Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

Integrált fejlesztői környezet Java nyelvhez

Témavezető:
Pataki Norbert
docens, Ph.D.

Szerző:
Farkasch Zoltán László
Programtervező informatikus BSc.

Budapest, 2022

Tartalomjegyzék

Bevezetés

Szakdolgozatom célja egy Java nyelvre specifikálódott IDE (*Integrated Developer Environment* - Integrált fejlesztő környezet) létrehozása volt, ami amellett, hogy egy IDE alapvető funkcionalitásait, mint a fájlok megnyitását, szerkesztését, mentését, valamint a forráskód fordítását és futtatását implementálja, külön a Java nyelvre specifikált funkciókkal is segíti a fejlesztőt. Az ötlet, nem meglepően, Java kód írása közben keletkezett, amikor épp a sokadik boiler-plate kódszakaszt írtam. Bár erre például a Lombok-könyvtár[1] már kínál megoldást, ez lényegesen megnehezíti a program debuggolását, elemzését.

Az általam készített alkalmazás megpróbálja *getterek*, *setterek*, konstruktorok generálásával, automatikus újra-packageléssel, importok javításával és más egyéb segítő funkcionalitásokkal a Java-fejlesztők életét megkönnyíteni.

Természetesen már léteznek Javara specifikált fejlesztői környezetek, a szakdolgozat is egy ilyen (IntelliJ IDEA[2]) segítségével készült. Így a program megírásánál azt tűztem ki célul, hogy a többi nagyszabású, néha szükségtelenül sok elemet tartalmazó fejlesztői környezettel ellentétben nem szeretnék „fölösleges” funkciókat beépíteni. A program legyen gyors, ne legyen nagy gépigénye, legyen könnyen használható és tartalmazzon mindent, ami egy kisebb-szabású projekt elkészítéséhez szükséges. Belátásom szerint sikerült egy jó alternatívát nyújtani azoknak a Java-programozóknak, akik egy kisebb szabású hobbi-projektbe vágnának bele, vagy esetleg csak egyetlen egy fájlból álló Java programot szeretnének írni, szerkeszteni.

Felhasználói dokumentáció

A feladat

A cél egy integrált fejlesztői környezet (IDE) megvalósítása Java nyelvhez, amely Java specifikus funkciókat (*getterek, setterek*, konstruktrok generálása stb.) is implementál. Ennek elérése érdekében készítettem egy grafikus felületet, amely a középen található és ketté osztható kódolási felületből, a bal oldalt található oldalsó menüből és az oldal tetején található menü-fejlécből áll. A kódolási felület a kódot különböző szintaktikus feltételek alapján kiszínezi, és lehetővé teszi a kód szerkesztését. Az oldalsó menü segítségével tudjuk fordítani és futtatni a forráskódot, illetve itt láthatjuk és manipulálhatjuk majd a projektünk fájljait. A fejléc ad lehetőséget - többek között - fájlok és projektek létrehozására.

A környezet

Az alkalmazás **Windows 10** operációs rendszerre lett készítve. A futtatáshoz továbbá szükséges a **Java 8-as futtatási környezetének**, valamint **a Java 17.0.2-es fejlesztői környezetének** a telepítése. Ezek segítségével lesz elérhető a Java virtuális gép, amin fut a program, valamint így lesznek elérhetők számunkra a futtatáshoz szükséges parancsok. A Windows parancssorban el kell navigálni a **BaristaIDE.jar** állományt tartalmazó mappába, majd le kell futtatni a következő parancsot: **java -jar BaristaIDE.jar**. Ez után a program elindul.

Egy másik módja a program futtatására a forráskód közvetlen lefordítása. Ehhez szükségünk lesz a jar-állomány futtatásához szükséges Java-verziókra, valamint egy helyesen bekonfigurált **maven** verzióra. A maven 3.6.3-as verziójának a telepítése után be kell állítanunk a **JAVA_HOME** környezeti változót. Ennek a környezeti változónak a Java fejlesztői környezetünk **bin** mappájára kell mutatnia. Tehát, ha a Java fejlesztői környezet elérési útvonala: **C:\Users\Y2SMAR\.jdks\openjdk-17.0.2**, akkor a **JAVA_HOME** környezeti változó a következőképpen fog kinézni: **C:\Users\Y2SMAR\.jdks\openjdk-17.0.2\bin**. A **maven** bekonfigurálása után nyissunk egy parancssort a forráskód forrásmappájában és futtassuk le a következő két parancsot egymás után: **mvn clean install** majd **mvn javafx:run.** Ez után a program elindul.

Használat

Elindítás után

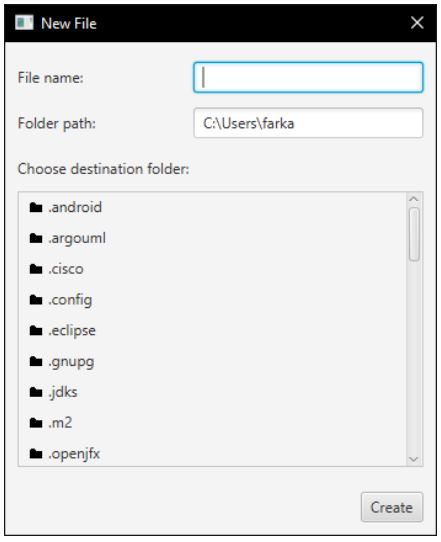
A program elindítása után az IDE teljes képernyős módban fog megnyílni. Látható lesz baloldalt az oldalsó menü, valamint a legfelül a menüsáv. A kódolási felület ekkor mindig üres. Az oldalsó menünek a *Compile, Run* és *Settings* gombjai inaktívak, be vannak szürkítve.

A felső menüsáv

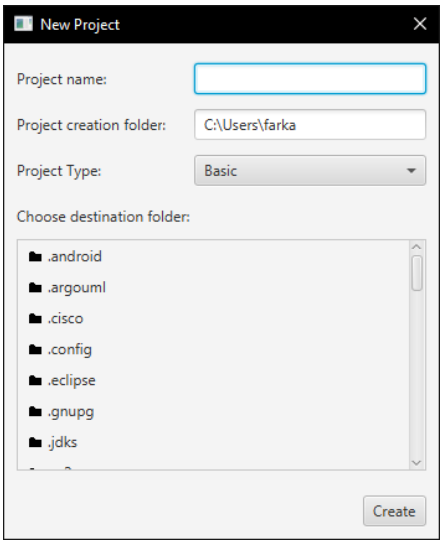
A felső menüsávban 2 opció található: *File és Help*. Ezekre kattintva egy leugró menüsáv jelenik meg további lehetőségekkel.

A ***File*** menü tartalma a következő: *New File, New Project, Open File, Load Project, Save*.

- A ***New File*** menüre kattintva megnyílik egy felugró ablak (1. ábra), ahol kiválaszthatjuk létrehozandó fájl nevét, és a létrehozási helyet. A létrehozandó fájl nevét kiterjesztéssel együtt kell megadni! A ***Create*** gombra kattintva a fájl létrejön és egyből megnyitásra kerül a kódolási felületen. Ha volt megnyitva projekt, az automatikusan bezáródik.
- A ***New Project*** menüre kattintva megnyílik egy felugró ablak (2. ábra), ahol kiválaszthatjuk a létrehozandó projekt nevét, és a létrehozási helyet. A ***Create*** gombra kattintva a program létrehozza a projektstruktúrát, és ezt meg is nyitja az oldalsó menüben. Ha meg volt nyitva projekt, vagy bármi más fájl, azok automatikusan bezáródnak.

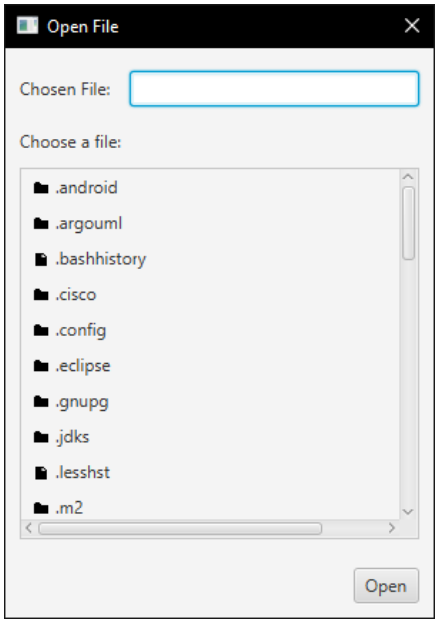


2. ábra: a felugró ***new File*** ablak

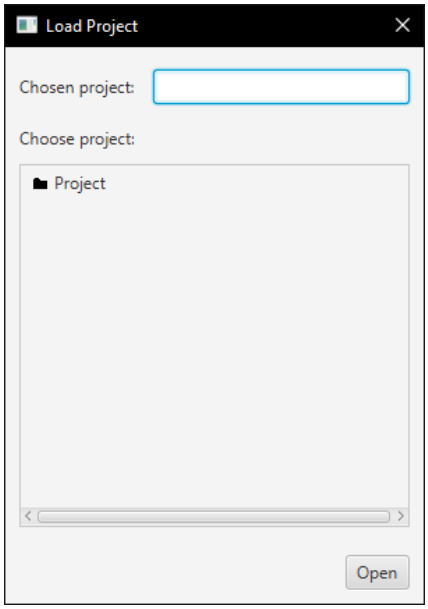


*1. ábra: a felugró **new Project** ablak*

- Az ***Open File*** menüre kattintva megnyílik egy felugró ablak (3. ábra), ahol kiválaszthatjuk melyik fájlt szeretnénk megnyitni. Az ***Open*** gombra kattintva a kiválasztott fájl megnyílik. Ha volt megnyitva projekt, az automatikusan bezáródik.
- A ***Load Project*** menüre kattintva megnyílik egy felugró ablak (4. ábra), ahol kiválaszthatjuk a megnyitni kívánt projektet. Az ***Open*** gombra kattintva a kiválasztott projekt megnyílik, az oldalsó menüben láthatóak lesznek a projektet alkotó mappák, fájlok. Ha volt megnyitva projekt, vagy bármi más fájl, akkor azok automatikusan bezáródnak.



