构建工具：Java

在Java世界中，

- 编译器 `javac` 将源文件（ `.java` ）变成 `.class` 文件；
- 该 `jar` 工具将class文件打包成 `.jar` 文件；
- 该 `java` 命令运行类文件或 jar 文件。

Java 运行时环境 (JRE) 仅包含 `java` 命令，如果您不想进行任何开发，那么这就是运行 Java 应用程序所需的全部内容。许多操作系统允许您双击 jar 文件（至少是包含名为 a 的特殊文件的文件 `manifest` ）以在 JRE 中运行它们。

Java 开发工具包 (JDK) 包含工具 `javac` 和 `jar` JRE。这就是你需要用java开发的东西。

`maven` 是一个Java包管理器和构建工具。它不是 Java 发行版的一部分，因此您需要单独安装它。

您可以在您的虚拟机中进行此练习，也可以在您自己的计算机上进行此练习，您可能已经为 OOP/算法单元安装了 Java，并且可以使用您最喜欢的编辑器。在这两种情况下，练习的工作方式应该完全相同，这里没有任何特定于 POSIX 的内容。

在 Debian 上安装

在 Debian 上，安装 `openjdk-17-jdk` 和 `maven` 软件包。这应该已经完成设置，以便您可以开始使用，但如果您安装了*多个版本*的 Java，您可能需要设置 `JAVA_HOME` 和 `PATH` 变量以指向您的安装。

例如：

```
export JAVA_HOME='/usr/lib/jvm/java-17-openjdk'
export PATH="${PATH}:${JAVA_HOME}/bin"
```

Advanced note

Debian 还有一个名为 的特殊命令 `update-alternatives` ，可以帮助您管理替代开发环境。阅读手册页！

在您自己的机器上安装

使用您的操作系统附带的任何包管理器。如果您不能并且必须手动安装：

- 下载OpenJDK发行版
- 将其解压到某处
- 将二进制文件文件夹添加到您的 `PATH`
- 将变量设置 `JAVA_HOME` 为指向解压 JDK 的文件夹。

要安装 maven，请按照这些说明进行操作，其中再次涉及下载 ZIP 文件，将其解压缩到某个位置，然后将子 `bin` 文件夹放在您的 `PATH` .

注意： `JAVA_HOME` 必须正确设置maven才能工作。

运行专家

打开 shell 并输入 `mvn archetype:generate` . 这使您可以从 *archetype 生成工件*，这是 maven 所说的，用于使用 maven 文件创建新文件夹。

如果您收到"未找到"错误，则很可能 Maven `bin` 文件夹不在您的路径上。如果您使用的是 POSIX 系统并使用了包管理器，则应该自动设置，但如果您已经下载并解压了 maven，那么您必须将 `export PATH="$PATH:..."` 三个点替换为文件夹的路径，并且最好把那条线也放进去 `~/.profile` 。

|||高级 在 Windows 上，如果您必须使用它，请在线搜索如何设置路径变量的说明，或者您可以将文件从资源管理器窗口拖放 `mvn.cmd` 到 Windows CMD 终端中，它应该粘贴完整的内容路径，然后按空格键并输入要传递的参数。|||

第一次运行maven会下载很多库。

Maven 将首先显示人类已知的所有原型的列表（统计时为 3046），但您只需按 ENTER 键即可使用默认值 2098（"快速启动"）。Maven 现在询问您要使用的版本，再次按 ENTER 键。

现在，您必须为您的项目输入三元组（groupId、artifactId、version）——这并不重要，但我建议如下：

```
groupId: org.example
artifactId: project
version: 0.1
```

对于以下问题，只需再次按 ENTER 键，直到收到成功消息。

Maven 已经创建了一个以您的artifactId 命名的文件夹，但是您可以根据需要移动和重命名它，只要您从该文件夹内运行它，maven 就不会介意。使用 `cd project` 或任何你称之为的东西进入文件夹。

如果您使用的是 POSIX shell，则应 `find .` 显示文件夹中的所有内容（在 Windows 中， `start .` 请在资源管理器中打开它）：

```
.
./src
./src/main
./src/main/java
./src/main/java/org
./src/main/java/org/example
./src/main/java/org/example/App.java
./src/test
./src/test/java
./src/test/java/org
./src/test/java/org/example
./src/test/java/org/example/AppTest.java
./pom.xml
```

