**Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»**Факультет программной инженерии и компьютерной техники  
Дисциплина: Основы профессиональной инженерии

**Лабораторная работа №3**

Вариант 3069

Выполнили: Гаврилин Олег Сергеевич,

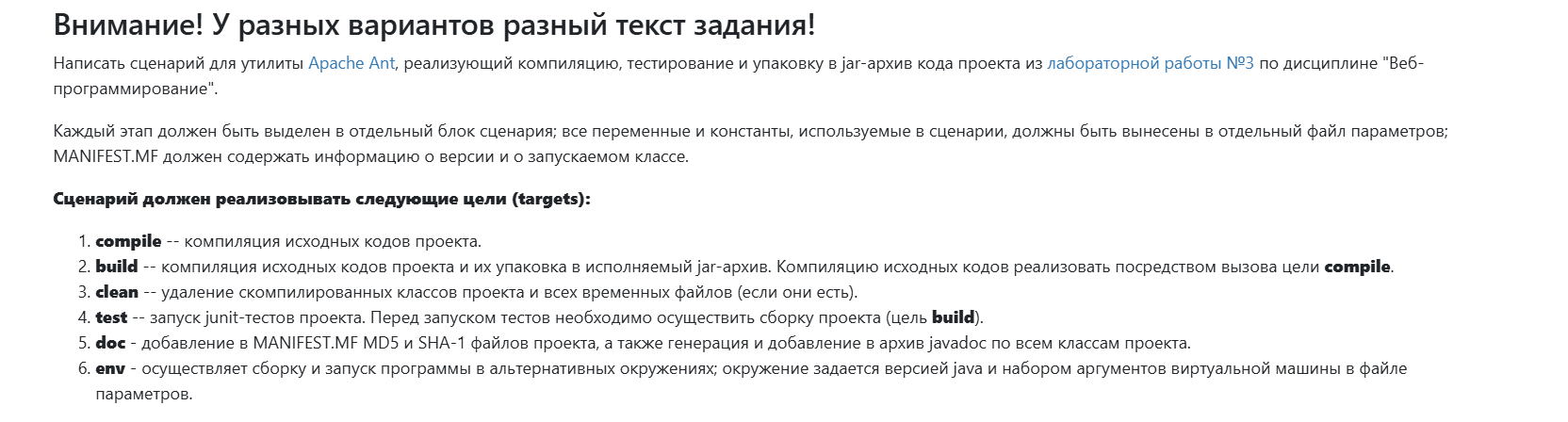
Зорин Георгий Юрьевич

группа P3230

Преподаватель: Цопа Е. А.

г. Санкт-Петербург, 2025 год

# **Вариант**



# **Ход работы**

**Build.gradle:**

import java.security.MessageDigest

def static hash(File file, String algorithm) {

MessageDigest md = MessageDigest.getInstance(algorithm)

file.withInputStream { is ->

byte[] buffer = new byte[8192]

int bytesRead

while ((bytesRead = is.read(buffer)) != -1) {

md.update(buffer, 0, bytesRead)

}

}

return md.digest().encodeHex().toString()

}

plugins {

id 'java'

id 'war'

}

group 'org.example'

version '1.0-SNAPSHOT'

repositories {

mavenCentral()

}

ext {

junitVersion = '5.10.2'

}

war {

archiveFileName = 'lab3.war'

}

tasks.withType(JavaCompile).configureEach {

options.encoding = 'UTF-8'

}

dependencies {

providedCompile 'jakarta.enterprise:jakarta.enterprise.cdi-api:3.0.0'

providedCompile 'jakarta.persistence:jakarta.persistence-api:3.0.0'

providedCompile 'jakarta.servlet:jakarta.servlet-api:5.0.0'

implementation 'org.glassfish:jakarta.faces:3.0.3'

implementation 'org.glassfish.web:jakarta.servlet.jsp.jstl:2.0.0'

implementation 'org.hibernate.orm:hibernate-core:6.6.1.Final'

implementation 'org.postgresql:postgresql:42.7.4'

testImplementation "org.junit.jupiter:junit-jupiter-api:${junitVersion}"

testRuntimeOnly "org.junit.jupiter:junit-jupiter-engine:${junitVersion}"

testImplementation 'org.mockito:mockito-core:5.12.0'