*3. ábra: a felugró **Open File** ablak*



*4. ábra: a felugró **Load Project** ablak*

- A ***Save*** menüpontra kattintva az éppen szerkesztett fájlok mentésre kerülnek.

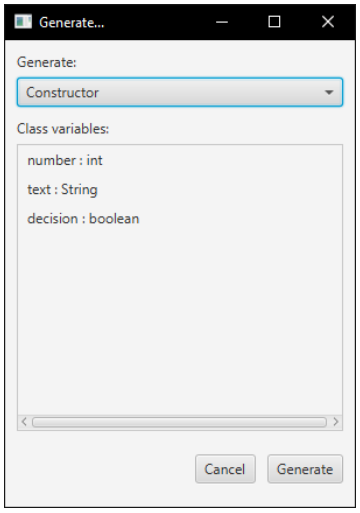
A ***Help*** menü csak egy menüpontot, a ***Documentation***-t tartalmazza, amire kattintva megnyílik a dokumentáció.

A kódolási felület

A kódolási felületen jelenik meg az éppen szerkeszthető fájl. Java fájl esetében a felület, a Java programnyelv szintaktikájának megfelelően kiszínezi a szöveget, kezeli a tabulálást és mutatja a sorok számát. Több fájl megnyitása esetén ezek között a kódolási felület tetején elhelyezkedő fülek segítségével váltogathatunk. Ha a füleket az egér segítségével behúzzuk a kódolási felületre, a felület kettéoszlik, és egyszerre két fájl is látható és szerkeszthető lesz (5. ábra).

5. ábra: párhuzamosan megnyitott két Java fájl, melyek be vannak színezve szintaktikai szabályoknak megfelelően

Ha Java fájlt szerkesztünk, akkor jobb kattintásra megjelenik egy menü, amiben szerepel a ***Generate…*** elem. Erre kattintva megjelenik egy ablak, ahol kiválaszthatjuk mit szeretnénk generáltatni a programmal, valamint, hogy milyen változókat szeretnénk használni a generáláshoz (6. ábra). A program képes konstruktort, *gettert*, *settert*, és egyszerre *gettert* és *settert* generálni az általa felismert osztályszintű változókból, a felhasználó által kiválasztottak alapján. A ***Generate*** gombra kattintva a kívánt elemek generálása megtörténik, és az első osztály szintű metódus fölé lesznek beillesztve. Ha az osztálynak még nincs osztály szintű metódusa, akkor a kurzor pozíciójában lesznek beillesztve.



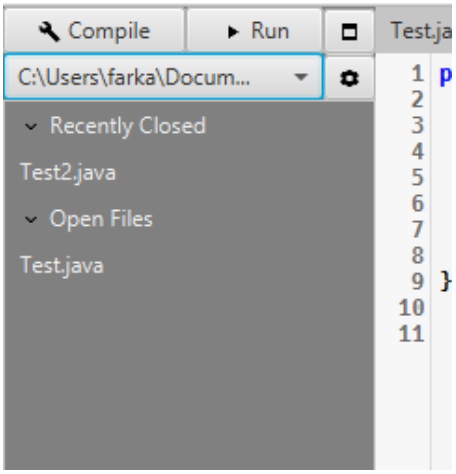
*6. ábra: a felugró **Generate…** ablak*

Az oldalsó menü

Az oldalsó menüben találhatók a futtatással, fordítással, valamint a projektkezeléssel kapcsolatos legfontosabb funkciók. Az itt található elemek viselkedése attól függ, hogy van-e megnyitva egy projekt, vagy nincs. A funkciók közti különbségeket az alábbi két részletben tárgyalom.

Projekten kívüli fájlok kezelése

Ha nincs éppen projekt megnyitva, akkor az *Open File* menü segítségével megnyitott, és a *New File* menüvel létrehozott fájlokat szerkeszthetjük, futtathatjuk. Az oldalsó menüben ekkor két lenyíló fül található: az ***Open Files***, ami az éppen megnyitott fájlokat tartalmazza, és a ***Recently Closed***, ami a nemrég bezártakat, hogy újra meg lehessen őket nyitni (7. ábra).



*7. ábra: az **oldalsó menü** kinézete abban az esetben, ha nincs megnyitva projekt*

- A ***Compile*** gomb megnyomásával az éppen kiválasztott *main* fájl kerül lefordításra. A fordítás eredménye egy külön ablakban fog megjelenni. A létrehozott *class* fájlokat a Java fájljaikkal megegyező mappába helyezi. Ha nincs kiválasztva *main* fájl, akkor ki van szürkítve, nem kattintható.

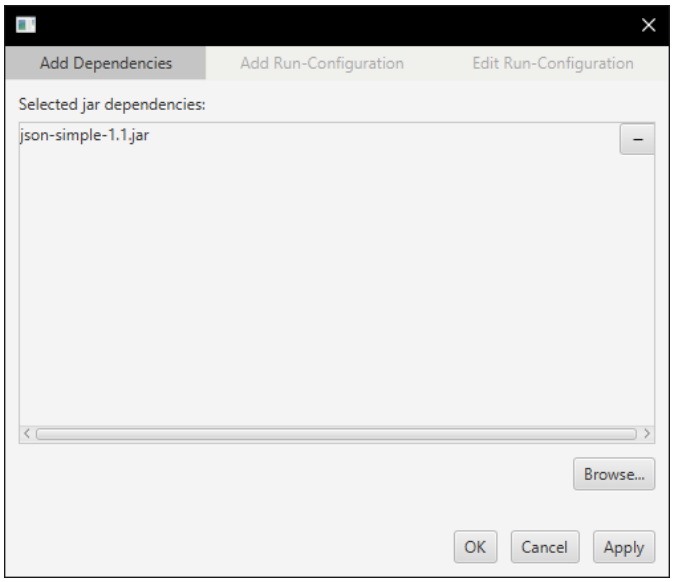
- A ***Run*** gomb megnyomásával a kiválasztott *main* fájl lefordul, és lefut. A lefutás eredménye egy külön ablakban fog megjelenni. Ha nincs kiválasztva *main* fájl, akkor ki van szürkítve, nem kattintható.

- A *Run* gomb melletti **ablak ikonra** kattintva megnyílik a **Windows Parancssor**. Ha van kiválasztott *main* fájl, akkor annak a mappájában nyílik meg, ha nincs akkor a felhasználó *home* mappájában.

- A *Run* és a *Compile* gomb alatt található a ***main* fájl választó leugró menü**. Ebben a menüben jelennek meg azok a fájlok, amik éppen meg vannak nyitva, és a program *main* fájlként észlelte őket. Az itt éppen kiválasztott *main* fájl az, amit lefordíthatunk, futtathatunk és állíthatunk be hozzá függőségeket.

- A *main* fájl választó menü melletti **fogaskerék ikonra** kattintva megnyílik egy

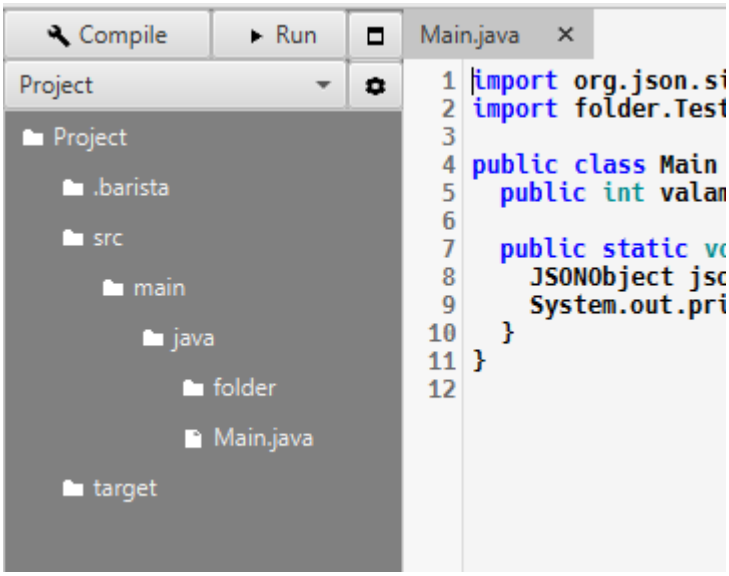
felugró ablak, ahol az éppen kiválasztható *main* fájlnak állíthatunk be függőségeket (8. ábra). A felső menüben található opciók közül az *Add Run-Configuration* és az *Edit Run-Configuration* ki van szürkítve, nem kattintható. Csak az éppen aktív ***Add Dependencies*** menüpont elérhető. Ebben a menüpontban láthatjuk az éppen kiválasztott *main* fájlhoz beállított függőségeket. A ***Browse…*** gombra kattintva megnyílik a **Windows Fájlkezelő**. Ekkor a felhasználó kiválaszthat egy tetszőleges helyen tárolt *jar* fájlt, ami aztán hozzá lesz adva a függőségekhez. Az ablak alján található **OK** gombbal elfogadjuk a változtatásokat és bezárjuk az ablakot, az ***Apply*** gombbal elfogadjuk a változásokat, de az ablak nem kerül bezárásra, a ***Cancel*** pedig gombbal bezárjuk az ablakot és elvetjük a változtatásokat.