这是标准的 Maven 文件夹结构。您的 java 源位于下面 `src/main/java` ，默认包名称是 `org.example` 或您作为 groupId 放置的任何名称，因此主文件是当前的 `src/main/java/org/example/App.java` 。由于从 IDE 或具有“折叠”路径的编辑器（例如 VS code）内部开发 Java 是很常见的，因此这种文件夹结构不是问题，尽管它在终端上有点笨重。

POM 文件

`pom.xml` 在编辑器中看看。您需要了解的重要部分是：

工件的标识符（组 ID、工件 ID、版本）：

```
<groupId>org.example</groupId>
<artifactId>project</artifactId>
<version>0.1</version>
```

构建属性确定要编译的 Java 版本（通过向编译器传递标志）。不幸的是，默认的maven模板似乎使用版本7（由于复杂的原因称为1.7），但版本8早在2014年就发布了，这对我们来说足够稳定，所以请将1.7更改为1.8（有一些重大变化）从版本 9 开始，这里不再赘述）：

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

依赖项部分是您添加要使用的库的位置。默认情况下，您的项目使用 `junit` 单元测试框架 - 请注意，这是声明 `<scope>test</scope>` 它仅用于测试，而不是项目本身。在声明项目的真正依赖项时，不要添加此行。

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

该 `<plugins>` 部分包含 maven 用于编译和构建项目的插件。本节不是强制性的，但包含它是为了将插件"锁定"到特定版本，这样即使发布了插件的新版本，也不会改变您的构建方式。

您应该在此处添加的一件事如下 `exec-maven-plugin` ，以便您可以实际运行您的项目：

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>3.0.0</version>
    <configuration>
        <mainClass>org.example.App</mainClass>
    </configuration>
</plugin>
```

重要的一行是 `mainClass` 您使用函数设置为类的全名（带有路径组件） `main()` 。

编译、运行和开发

`mvn compile` 编译该项目。第一次执行此操作时，会下载很多插件，之后速度会很快。像 一样 `make` ，它只编译自上次运行以来已更改的文件，但如果这不同步（例如因为您中途取消了编译），则将 `mvn clean` 删除所有已编译的文件，以便下一次编译将重建所有内容。

该 `App.java` 文件包含一个基本的"Hello World！"程序（看看这个文件）。 `mvn exec:java` 如果您已按上述方式设置了插件，则可以运行编译后的项目。第一次运行它并下载了所需的所有文件后，来自 maven 本身的行将以 `[INFO]` or `[ERROR]` 或类似的开头，因此没有任何前缀的行将由程序本身打印。您应该在屏幕上看到 hello world 消息。

现在的开发工作流程如下：进行编辑，然后运行 `mvn compile test exec:java` 重新编译，运行测试，然后运行程序。（就像 一样 `make` ，您可以在一个命令上放置多个目标，并用空格分隔。）

`mvn test` 运行测试 `src/test/java` 。已经为您创建了一个示例测试（请查看）。

`mvn package` 在文件夹中创建项目的 jar 文件 `target/` 。

我假设您将把 Java 项目存储在 git 存储库中。在这种情况下，您应该 `.gitignore` 在与 相同的文件夹中创建一个文件并向其中 `pom.xml` 添加该行 `target/` ，因为您不希望在存储库中编译类和其他临时文件并构建报告。文件 `src/` 夹、文件 `pom.xml` 和 `.gitignore` 文件本身都应该签入存储库。

练习：对java源代码进行更改，然后重新编译并使用maven运行。

添加依赖项

Thymeleaf是一个 Java 模板库。例如，它可以让您编写模板文件或字符串（取决于您的库的语法）

