SE1-Play
Aufgabe B1

Die "magischen" Befehle wie **make**, **mk**, **build** und **wipe** sind meistens Kurzbefehle, die verwendet werden, um häufige Aufgaben im Build-Prozess zu automatisieren.

1. Was sind make, mk, build, wipe und ähnliche Befehle?

Diese Befehle sind normalerweise Shell-Skripte oder Shell-Funktionen, die eine Reihe von Befehlen ausführen, um bestimmte Aufgaben auszuführen, wie Kompilieren, Testen oder Reinigen des Projekts. Sie sind nützlich, um häufige Prozesse in der Projektentwicklung zu vereinfachen und zu automatisieren.

2. Wo werden sie definiert?

Diese Befehle werden üblicherweise in Shell-Skripten oder Shell-Konfigurationsdateien definiert. Sie können in einer Datei wie **.bashrc**, **.bash_profile**, **.zshrc**, **.env/project.sh**, oder anderen Skripten innerhalb des Projekts definiert sein. Die genaue Stelle hängt von Ihrer Umgebung und der Art der Projekteinrichtung ab.

3. In welcher Programmiersprache sind sie implementiert?

Diese Befehle werden in der Regel in Shell-Sprachen wie Bash oder Zsh implementiert. Sie können aber auch in anderen Skriptsprachen wie Python oder Perl implementiert sein, je nachdem, welche Tools Ihr Projekt verwendet.

4. Wie finden Sie ihre Definition?

- Wenn der Befehl ein Alias ist, können Sie in Ihrer Shell-Konfiguration nachsehen, indem Sie **alias** ausführen oder die Konfigurationsdatei durchsuchen.
- Wenn der Befehl ein Skript ist, können Sie das Skript finden, indem Sie nach Dateinamen suchen oder innerhalb Ihres Projekts nach relevanten Dateien schauen.

5. Was macht der gezeigte Code-Fragment?

```
local module_jars=( $(find libs/*/ -name '*.jar' 2>/dev/null) )
local entries=(
    "bin/classes"
    "bin/test-classes"
    "bin/resources"
    ${module_jars[@]}
)

[ "$(uname | grep 'CYGWIN\|MINGW')" ] && local sep=';' || local sep=':'

if [ -z "$CLASSPATH" ]; then
    export CLASSPATH=""
    for entry in ${entries[@]}; do
        [ ! -z "$CLASSPATH" ] && CLASSPATH+=${sep}
        CLASSPATH+=$entry
    done
fi
```

Das Code-Fragment erstellt einen Classpath, der mehrere Verzeichnisse und Dateien enthält.

Hier ist eine Erklärung des Codes:

- **Zeile 1**: Sucht alle JAR-Dateien in Unterverzeichnissen von **libs** und speichert sie im Array **module_jars**.
- **Zeile 2**: Erstellt ein Array **entries**, das die Verzeichnisse **bin/classes**, **bin/test-classes**, **bin/resources** und die gefundenen JAR-Dateien enthält.
- **Zeile 5**: Bestimmt das Trennzeichen für den Classpath (semikolon **;** für Windows, Doppelpunkt **:** für Unix/Linux/MacOS).
- **Zeile 7**: Falls **CLASSPATH** leer ist, initialisiert er ihn mit einer leeren Zeichenfolge.
- **Zeile 8-11**: Fügt jedes Element im **entries**-Array zum Classpath hinzu. Wenn der Classpath nicht leer ist, wird das entsprechende Trennzeichen eingefügt.

  Zusammen ergibt dies einen Classpath, der verwendet wird, um Java-Klassen, Ressourcen und Bibliotheken zu laden.