}

//

test {

useJUnitPlatform()

}

tasks.register("testAfterBuild") {

group = "verification"

description = "build, then testing"

dependsOn("build")

finalizedBy("test")

}

// doc task

javadoc {

source = sourceSets.main.allJava

destinationDir = layout.buildDirectory.dir("docs/javadoc").get().asFile

options.encoding = 'UTF-8'

}

tasks.register('doc', Jar) {

group = 'Documentation'

description = 'Packing Javadoc archive with MD5/SHA-1 hashes in MANIFEST.MF'

dependsOn javadoc

archiveClassifier.set('javadoc')

from("docs/javadoc")

manifest {

attributes(['Manifest-Version': '1.0'])

def classDir = layout.buildDirectory.dir("classes/java/main").get().asFile

def filesToHash = fileTree('src').include('\*\*/\*.java', '\*\*/\*.jsp') +

fileTree(classDir).matching { include '\*\*/\*.class' }

filesToHash.each { file ->

def base = file.name.replaceAll(/\.(java|class|jsp)/, '')

def md5Key = "MD5-${base}".toString()

def sha1Key = "SHA1-${base}".toString()

if (md5Key.length() <= 60 && sha1Key.length() <= 60) {

attributes.put(md5Key, hash(file, 'MD5'))

attributes.put(sha1Key, hash(file, 'SHA-1'))

}

}

}

}

def loadEnvFile(String path) {

def file = file(path)

if (!file.exists()) throw new GradleException("Env file not found: $path")

return file.readLines()

.collect { it.trim() }

.findAll { it && !it.startsWith('#') }

.collectEntries { line ->

def (k, v) = line.split('=', 2)

[(k): v]

}

}

tasks.register('env', JavaExec) {

group = 'Execution'

description = 'Run application with environment-specific JVM settings'

def profile = project.findProperty("profile") ?: "dev"

def envPath = "env/${profile}.env"

def envVars = loadEnvFile(envPath)

classpath = sourceSets.main.runtimeClasspath

jvmArgs = []

if (envVars["XMS"]) jvmArgs += "-Xms${envVars["XMS"]}"

if (envVars["XMX"]) jvmArgs += "-Xmx${envVars["XMX"]}"

if (envVars["CONFIG\_ENV"]) jvmArgs += "-Dconfig.env=${envVars["CONFIG\_ENV"]}"

if (envVars["ADD\_OPENS"]) jvmArgs += "--add-opens=${envVars["ADD\_OPENS"]}"

def javaVersionStr = envVars.get("JAVA\_VERSION")

def javaVersion = 17

if (javaVersionStr != null && javaVersionStr.isInteger()) {

javaVersion = javaVersionStr.toInteger()

} else if (javaVersionStr != null) {

throw new GradleException("JAVA\_VERSION must be an integer. Found: '${javaVersionStr}'")

}

javaLauncher = javaToolchains.launcherFor {

languageVersion = JavaLanguageVersion.of(javaVersion)

}

doFirst {

println "Run: $profile"

println "Java version: $javaVersion"

println "JVM args: $jvmArgs"

}

}

**Tests:**

package org.example.lab3;

import org.example.lab3.repository.ResultRepository;

import org.example.lab3.service.CheckHitService;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.mock;

public class DotCheckerTest {

ResultRepository mockRepo = mock(ResultRepository.class);

CheckHitService service = new CheckHitService(mockRepo);

@Test

void testFirstQuarter(){

assertTrue(service.checkDot(1f, 1f, 3f));

assertFalse(service.checkDot(4f, -1f, 3f));

}

@Test

void testFourthQuarter(){

assertTrue(service.checkDot(0.5f, -0.5f, 4f));

assertFalse(service.checkDot(4f, -1f, 3f));

}

}

**Вывод**

В этой лабораторной работе был создан сценарий сборки проекта с помощью Apache Ant. Он выполняет компиляцию кода, сборку исполняемого jar-файла с манифестом, запуск тестов через JUnit, очистку временных файлов, генерацию javadoc и запуск программы в разных окружениях. Все параметры вынесены в отдельный файл, что делает настройку удобной. Работа помогла понять, как автоматизировать сборку Java-приложений и запускать проект с разными настройками.