вызывающий метод стороннего класса находится в пакете «org.bsuir.gr######.фамилия.laba4.proj2.helper».
4. Проект собирается, упаковывается в jar архив и запускается ANT-скриптом написанным студентом самостоятельно!.
5. На **отличную оценку** дополнительно проделайте все то же самое с MAVEN (в этом случае использование IDE разрешается).
6. Составить отчет содержащий: 6 различных UML-диаграмм для проекта 2, листинг приложений и ANT(+MAVEN)-скриптов.