*8. ábra: a felugró **Settings** ablak*

Projektek kezelése

Ha a *Load Project* vagy a *New Project* menüvel megnyitottunk, illetve létrehoztunk egy projektet, akkor ezt fordíthatjuk, futtathatjuk, illetve egyéb futtatási beállításokat hozhatunk létre az oldalsó menü segítségével. Az oldalsó menüben ekkor megjelennek az éppen megnyitott projekt mappái és fájljai, amikben lehet böngészni (9. ábra).



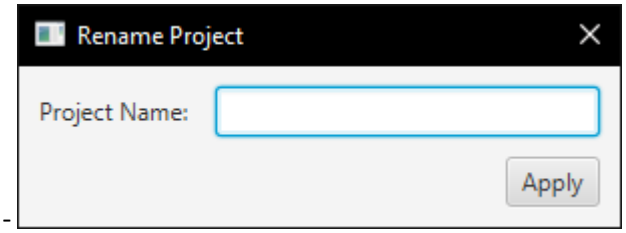
*9.ába: az **oldalsó menü** kinézete, ha van megnyitva projekt*

Az oldalsó menüben megjelenő fájlok és mappák közül vannak, amit a projekt létrehozásánál a program generált, és kiemelt jelentőséggel rendelkeznek. Egy projekt létrehozásánál a projekt **gyökérmappájának** a neve megegyezik a projektével. Az oldalsó menüben ez a legfelső mappa, amiben minden elhelyezkedik. Ebben a mappában található a program által generált ***.barista*** mappa. Ebben helyezkednek el a projekt tulajdonságait szamon tartó ***ProjektConfig.json*** és ***RunConfig.json*** fájlok**.** Az IDE zökkenőmentes működése érdekében ezeket a fájlokat **nem ajánlott megnyitni és szerkeszteni**! A gyökérmappa alatt található még a ***target*** mappa is. A fordítással keletkezett *class* fájlok itt lesznek megtalálhatók. Az utolsó mappa ezen a szinten az ***src***, ami a projekt forrásfájljait hivatott tárolni. Ezen belül található a ***main*** mappa, ami tartalmazza a ***java*** mappát. A *java* mappa a projekt **forrásmappája**, tehát a fordítási folyamatnak ez a mappa a kiindulási pontja. Ebbe a mappába kell helyezni a java forrásfájlokat. A program alapjáraton generál egy ***Main.java*** fájlt, ami alapértelemezetten a projekt *main* fájljának lesz beállítva.

A fájlok és mappák szabadon mozgathatók a projekten belül *drag-and-drop* módszerrel, valamint, ha a jobb egérgombbal kattintunk, megjelenik egy helyi menü. Attól függően, hogy fájlra, mappára vagy a projekt gyökérmappájára kattintunk, más-más lehetőségek lesznek elérhetők.

A **gyökérkönyvtárra** való jobbklikk a következő lehetőségeket hozza elő:

- A ***Rename Project*** menü megjelenít egy felugró ablakot, ahol egy új név megadásával átnevezhetjük a projektünket (10. ábra).



*10. ábra: a felugró **Rename Project** ablak*

- A ***Close Project*** menü megnyomásával a projekt bezárható

- A ***Delete Project*** menü megnyomásával a projektet törölni lehet, miután azt egy felugró ablakon keresztül megerősítettük

A következő menüpontok mind a **gyökérkönyvtárra** mind a más **mappákra** történő jobbklikknél láthatók lesznek:

- A ***New File*** menü megjelenít egy felugró ablakot, ahol megadhatjuk a létrehozandó fájl nevét és kiterjesztését (11. ábra). A kiterjesztést a legördülő menüből kell kiválasztani, a nevet pedig kiterjesztés nélkül megadni. Az ***other*** kiválasztásával tetszőleges kiterjesztésű fájlokat hozhatunk létre. Ekkor a fájl nevét már a kiterjesztéssel együtt kell megadni. Ha a ***.java*** kiterjesztést választjuk ki, akkor lehetőségünk van a fájlt egy sablon alapján generálni a következők közül: ***Class, Enum, Interface, Annotation, Record.*** A ***Create*** gombra kattintva a fájl létrejön abban a mappában, ahonnan a menüt megnyitottuk, és meg is nyílik a kódolási felületen.

*11. ábra: a felugró **Create New File** ablak*

-

- A ***New Folder*** megjelenít egy felugró ablakot, ahol megadhatjuk a létrehozandó mappa nevét (12. ábra). A ***Create*** gombra kattintva a mappa létrejön abban a

mappában, ahonnan a menüt megnyitottuk.

- 

*12. ábra: a felugró **New Folder** ablak*

Mappákra való jobbklikk (a fentebb tárgyaltakkal együtt) a következő lehetőségeket hozza elő:

- A ***Rename*** menü megjelenít egy felugró ablakot, ahol az új név megadásával átnevezhetjük a mappánkat (13. ábra). Ha ez a mappa egyben egy **java csomag** is, a benne lévő java fájlok automatikusan „átcsomagolódnak”. Tehát ha egy java fájl eddig a ***hu.farkasch.main.foo*** csomagban volt, és a ***foo*** mappának a ***bar*** nevet adjuk, akkor a fájl új csomagja a ***hu.farkasch.main.bar*** lesz.

- 

*13. ábra: a felugró **Rename Folder** ablak.*

- A ***Delete*** menü megnyomásával a **mappa és az összes benne lévő elem** törlésre kerül, miután azt a felhasználó egy felugró ablakon megerősítette.

Fájlokra való jobbklikk a következő lehetőségeket hozza elő:

- A ***Rename*** menü megjelenít egy felugró ablakot, ahol az új név megadásával átnevezhetjük a fájlunkat (14. ábra). Ha ez a fájl egy **java fájl**, akkor az **osztály neve és vele együtt az összes rá utaló referencia** is át lesz nevezve a projektben. Tehát ha egy fájl hivatkozik egy ***Foo*** objektumra a következőképpen:

Foo foo = new Foo();

Akkor, ha a ***Foo***-t átnevezzük ***Bar***-ra, ez a kódrészlet így fog kinézni:

Bar foo = new Bar();

*14.ábra: a felugró **Rename File** ablak*

- A ***Delete*** menü megnyomásával a fájl törlésre kerül, miután azt a felhasználó egy felugró ablakon keresztül megerősítette.

Ha meg van nyitva projekt, akkor az oldalsó menü tetején lévő gombok a következőképpen viselkednek:

- A ***Compile*** gomb megnyomásával a projektben található forrásfájlok lefordulnak. Ez a folyamat a **Projekt\src\main\java** mappából lesz elindítva. A fordítás eredménye egy külön ablakban fog megjelenni. A létrehozott *class* fájlok a **Projekt\target** mappában lesznek elhelyezve.

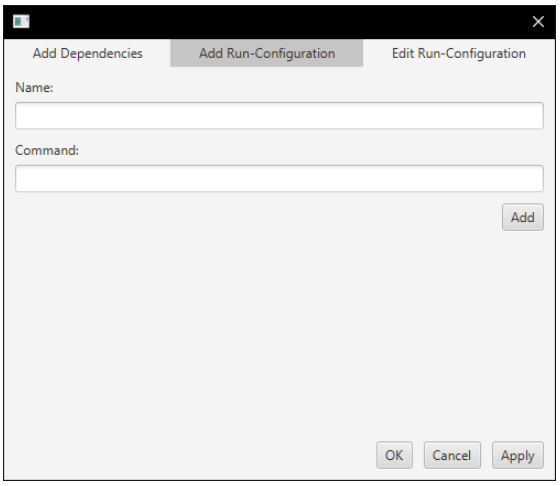
- A ***Run*** gomb megnyomásával a projekt lefordul, és a kiválasztott módon lefut. A lefutás eredménye egy külön ablakban fog megjelenni.

- A *Run* gomb melletti **ablak ikonra** kattintva megnyílik a **Windows Parancssor**. A parancssor a projekt **forrásmappájában** nyílik meg.

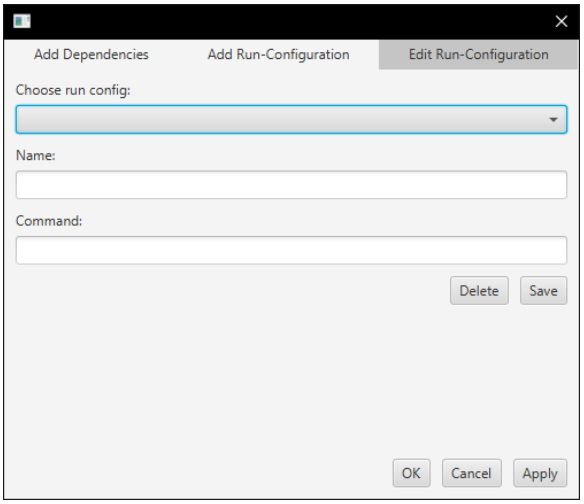
- A *Run* és a *Compile* gomb alatt található a **futási konfigurációk kiválasztására** szolgáló leugró menü. A menü alapjáraton tartalmaz egy futási konfigurációt, ami mindig a projekt eredeti nevét viseli. Ha a felhasználó specifikált másik futási konfigurációt, akkor azt ebből a menüből lehet kiválasztani.

