Szoftverfejlesztés – követelmények a házi feladat projektek elkészítéséhez

2022. március 27.

Kivonat

Ez a dokumentum a Szoftverfejlesztés tárgyból házi feladatként elkészítendő projektek formai és tartalmi követelményeit foglalja össze. A követelmények be nem tartása elégtelen osztályzatot jelent!

1. Bevezetés

A Szoftverfejlesztés tárgyból kiadott házi feladatok megoldásait egy Apache Maven 3 projekt formájában kell elkészíteni. A fejlesztés tetszőleges integrált fejlesztői környezetben történhet.

2. Funkcionális követelmények

A program meg kell, hogy feleljen az alábbiaknak:

- Tartalmazzon olyan funkcionalitást (üzleti logikát), mely tesztelhető, és hozzá egységteszteket kell készíteni.
- Rendelkezzen értelmes grafikus felhasználói felülettel, melynek megvalósítása az JavaFX felhasználásával kell, hogy történjen.
- A megvalósítás a modell-nézet-vezérlő (MVC) architekturális minta szerint kell, hogy történjen.
- Az adatok tárolása XML vagy JSON állományokban, vagy relációs adatbázisban a Spring Data JPA vagy a Jdbi révén kell, hogy történjen. (Kezeljen a program inputot és outputot is.)

3. Java-specifikus követelmények

A projekt összeállításához a JDK 11, 17 vagy 18 használatával kell, hogy történjen.

4. Apache Maven-specifikus követelmények

A `pom.xml` állományban kötelező az alábbi elemek megadása és megfelelő kitöltése:

- `description`
- `developers` (benne legalább a fejlesztő nevével és email címével)
- `properties/project.build.sourceEncoding`
- (`properties/maven.compiler.source` és `properties/maven.compiler.target`) vagy `properties/maven.compiler.release`
- `properties/exec.mainClass`

A projekt webhelyének előállításához legalább az alábbi jelentéskészítő bővítmények használata kötelező:

- Maven Javadoc Plugin
- Maven JXR Plugin
- Maven Checkstyle Plugin[1]
- Maven Surefire Report Plugin
- JaCoCo Maven Plugin

A projekthez használt összes adat-, konfigurációs és egyéb hasonló állományt az `src/main/resources/` könyvtárban kell elhelyezni, a teszteléshez használt állományokat pedig az `src/test/resources/` könyvtár alatt.

[1] A Checkstyle bővítményt kizárólag az API dokumentáció ellenőrzésére kell használni.

5. Hordozhatóság

A projekt lefordítható és működőképes kell, hogy legyen tetszőleges olyan környezetben, ahol rendelkezésre áll a JDK és az Apache Maven megfelelő verziója. Például az `mvn site` parancs parancssori végrehajtása a projekt könyvtárában Linux, macOS és Windows környezetben is a webhely előállítását kell, hogy eredményezze.

Állománynevekben kizárólag US-ASCII karakterek használhatóak.

6. Dokumentálás

Az alább felsorolt kivételektől eltekintve minden felső szintű típushoz (az-az osztályhoz és interfészhez) és tagjaikhoz részletes magyar vagy angol nyelvű API dokumentációt kell írni, melynek tartalmaznia kell a használathoz szükséges valamennyi tudnivalót. Nem szükséges dokumentálni

- az egységteszteket,
- a grafikus felhasználói felület osztályait,
- a `main(String[] args)` metódust,
- a nem `public` láthatóságú tagokat.

A dokumentációs megjegyzések meg kell, hogy feleljenek az órákon tárgyalt tartalmi és formai követelményeknek.

A szövegben igyekezni kell kihasználni a kiemelési lehetőségeket, például metódusok paramétereinek neveit `<code></code>` elemekben vagy a `{@code}` szövegközi címkével kell megadni, hivatkozásokhoz a `{@link}` szövegközi címkét kell használni. Érthetően kell fogalmazni és törekedni kell a helyes nyelvhasználatra.