```
Hello, ${name}!
```

您稍后可以使用特定的名称值进行渲染。这是创建 Web 应用程序的标准方法之一，例如要显示某人的个人资料页面，您可以编写一个页面模板来处理布局、样式、链接等，但使用字段的模板变量（姓名、电子邮件、照片）等）当某人访问特定人员的个人资料页面时您呈现的内容。您将在明年的 SPE 项目中更详细地看到这一点。

要使用 Thymeleaf 或任何其他库，您首先必须将其添加到您的 pom 文件中。转到 mvnrepository.org并搜索 Thymeleaf，然后找到最新的稳定（“发布”）版本。有一个框，您可以在其中复制 `<dependency>` 块以粘贴到您的 pom 文件中。接下来 `mvn compile` 将下载 thymeleaf 及其所有依赖项。

接下来，创建一个 `unit` 在文件夹中调用的模板文件 `src/main/resources/templates` （您必须首先创建该文件夹），并将以下行放入其中：

```
Unit: [(${name})]

In this unit, you will learn about:

[# th:each="topic: ${topics}"]
  - [(${topic})]
[/]
```

这是 thymeleaf“文本”语法，其中第一行呈现变量的值，倒数第三行是相当于“for”循环的模板，该循环为列表中的每个元素（或其他元素）呈现一次其内容。集合数据结构）。

Thymeleaf 需要知道在哪里可以找到它的模板文件，在这个例子中，我们将演示从类路径加载资源，因为这是在 java 应用程序中使用资源的正确方法（对于 Web 应用程序有特殊的注意事项，但它们无论如何，通常最终都会使用类路径）。

在 Java 源文件中，您现在可以执行以下操作。首先，您需要导入：

```
import java.util.List;
import java.util.Arrays;

import org.thymeleaf.TemplateEngine;
import org.thymeleaf.context.Context;
import org.thymeleaf.templatemode.TemplateMode;
import org.thymeleaf.templateresolver.ClassLoaderTemplateResolver;
```

和代码：

```
ClassLoaderTemplateResolver resolver = new ClassLoaderTemplateResolver();
resolver.setTemplateMode(TemplateMode.TEXT);
resolver.setPrefix("templates/");

TemplateEngine engine = new TemplateEngine();
engine.setTemplateResolver(resolver);

Context c = new Context();
c.setVariable("name", "Software Tools");
List<String> topics = Arrays.asList("Linux", "Git", "Maven");
c.setVariable("topics", topics);
String greeting = engine.process("unit", c);

System.out.println(greeting);
```

编译并运行它，您应该看到：

```
Unit: Software Tools

In this unit, you will learn about:

  - Linux
  - Git
  - Maven
```

让我们看看代码是如何工作的。

1. 模板解析器是一个类，当您给模板命名时（此处为“单元”），它会找到模板。在本例中，我们使用一个解析器来加载类路径，因此我们只需将模板文件放在 `src/main/resources` ;下的某个位置即可。我们告诉它我们希望将模板文件视为文本（例如不是 HTML），并且模板文件位于名为 的子文件夹中 `templates` 。
2. 模板引擎是在解析器找到源文件后执行渲染模板工作的类。
3. 要呈现模板，您需要一个供解析器查找的模板名称，以及一个上下文 - 您可以在其上设置键/值参数的对象。在本例中，我们将键“名称”设置为“软件工具”，将键“主题”设置为三个主题的列表。键的名称和类型显然必须与模板文件中的内容匹配。

练习：通过创建一个单元类来重写此示例，使其更加面向对象：

```
public class Unit {
    private String name;
    private List<String> topics;
    public Unit(String name, List<String> topics) {
        this.name = name;
        this.topics = topics;
    }
    public String getName() { return this.name; }
    public List<String> getTopics() { return this.topics; }
}
```

您仍然需要一次 `setVariable` 调用，并且在模板中语法 `[(${unit.name})]` 应转换为对 getter 的调用。

Advanced note

Java 的最新版本有一些奇妙的东西，叫做“ `records` 让你的生活变得*更*轻松”。上面的所有代码都翻译为：

```
public record Unit(String name, List<String> topics) {}
```

不幸的是，对最新 Java 版本的支持有点不稳定（在*现*实世界中更糟）。您需要删除在 pom.xml 中添加的 `maven.compiler.target` 和位，并将其替换为新的： `maven.compiler.source`

```
<maven.compiler.release>17</maven.compiler.release>
```