- A futási konfiguráció választó menü melletti **fogaskerék ikonra** kattintva megnyílik egy felugró ablak, ahol az éppen megnyitott projektnek állíthatunk be függőségeket, valamint vehetünk fel új, és szerkeszthetünk meglévő futási konfigurációkat (15. ábra). Az ***Add Dependencies*** menüpontban láthatjuk az éppen megnyitott projekthez beállított függőségeket. A ***Browse…*** gombra kattintva megnyílik a **Windows Fájlkezelő**. Ekkor a felhasználó kiválaszthat egy tetszőleges helyen tárolt *jar* fájlt, ami aztán hozzá lesz adva a függőségekhez. Az ***Add Run-Configuration*** menüpontban vehetünk fel új futási konfigurációt (15. ábra). Ehhez meg kell adnunk a konfiguráció nevét, és azt a parancsot, ami lefut majd a *Run*

gombra kattintáskor. A ***Save*** gombra kattintva az új konfiguráció **előzetesen el lesz mentve** a meglévő konfigurációk közé. Az ***Edit Run-Configuration*** menüpontban tudjuk a meglévő futási konfigurációkat szerkeszteni (16. ábra). Ehhez a lenyíló menüből válasszuk ki a szerkeszteni kívánt konfigurációt, végezzük el a változtatásokat, majd nyomjuk meg a ***Save*** gombot, mely **előzetesen elmenti** a megtett változtatásokat. Ebben a menüben tudunk meglévő konfigurációkat törölni a ***Delete*** gombbal. Az ablak alján található **OK** gombbal elfogadjuk a változtatásokat és bezárjuk az ablakot, az ***Apply*** gombbal elfogadjuk a változásokat, de az ablak nem kerül bezárásra, a ***Cancel*** gombbal pedig bezárjuk az ablakot és elvetjük a változtatásokat.



*15. ábra: az **Add Run-Configuration** fül*



*16. ábra: az **Edit Run-Configuration** fül*

Fejlesztői dokumentáció

A feladat megoldása

A program jelentős része **Java**ban írtam, de a kódolási felület **HTML-**lel és **JavaScript**tel hoztam létre. A program így legfelső szinten két részre különül el: a kódolási felület JavaScriptes funkcionalitásaira, és a program egyéb funkcionalitását implementáló Java kódra. A kód Java részét ezen belül 4 részre lehet osztani: a felületet implementáló kódok, amelyek a **com.farkasch.barista.gui** csomagban helyezkednek el, a háttérlogikát implementáló kódok, amelyek a **com.farkasch.barista.services** csomagban helyezkednek el, a program segítőeszközeit implementáló kódok, amelyek a **com.farkasch.barista.util** csomagban helyezkednek el és végül a programot tesztelő kódok, amik egy külön **test** mappában helyezkednek el. Ezeknek a csomagoknak a tartalmát a dokumentáció során részletezni fogom.

A program megírása során ügyeltem a Java specifikus kódolási szabályok betartására, az objektum orientáció előnyeinek kihasználására. A program elkészítéséhez igyekeztem a lehető legnaprakészebb Java verziót (a szakdolgozat elkezdésekor ez a **Java 17.0.2**-es verziója volt) és segédkönyvtárakat használni. Az IDE megírásához a következő technológiákat, könyvtárakat használtam fel: **Java Spring**, **JavaFX**, JSON.simple[3], javax.annotation[4], Guava[5], **CodeMirror**. A fontosabbakat alább részletezem.

A projektem függőségeit **maven**[6] segítségével kezelem, illetve a program forráskódját a **GitHub**ra[7] is feltöltöttem.

Felhasznált technológiák

JavaFX

A **JavaFX** egy nyílt forráskódú **kliens-applikációs keretrendszer** a Java nyelvhez, amit az **Oracle** fejleszt, de 2018 óta része az **OpenJDK**nak is **OpenJFX** néven. Először 2008-ban jelent meg, és az elavultabb elődjeit, az **AWT**-t (1995) és a **Swing**et volt hivatott leváltani. Ennek ellenére (bár ma már a JavaFX is a legnépszerűbb Javas felület-fejlesztő keretrendszerek között van) ez nem teljesen sikerült, és a Swinggel „párhuzamosan” létezik. Fejlesztése az AWT-n alapul, így ennek a könyvtárnak az

elemei is használhatók egy JavaFX alkalmazásban.

A felület fejlesztésénél azért a JavaFX-re esett a választásom, mert egy modern kinézetű alkalmazást szerettem volna létrehozni. Ezt a JavaFX **CSS-sel való szerkeszthetősége** teszi lehetővé, továbbá vonzónak **találtam a modernebb, magasabb szintű, egyszerűbben használható API-**ját is. A JavaFX-nek van egy saját **XML alapú nyelve**, az **FXML.** Ez lehetővé teszi, hogy a felhasználói felület **felépítését az oldal logikájától függetlenül** végezhessük el. Ezzel az eszközzel nem éltem, hanem az oldal felépítését a Java forráskódban szabtam meg, mert ez a programozási stílusomhoz közelebb állt.

Alkalmazásomban az **org.openjfx javafx-controls** és **javafx-web** maven függőségek 18-as verzióit használtam.

A JavaFXről bővebb információ érhető el a JavaFX hivatalos oldalán [8].

Spring Framework és Spring Boot

A **Spring Framework**[9] Java alapú alkalmazások fejlesztését könnyíti meg sok hasznos funkcióval. Főleg MVC webalkalmazások fejlesztéséhez használják, de egyéb nem webes programok elkészítéséhez is kitűnően alkalmas.

Programomban főleg a Spring Framework által biztosított automatikus ***dependency injectiont*** alkalmaztam. Az ***@Autowired*** annotációval megjelölt **Spring komponenseket** a Spring automatikusan létrehozza és használhatóvá teszi, így azt a felhasználó osztálynak már valóban csak felhasználnia kell, és semmilyen inicializáló lépést nem kell csinálnia.

A **Spring Boot**[10] megkönnyíti a *stand-alone* Spring alapú alkalmazások fejlesztését. **Automatikusan konfigurálja az egyéb Springes** (és harmadik féltől származó) **keretrendszereket** (ebben az esetben a Spring Frameworköt), valamint maga a Spring alkalmazás is ennek a segítségével fut.

Alkalmazásomban az **org.springframework.boot spring-boot, spring-boot-autoconfigure** és **spring-boot-maven-plugin** maven függőségek 2.6.5-ös verzióit használtam.

CodeMirror

A **CodeMirror** egy JavaScript-ben megírt, nyílt forráskódú API, aminek segítségével kezelem a kódolási felületet. Mind alacsony, mind magas szintű hívásokat támogat, így a kódolási felület testre szabása egyedül a programozótól függ.

Kódomban a **CodeMirror beépített szintaxis-felismerőjét** használtam, az általam definiált stílussal. Továbbá a **CodeMirror API-jával** valósítottam meg a **kódgenerálást** és más, a kódot változtató műveleteket. Alkalmazásom a **CodeMirror 5.65.3-as verzióját** használja.

A CodeMirrorról bővebb információ érhető el a CodeMirror hivatalos oldalán [11].

Programozási, fejlesztési elv

A fejlesztés során az objektumorientáltság adta a legfőbb irányvonalat, miszerint a program fontosabb, összetartozó részeit **objektumokba** foglaljuk. A Java nyelv ebbe szinte belekényszeríti a programozót, hisz a szintaxisa megköveteli az **osztályok** létrehozását.

Az objektumok csomagolása a felhasználásuk alapján történt első sorban. A **gui** csomagba kerültek a felülettel kapcsolatos osztályok. Ezek az objektumok nagyrészt a **JavaFX könyvtár osztályaiból származnak le**, metódusaiknál és változóiknál törekedtem arra, hogy **csak a felület kinézetére, szerkezetére legyenek hatással**. A **services** csomagban vannak az alkalmazás ***backendjével*** foglalkozó fájlok. Ezek a fájlok végzik el a fontos műveleteket, itt történnek meg a háttérfolyamatok. A **test** mappában helyezkednek el az alkalmazás tesztjei. Ezek nagyrészt a services csomag **Unit-tesztjeiből** állnak. A **util** csomag tartalmazza a különböző általam definiált **segédosztályokat és enumokat**. Fontos még a **resources** mappa is, ami a **kódolási felület logikáját**, és a rajta felhasznált betűstílust tartalmazza.

A Spring keretrendszer által biztosított ***dependency injection*** biztosítása érdekében a Spring számára „láthatóvá kell tenni” a létrehozott osztályokat. Ezt úgy tehetjük, hogy osztályainkat a Spring olyan annotációi egyikével jelöljük meg, amik **Spring *beanként*** jelölik meg őket.