A dokumentációt a Checkstyle bővítménnyel kell ellenőrizni, melyhez az alábbi tartalmú konfigurációs állományt kell használni:

```
<?xml version="1.0"?>
<!DOCTYPE module PUBLIC
        "-//Checkstyle//DTD Checkstyle Configuration 1.3//EN"
        "https://checkstyle.org/dtds/configuration_1_3.dtd">
<module name="Checker">
```

```
        <module name="SuppressWithPlainTextCommentFilter"/>
        <module name="JavadocPackage"/>
        <module name="TreeWalker">
            <module name="AtclauseOrder"/>
            <module name="JavadocMethod"/>
            <module name="JavadocStyle"/>
            <module name="JavadocType"/>
            <module name="JavadocVariable">
                <property name="scope" value="public"/>
            </module>
            <module name="MissingJavadocMethod"/>
            <module name="MissingJavadocPackage"/>
            <module name="MissingJavadocType"/>
        </module>
    </module>
```

Minden egyes csomagban el kell helyezni egy `package-info.java` megjegyzés állományt.

7. Naplózás

Kötelező a programban a naplózás, mely az alábbi programkönyvtárakkal történhet:

- Apache Log4j 2
- tinylog 2
- SLF4J *backend*ként az Apache Log4j 2 vagy tinylog 2 programkönyvtárakkal.

Naplózáshoz nem használhatók `System.out.println()` metódushívások! Az `src/main/resources/` könyvtárban biztosítani kell a használt naplózó programkönyvtárhoz egy konfigurációs állományt. Naplózni legalább a konzolra kell.

8. Egységtesztek

A program fő funkcióit biztosító osztályokhoz a JUnit 5 keretrendszerben kell egységteszteket készíteni. Nem szükséges egységteszteket írni például a

felhasználói felülethez és kivételosztályokhoz. Minden teszt végrehajtása sikeres kell, hogy legyen. Az egységtesztek mozgassák meg a tesztelt osztályok kódjának lehetőleg minél nagyobb részét. A tesztek lefedettségéről a JaCoCo Maven Plugin használatával kell jelentést készíteni, hogy az objektív módon ellenőrizhető legyen.

9. Állományok kezelése

Szigorúan tilos a forrásokban állományokhoz való hozzáféréshez platformspecifikus abszolút elérési útvonalak használata, mint például

```
File f = new File("C:\\file.txt");
```

vagy

```
OutputStream os =
    new FileOutputStream("/home/me/.history");
```

Állományokhoz kizárólag az osztálybetöltő révén férjünk hozzá. A programban tilos az `src/main/resources/` és az `src/test/resources/` könyvtár alatt elhelyezett állományokra közvetlenül hivatkozni. Ne feledjük, hogy ezek automatikusan a `target/classes/` illetve a `target/test-classes/` könyvtár alá lesznek másolva, ahol elérhetjük őket az osztálybetöltővel. Így például tilos az alábbi:

```
File file = new File("src/main/resources/file.txt");
```

10. JAR készítése

A `package` életciklus fázis végrehajtásakor létre kell hozni a Maven Shade Plugin segítségével egy, a függőségeket is tartalmazó végrehajtható JAR állományt. A JAR futtatása az alkalmazás indítását kell, hogy eredményezze.

11. .gitignore állomány

A projekt kell, hogy tartalmazzon egy `.gitignore` állományt, mely kizárja a fejlesztőeszközök – az integrált fejlesztői környezet és a Maven – által létrehozott állományok verziókezelés alá történő helyezését. Az állomány létrehozható a `https://gitignore.io/` szolgáltatás segítségével.

12. README.md állomány

A tárolóban el kell helyezni egy, a projekt tömör leírását tartalmazó `README.md` állományt is, melyben legalább a feladat eredeti szövegét meg kell adni.

13. Fejlesztés és megosztás

A projekt fejlesztése a GitHub Classroom-ban kell, hogy történjen az ehhez az oktató által biztosított privát tárolóban. A tárolóban jól nyomon követhető kell, hogy legyen a fejlesztés menete. (Nem elfogadható, ha a tároló egyetlen *commit*ot tartalmaz!)

A *commit*okhoz tömör és értelmes üzenetek kell, hogy tartozzanak.

Szigorúan tilos a tárolóba állományokat a GitHub felületén manuálisan feltölteni, minden Git művelet egy Git klienssel kell, hogy történjen.

A tárolóba kell, hogy kerüljön minden kód a határidőig.

14. Egyéb követelmények

Beadni kizárólag saját munkát lehet. A program meg kell, hogy feleljen a feladatban megfogalmazott elvárásoknak, hibamentesen kell, hogy biztosítson valamennyi megkövetelt funkciót.

Debrecen, 2022. március 27.