Mielőtt folytatom, fontos tisztázni, hogy mi az a ***bean*** a Javaban és a Springben. Egy *bean* egy olyan objektum, ami sok másik objektumot tartalmaz, a programban több

helyről elérhető ezáltal könnyebb karbantartást eredményez. Egyszerűbben kifejezve: egy *bean* egy „újrahasznosítható szoftver-komponens”. A Springben többféle *beant* különböztetünk meg. A program szempontjából két típus az, ami fontos: a ***singleton*** és a ***prototype***. A ***singleton*** az alapértelmezett *bean* típus. *Singleton beanek* **módosítható** és ***„stateful”*** azaz „állapottal rendelkező” objektumok. A *bean* az **egész Spring rendszerben elérhető**, és ha változtatunk rajta valamit, akkor az a program egy másik pontjában is tükröződni fog. Egy *singleton beanből* csak **egyetlen egy létezhet egy osztályon belül**, valamint **nem lehet őket a *new*** kulcsszóval inicializálni (ez az összes Spring *bean* típusra igaz). A ***prototype*** *bean* szintén módosítható viszont „***stateless***” azaz „állapottal nem rendelkező” objektum. A *singleton beannel* ellentétben lehet őket **többször is egy osztályban inicializálni**, valamint minden alkalommal, amikor lesz egy ilyen *bean* kérve a Springtől egy **új objektum lesz létrehozva**.

Ha egy osztályból Spring *beant* szeretnénk csinálni meg kell jelölni egy olyan annotációval, ami Springnek jelzi, hogy az osztály egy *bean.* A programban a ***@Service*** annotációval illettem a **services** csomagban lévő osztályokat és ***@Component*** annotációval az **egyéb** osztályokat, amelyeket a Spring *beanként* kell, hogy felismerjen.

A program felépítése – a services csomag osztályai

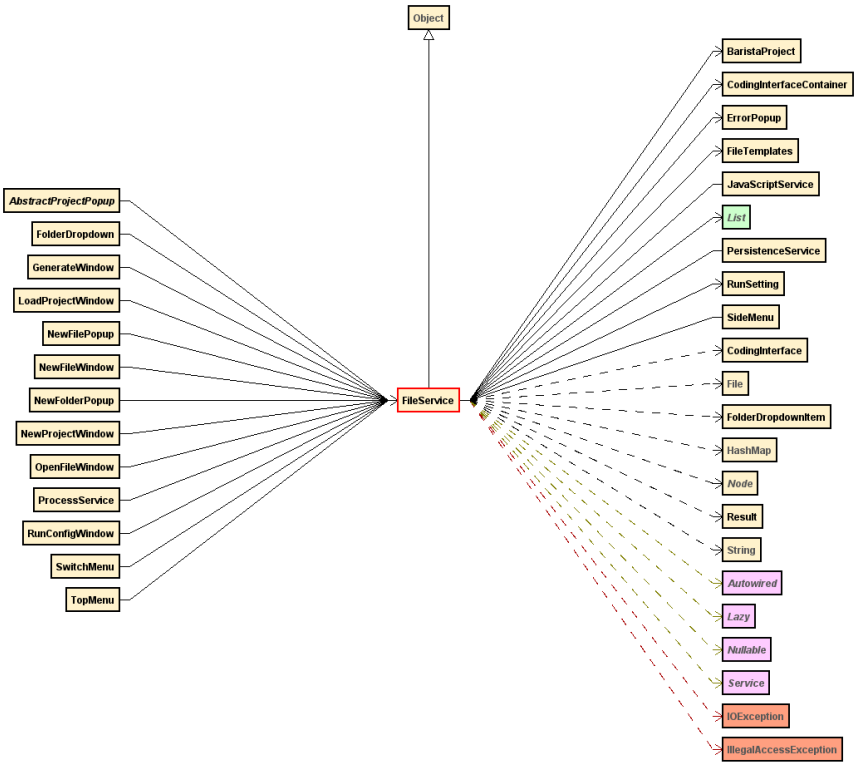
A FileService osztály

A **FileService** osztály kezeli **a mindenféle fájllal kapcsolatos műveleteket**, többek között ezek létrehozását, törlését és szerkesztését. Alább látható az osztály **UML diagramja,** valamint az **osztály relációs diagramja** (17.ábra).

Object
«@Service» com::farkasch::barista::services:: **FileService**
Fields «@Lazy» «@Autowired» – codingInterfaceContainer : CodingInterfaceContainer «@Lazy» «@Autowired» – errorPopup : ErrorPopup «@Autowired»

– fileTemplates : FileTemplates «@Lazy» «@Autowired» – javaScriptService : JavaScriptService «@Lazy» «@Autowired» – persistenceService : PersistenceService «@Lazy» «@Autowired» – sideMenu : SideMenu
Properties «readOnly» + projects : List<BaristaProject> + runConfig : List<RunSetting>
Constructors + FileService( ) : void
Methods + addNewJarConfig(String, List<String>) : void + cleanupJarJson( ) : void + createErrorLog(String) : File + createFile(String) : Result + createFile(String, FolderDropdownItem) : Result – createFolder(String) : Result + createFolder(String, FolderDropdownItem) : Result + createNewProject(BaristaProject) : Result + deleteFile(File, boolean) : boolean + deleteFolder(FolderDropdownItem) : boolean + deleteProject(BaristaProject) : void + folderContains(String, String) : boolean + getClassLevelVariables(String) : List<String> + getDirsAndFiles(String) : List<File> + getGenerateInsertPosition(String, CodingInterface) : int + getJarsForFile(String) : List<String> «synthetic» – lambda$deleteFile$0(File, Node) : boolean «synthetic» – lambda$deleteFolder$1(File, String) : boolean «synthetic» – lambda$deleteFolder$2(File, String) : boolean «synthetic» – lambda$deleteFolder$3(File, String) : boolean «synthetic» – lambda$deleteFolder$4(File, Node) : boolean «synthetic» – lambda$deleteFolder$5(Node) : void «synthetic» – lambda$getClassLevelVariables$15(String) : boolean «synthetic» – lambda$getDirsAndFiles$6(File) : boolean «synthetic» – lambda$getDirsAndFiles$7(File) : boolean «synthetic» – lambda$renameFolder$10(File, File, String) : String «synthetic»

- lambda$renameFolder$11(File, File, Node) : void
  «synthetic»
- lambda$renameFolder$12(File, File, FolderDropdownItem) : void
  «synthetic»
- lambda$renameFolder$13(File, String) : boolean
  «synthetic»
- lambda$renameFolder$14(File, File, String) : void
  «synthetic»
- lambda$renameFolder$8(File, File, String) : String
  «synthetic»
- lambda$renameFolder$9(File, File, String) : String
+ loadProject(BaristaProject) : void
+ moveFile(File, String) : Result
+ moveFolder(File, File) : Result
+ prepareForTesting(HashMap<String, Object>) : void
- recursiveMove(File, File) : void
- redoImports(File, File) : void
- renameFile(File, String) : Result
+ renameFile(File, String, FolderDropdownItem) : Result
+ renameFolder(File, String, FolderDropdownItem) : Result
+ renameProject(String, FolderDropdownItem) : void
- renameReferences(File, String, String) : void
- repackage(File, File, File) : void
+ saveFile(File, String) : void
+ saveProject( ) : void
+ updateJarsInJarConfig(String, List<String>) : void
+ updateNameInJarConfig(String, String) : void



*17. ábra: A **FileService** osztály relációs diagramja*

public void saveFile(File file, String content): a megadott fájl tartalmát, a megadott String tartalmával felülírom, ezzel mentve a megtett változtatásokat a fájlon. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public Result createFile(String path): létrehozok a megadott elérési útvonallal egy fájlt. Ha a fájl létrehozása sikeres, akkor **Result.OK**-vel térítek vissza, különben **Resul.FAIL**-lel. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public Result createFile(String path, FolderDropdownItem creationFolder): létrehozok a megadott elérési útvonallal egy fájlt, majd hozzáadom az éppen megnyitott projekthez. Ha a fájl létrehozása sikeres, akkor **Result.OK**-vel térítek vissza, különben **Resul.FAIL**-lel. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

private Result createFolder(String path): létrehozok a megadott elérési útvonallal egy mappát. Ha a mappa létrehozása sikeres, akkor **Result.OK**-vel térítek vissza, különben **Resul.FAIL**-lel.

public Result createFolder(String path, FolderDropdonwItem creationFolder): létrehozok a megadott elérési útvonallal egy mappát, majd hozzáadom a projekthez. Ha a mappa létrehozása sikeres, akkor **Result.OK**-vel térítek vissza, különben **Resul.FAIL**-lel.

public boolean deleteFile(File file, boolean partOfProject): törlöm a megadott fájlt, és visszatérítek ennek eredményével. Ha a megadott File nem egy fájl, akkor **false** értékkel térítek vissza. Ha a fájl része volt egy projektnek, akkor el lesz távolítva belőle. Ha a fájl meg volt nyitva a kódolási felületen, akkor onnan is el lesz távolítva.

public boolean deleteFolder(FolderDropdownItem folderDropdownItem): eltávolítom a megadott mappát a projektből, valamint törlöm a mappát és a teljes tartalmát. A visszatérési érték a törlés eredménye.

public void addNewJarConfig(String fileName, String… jars): a megadott *main* fájlhoz beállítom a megadott *jar* függőségeket. Ezt úgy teszem, hogy megnyitom az

…/Appdata/Roaming/BaristaIDE/config/JarConfig.json fájlt, ahhoz hozzáadom a megadott *main* fájlt, és az ehhez tartozó *jar* függőségeket a megadott *jarok* alapján. Ha a metódusban felmerül **IOException** vagy **ParseException,** akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public void updateNameInJarConfig(String oldFileName, String fileName): megváltoztatom a megadott *main* fájl nevét a **…/Appdata/Roaming/BaristaIDE/config/JarConfig.json** fájlban a metódusban megadott új névre. Ha a metódusban felmerül **IOException** vagy **ParseException,** akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public void updateJarsInJarConfig(String fileName, List<String> jars): megkeresem az **…/Appdata/Roaming/BaristaIDE/config/JarConfig.json** fájlban a keresett *main* fájlt és felülírom a hozzá tartozó *jar* függőségeket. Ha a metódusban felmerül **IOException** vagy **ParseException,** akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public List<String> getJarsForFile(String fileName): megkeresem az **…/Appdata/Roaming/BaristaIDE/config/JarConfig.json** fájlban a keresett *main* fájlt és visszatérítek a fájlhoz tartozó *jar* függőségek listájával. Ha a metódusban felmerül **IOException** vagy **ParseException,** akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public Result createNewProject(BaristaProject baristaProject): létrehozom a megadott BaristaProject alapján az új projektet. Létrehozom a szükséges mappákat, fájlokat és frissítem a **…/Appdata/Roaming/BaristaIDE/config/ProjectConfig.json** fájlt. A sikeres létrehozás után a **Result.OK**-val térítek vissza és megnyitom a projektet a **loadProject** meghívásával. Ha valami a létrehozásnál nem volt megfelelő, akkor **Result.FAIL**-lel térítek vissza. Ha a metódusban felmerül **IOException** vagy **ParseException,** akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public void loadProject(BaristaProject baristaProject): a megadott BaristaProjectet

felöltöm a szükséges adatokkal, majd meghívom a **SideMenu openProject**, és a **PersistenceService setOpenProject** metódusát, így megnyitva a projektet a felületen és beállítva mint éppen megnyitott projektet. Ha a metódusban felmerül **FileNotFoundException** vagy **ParseException**, akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public void saveProject(): az éppen megnyitott projektet elmentem, úgy hogy felülírom a projektben található **ProjectConfig.json** értékeit az éppen megnyitott BaristaProject értékeivel. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt

public void deleteProject(BaristaProject baristaProject): törlöm a megadott projektet, úgy, hogy eltávolítom a projektet az **…/Appdata/Roaming/BaristaIDE/config/ProjectConfig.jsonból** majd kitörlöm a fájl gyökérmappáját és teljes tartalmát. Ha a projekt meg volt nyitva, akkor ez előtt bezárom a **SideMenu closeProject** metódusának meghívásával. Ha a metódusban felmerül **IOException** vagy **ParseException,** akkor ezeket elkapja, és visszajelez a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public void renameProject(String name, FolderDropdownItem folderDropdownItem): megváltoztatom az éppen megnyitott projekt nevét a megadott új névre. Ennek érdekében meghívom a **renameFolder** metódust a **folderDropdownItem**-ra ami az oldalsó menüben megnyitott projekt legfelső mappája, és mentem a projektet. Ezek után megváltoztatom a projekt nevét a **…/Appdata/Roaming/BaristaIDE/config/ProjectConfig.json-ban** is. Ha a metódusban felmerül **IOException** vagy **ParseException,** akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public List<BaristaProject> getProjects(): kiolvasom a **…/Appdata/Roaming/BaristaIDE/config/ProjectConfig.json-ban** található projektek neveit és gyökérmappáit, és egy ezeket tartalmazó listával térítek vissza. A visszatérített BaristaProject-ek **csak a nevet és a gyökérmappát tartalmazzák!** Ez a metódus csak felsorolási jelleggel, a teljesség igénye nélkül adja vissza a projekteket. Ha a metódusban felmerül **IOException** vagy **ParseException,** akkor

ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public List<RunSetting> getRunConfig(): az éppen betöltött projekt futási konfigurációit adom vissza. Ezeket a projekt **RunConfig.json** fájljából töltöm be. Az alapértelmezett futási konfiguráció ebben nem szerepel, így azt itt hozom létre, és adom hozzá. Ha a metódusban felmerül **IOException** vagy **ParseException,** akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public void setRunConfig(List<RunSettings> runSettings): a megadott futási konfigurációkkal felülírom az éppen betöltött projekt **RunConfig.json** fájljában található adatokat. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public List<File> getDirsAndFiles(String fodlerPath): visszaadom a megadott mappában található nem elrejtett fájlokat és mappákat. Ha a **folderPath null**, akkor a **felhasználó *homejának*** a mappáival és fájljaival térítek vissza. Ha a megadott útvonal egy fájlra mutat, vagy a mappa üres, akkor egy üres listát adok vissza.

public Result renameFile(File file, String name, FolderDropdownItem folderDropdownitem): átnevezem a megadott fájlt. Ezt a rendszerben, a felületen és a projektben is megteszem. Először meghívom a privát **renameFile** metódust, majd az azáltal visszatérített új fájllal dolgozok tovább. A **folderDropdownItem null,** akkor a **SideMenu openFiles** és **recentlyClosed** értékeit frissítem az új fájllal, ha **nem null,** akkor kicserélem a régi fájlt az új fájlra a projektben, illetve meghívom a privát **renameReferences** metódust. Ha a fájl átnevezése sikeres, akkor **Result.OK**-vel térítek vissza, különben **Resul.FAIL**-lel.

private Result renameFile(File file, String name): átnevezem a megadott fájlt. Ezután a megnyitott **CodingInterfacekben** átnevezem a régi fájlt az újra, ha meg volt nyitva. Ha a fájl átnevezése sikeres, akkor **Result.OK**-vel térítek vissza, különben **Resul.FAIL**-lel. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public Result renameFolder(File oldFolder, String name FolderDropdownItem

folderDropdownItem): átnevezem a megadott mappát. Ezt a rendszerben, a felületen és a projektben is megteszem. Ha a mappa egy java csomag, akkor a benne lévő fájlokat „átcsomagolom” a **repackage** meghívásával, illetve kijavítom a rá vonatkozó importokat a **redoImports** meghívásával. Ha a mappa „különleges” tulajdonságokkal rendelkezik, azt is beállítom a **ProjectConfig.json**-ban. Ha a mappa átnevezése sikeres, akkor **Result.OK**-vel térítek vissza, különben **Resul.FAIL**-lel.

public Result moveFile(File fileToMove, String destinationFolder): áthelyezem a kiválasztott fájlt a megadott mappába. Ezt a rendszerben, a felületen és a projektben is megteszem. Ha már van ilyen nevű fájl a mappában, akkor **Result.FAIL**-lel térítek vissza. Az áthelyezés után a fájlt „átcsomagolom” a **repackage** meghívásával, illetve kijavítom a rá vonatkozó importokat a **redoImports** meghívásával. Ha a fájl áthelyezése sikeres, akkor **Result.OK**-vel térítek vissza, különben **Resul.FAIL**-lel. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public Result moveFolder(File directoryToMove, File targetDirectory): áthelyezem a mappát és az egész tartalmát a célhelyre. Ezt a **recursiveMove** meghívásával teszem. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public List<String> getClassLevelVariables(String content): visszatérítek a megadott kódrészlet osztályszintű változóival. A változókat úgy szűröm ki, hogy végig iterálok a String karakterein, és csak akkor adom őket hozzá egy átmeneti Stringhez, ha pontosan 1 nyitó kapcsos zárójel van előttük. Az így létrehozott Stringet végül reguláris kifejezés (regular expression, regex) segítségével formázom, és Listába szedem.

public int getGeneratePosition(String content, CodingInterface codingInterface): megkeresem azt a sorszámot, ahova a generált kódrészletet be kell szúrni. Ezt úgy teszem, hogy a megadott kódrészletben regex segítségével megkeresem az első metódust, és ennek a sorszámát adom vissza. Ha nincs metódus a kódrészletben a kurzor jelenlegi pozícióját adom vissza, amit a **JavascriptService getCursorLine** metódusával kapok meg.

private void recursiveMove(File item, File targetDirectory): rekurzívan bejárok egy mappát és a benne lévő elemeket (az előző mappa hierarchiájának megtartásával) egy új helyre másolom, és ennek megfelelően szerkesztem az éppen megnyitott projektet.

private void renameReferences(File file, String oldRefernce, String new Reference): a kért fájlban megkeresem, és kicserélem a megadott referenciákat. Ha a fájl épp meg van nyitva a kódolási felületen, akkor ott is elvégzem a változtatásokat. Ha a metódusban felmerül **FileNotFoundException** akkor ezt elkapja, és visszajelez a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

private void repackage(File fileToMove, File destinationFile, File currentFile): „átcsomagolom" a megadott fájlt. A **fileToMove** a régi csomaggal, a **destinationFile** az új csomaggal ellátott fájlra mutat, és a **currentFile** pedig az a fájl, amit éppen változtatni szeretnék. Ha a **currentFile** null, akkor a **fileToMove**-ot veszem **currentFile**-nak. Ha az éppen módosított fájl meg van nyitva a kódolási felületen, akkor ott is megtörténnek a változtatások. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

private void redoImports(File fileToMove, File destinationFile): megkeresem azokat a fájlokat, ahol a régi osztály importálva volt, és átcserélem őket az új, helyes importokra. Ha az éppen módosított fájl meg van nyitva a kódolási felületen, akkor ott is megtörténnek a változtatások. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

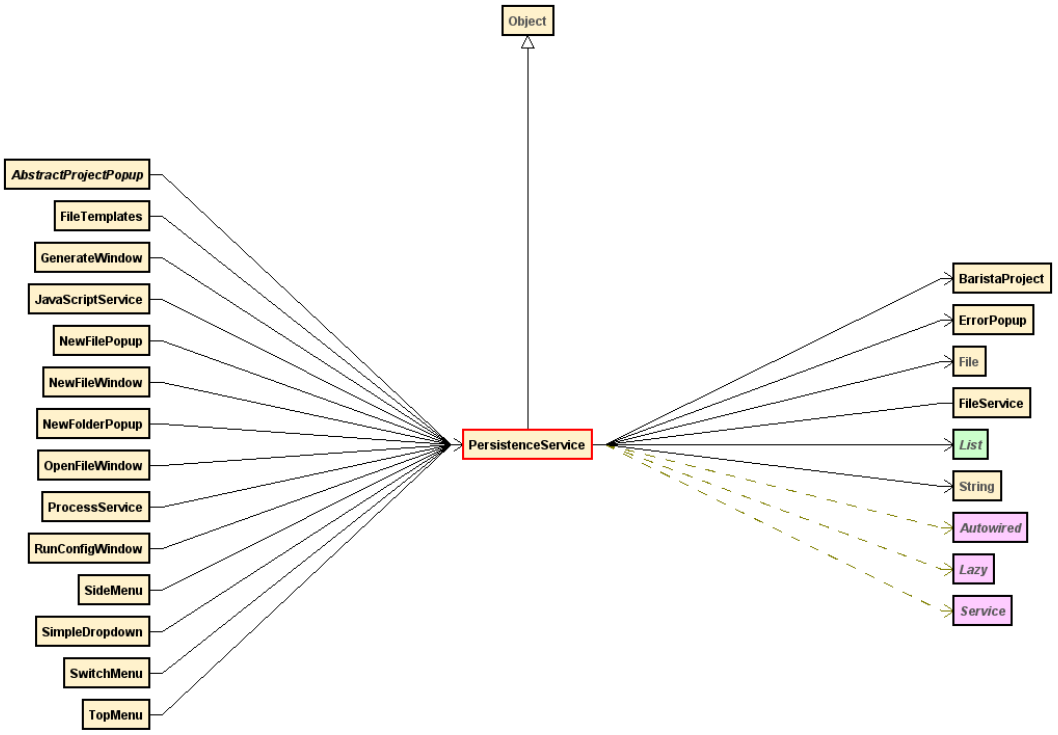
public File createErrorLog(String content): létrehozok egy szöveges fájlt **error_log_[mai dátum]** néven, az **…\AppData\Roaming\BaristaIDE\logs** mappában. Tartalma a megadott szöveg lesz.

public boolean folderContains(String folderPath, String itemPath): megvizsgálom, hogy a megadott elem szerepel-e a megadott mappában, és ezzel térítek vissza.

A PersistenceService osztály

Ez az osztály kezeli az **éppen megnyitott projektekhez, fájlokhoz tartozó és egyéb adatokat**, amikhez **más objektumoknak feltétlenül szüksége** van, mint pl. az éppen megnyitott projekt. Alább látható a **PersistenceService UML diagramja** és **relációs diagramja** (18. ábra).

Object «@Service» com::farkasch::barista::services:: **PersistenceService**
Fields «@Lazy» «@Autowired» – errorPopup : ErrorPopup «@Lazy» «@Autowired» – fileService : FileService
Properties + activeFile : File «readOnly» + contentToOpen : String «readOnly» + contentToSwitch : String + fileToOpen : File + fileToSwitch : File + generateInsertPosition : int + generatedContent : String + mainFiles : List<File> + openFiles : List<File> + openProject : BaristaProject + previouslyActiveFile : File + recentlyClosed : List<File>
Constructors + PersistenceService( ) : void
Methods + addMainFile(File) : void + addOpenFile(File) : void + addRecentlyClosed(File) : void + removeMainFile(File) : void + removeOpenFile(File) : void + removeRecentlyClosed(File) : void + updateAndGetCurrentMainFiles( ) : List<File>



*18. ábra: a **PersistenceService** relációs diagramja*

A **PersisitenceService** változói fontos adatokat tárolnak a program szempontjából. Ezek mind privát változók *getter* és *setter* metódusokkal.

- **activeFile** : **File** -> a legutoljára szerkesztett fájl, az a fájl, ahol éppen a kurzor tartózkodik.

- **previouslyActiveFile : File** -> az a fájl, amit az activeFile előtt lett szerkesztve.

- **fileToOpen** : **File** -> fájl, aminek az adatait át kell adni a JavaScript által kezelt kódfelületnek. Ezt akkor használom, ha a fájlt **kattintással** nyitották meg.

- **fileToSwitch : File** -> fájl, aminek az adatait át kell adni a JavaScript által kezelt kódfelületnek. Ezt akkor használom, ha a fájlt ***drag-and-droppal*** nyitották meg.

- **openFiles : List<File>** -> az éppen megnyitott fájlok, ha nincs aktív projekt

- **mainFiles : List<File>** -> az éppen megnyitott *main* fájlok, ha nincs aktív projekt

- **recentlyClosed : List<File>** -> a nemrég bezárt fájlok, ha nincs aktív projekt

- **openProject : BaristaProject** -> az éppen megnyitott projekt

- **generatedContent : String** -> a program által generált kód, amit a JavaScript innen ér el

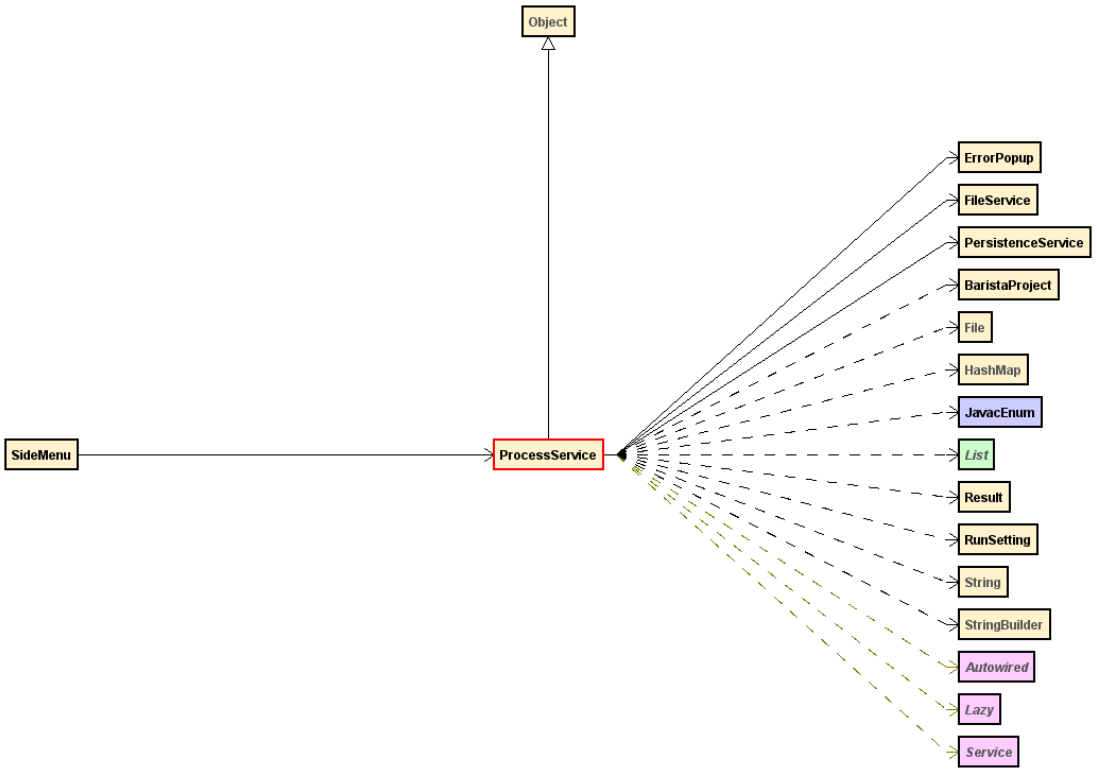
- **generateInsertPosition : int** -> annak a sornak a száma, ahová a generált kódot be kell szúrni.

A PersistenceService egyetlen igazán fontos metódusa az **updateAndGetCurrentMainFiles()**, ami végig pásztázza az éppen megnyitott fájlokat és felismeri azokat, amelyek tartalmazzák a „**public static void main(String[] args)**” kódrészletet. Ezeket a fájlokat *main* fájlként fogja felismerni, és ezekkel frissíti a **mainFiles** változót.

A ProcessService osztály

A **ProcessService** felelős az IDEn kívüli programok elindítására. Ez az osztály végzi a fordítást, futtatást, illetve a Windows Parancssor megnyitását. Alább látható a ProcessService **UML diagramja** és **relációs diagramja** (19. ábra).

Object
«@Service» com::farkasch::barista::services:: **ProcessService**
Fields «@Lazy» «@Autowired» – errorPopup : ErrorPopup «@Lazy» «@Autowired» – fileService : FileService «@Lazy» «@Autowired» – persistenceService : PersistenceService
Constructors + ProcessService( ) : void
Methods – Compile(String, List<String>, HashMap<JavacEnum, Object>) : Result + CompileFile(String, String) : Result + CompileProject(BaristaProject) : Result – Run(File, String, String, RunSetting) : void + RunFile(String, String) : void + RunProject(RunSetting) : void – createArgumentFile(String, HashMap<JavacEnum, Object>) : File – createSourceFile(String, List<String>) : File «synthetic» – lambda$Compile$0(StringBuilder, String) : void + openCommandPrompt(File) : void



*19. ábra: a **ProcessService** relációs diagramja*

public Result CompileFile(String filePath, String fileName): a megadott fájl útvonala és neve alapján lefordítom azt. Lekérem a fájlhoz tartozó jar függőségeket a **FileService getJarsForFile** metódusával, majd meghívom a **Compile** metódust. Ha a fordítás sikeres volt, akkor **Result.OK**-val térítek vissza, egyébként **Result.FAIL**-lel.

public Result CompileProject(BaristaProject baristaProject): lefordítom a megadott projektet. Ezt úgy teszem, hogy lekérem a **baristaProject függőségeit** és a ***target* mappáját**, majd meghívom a **Compile** metódust. Ha a fordítás sikeres volt, akkor **Result.OK**-val térítek vissza, egyébként **Result.FAIL**-lel.

private Result Compile(String sourceDirectory, List<String> files, HashMap<JavacEnum, Object> args): lefuttatom a **javac** parancsot a megadott paraméterek alapján. Az **args** *map-ből* származó argumentumokkal meghívom a **createArgumentFile** metódust, valamint a **files** listában található fájlokkal meghívom a **createSourceFile** metódust. Az így kapott két fájllal a **sourceDirectory** mappában lefuttatom a **„javac @argfile.txt @sourcefile.txt”** parancsot és ennek megvárom a végeredményét. Ha volt az **ErrorStream**-en kimenet, az azt jelenti, hogy a fordítás sikertelen volt, ilyenkor megnyitásra kerül egy Windows Parancssor, ami kiíratja a felmerülő hibákat. Egyéb esetben a fordítás sikeres volt. Ha a fordítás

sikeres volt, akkor **Result.OK**-val térítek vissza, egyébként **Result.FAIL**-lel. Ha a metódusban felmerül **IOException** vagy **InterruptedException,** akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public void RunFile(String filePath, String fileName): a megadott fájl útvonala és neve alapján lefuttatom azt. Meghívom a **CompileFile** metódust, és ha az sikeres, akkor meghívom a **Run** metódust.

public void RunProject(RunSetting runSetting): futtatom az éppen megnyitott projektet a megadott **runSetting** alapján. Meghívom a **CompileProject** metódust, és ha az sikeres, akkor meghívom a **Run** metódust.

private void Run(File argFile, String mainFile, String sourcePath, RunSetting runSetting): lefuttatom a **java** parancsot a megadott paraméterek alapján. Ha nincs megadva futási konfiguráció akkor a létrehozott *batch* fájl tartalma: **„@Echo off java @[argFile] [mainFile] pause”**, ha van megadva futási konfiguráció, akkor a *batch* fájl tartalma: **„@Echo off java [runSetting.getCommand()] pause”**. Ez a *batch* lesz lefuttatva. Miután a *batch* lefutása befejeződött, mind a *batch* mind az argumentum fájl törölve lesz. Ha a metódusban felmerül **IOException** vagy **InterruptedException,** akkor ezeket elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

public void openCommandPrompt(File path): megnyitom a Windows Parancssort a megadott fájl útvonalán. Ha a **path** null, akkor a felhasználó ***home*** könyvtárában lesz megnyitva. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelez a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

private File createArgumentFile(String path, HashMap<JavacEnum, Object> args): létrehozok egy **arguments.txt** fájlt, amiben a **javac** és a **java** parancs futtatásához szükséges parancssori argumentumok szerepelnek és ezzel vissza is térítek. Az argumentumokat az **args** *map* alapján állítom össze. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

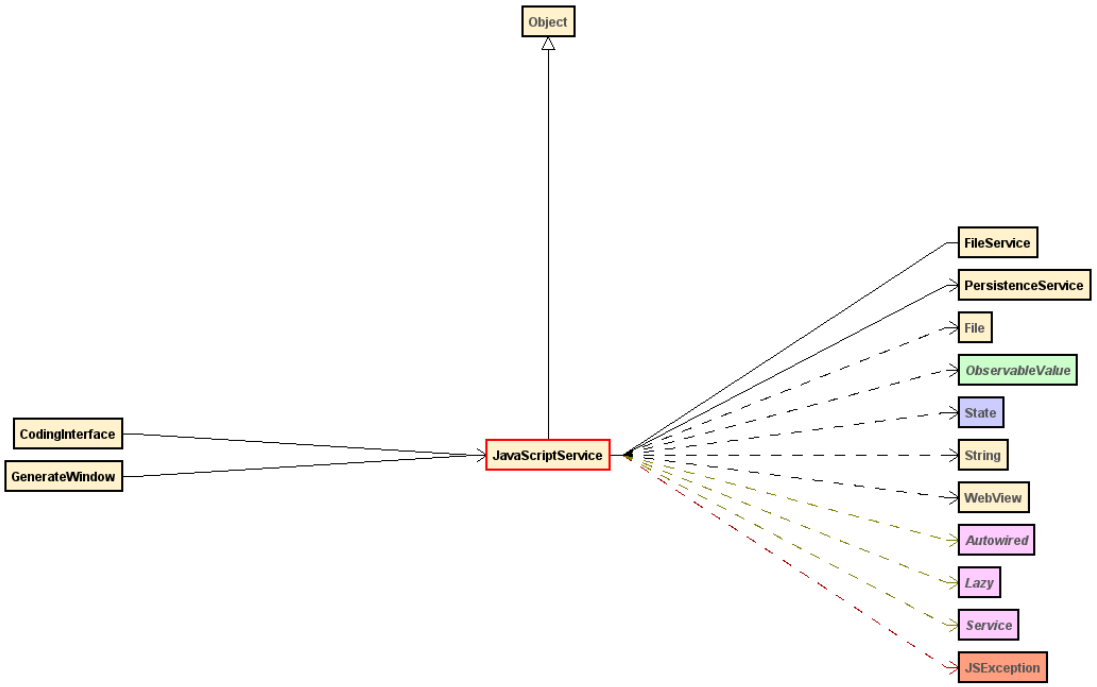
private File createSourceFile(String path, List<String> files): létrehozok egy

source.txt nevű fájlt, ami tartalmazza a **javac** parancshoz szükséges fájlok felsorolását és ezzel vissza is térítek. Ha a metódusban felmerül **IOException,** akkor ezt elkapom, és visszajelzek a felhasználónak egy **ErrorPopup megjelenítésével**, hogy hiba történt.

A JavaScripService osztály

A **JavaScripService** feladata a JavaScripttel való kommunikáció. Ezen az osztályon keresztül **kapcsolom össze a syntaxhighlight.js-t az IDE Javas részével**, valamint ebből az osztályból **futtatatom le az éppen szükséges JavaScript metódusokat**. Alább látható a JavaScriptService **UML diagramja** és **relációs diagramja** (20. ábra).

Object «@Service» com::farkasch::barista::services:: **JavaScriptService**
Fields
«@Lazy» «@Autowired» – fileService : FileService «@Lazy» «@Autowired» – persistenceService : PersistenceService
Constructors
+ JavaScriptService( ) : void
Methods
+ activateJavaMode(WebView) : void + activateTextMode(WebView) : void + getContent(WebView) : String + getCursorLine(WebView) : int + insertGeneratedContent(WebView) : void «synthetic» – lambda$open$0(WebView, String[], ObservableValue, State, State) : void «transient» – open(WebView, String[]) : void + setContent(WebView, File, boolean) : void + switchContent(WebView, File, boolean) : void



*20. ábra: a **JavaScriptService** relációs diagramja*

public String getContent(WebView view): meghívom a **view**-n a **syntaxhighlight.js getContent** metódusát, és annak az eredményével térítek vissza.

public void setContent(WebView, File fileToSet, boolean firstOpen): ha a **firstOpen true**, akkor meghívom az **open** metódust, ha **false,** akkor meghívom a **view**-n a **syntaxhighlight.js loadContent** metódusát.

public void switchContent(WebView, File fileToSet, boolean firstOpen): ha a **firstOpen true**, akkor meghívom az **open** metódust, ha **false,** akkor meghívom a **view**-n a **syntaxhighlight.js switchContent** metódusát.

public int getCursorLine(WebView view): meghívom a **view**-n a **syntaxhighlight.js getCursorLine** metódusát és visszatérítek az eredményével.

public void insertGeneratedContent(WebView view): meghívom a **view**-n a **syntaxhighlight.js insertGeneratedContent** metódusát.

private void open(WebView view, String.. scripts): meghívom a neki megadott scripteket a **view**-n. Ezt úgy teszem meg, hogy először megvárom, hogy a **view** betöltődjön, majd amikor ez megtörtént, végrehajtom a scripteket, valamint ekkor **adom át a szükséges Java osztályokat a syntaxhighlight.js-nek**, hogy elérhetővé váljanak számára.

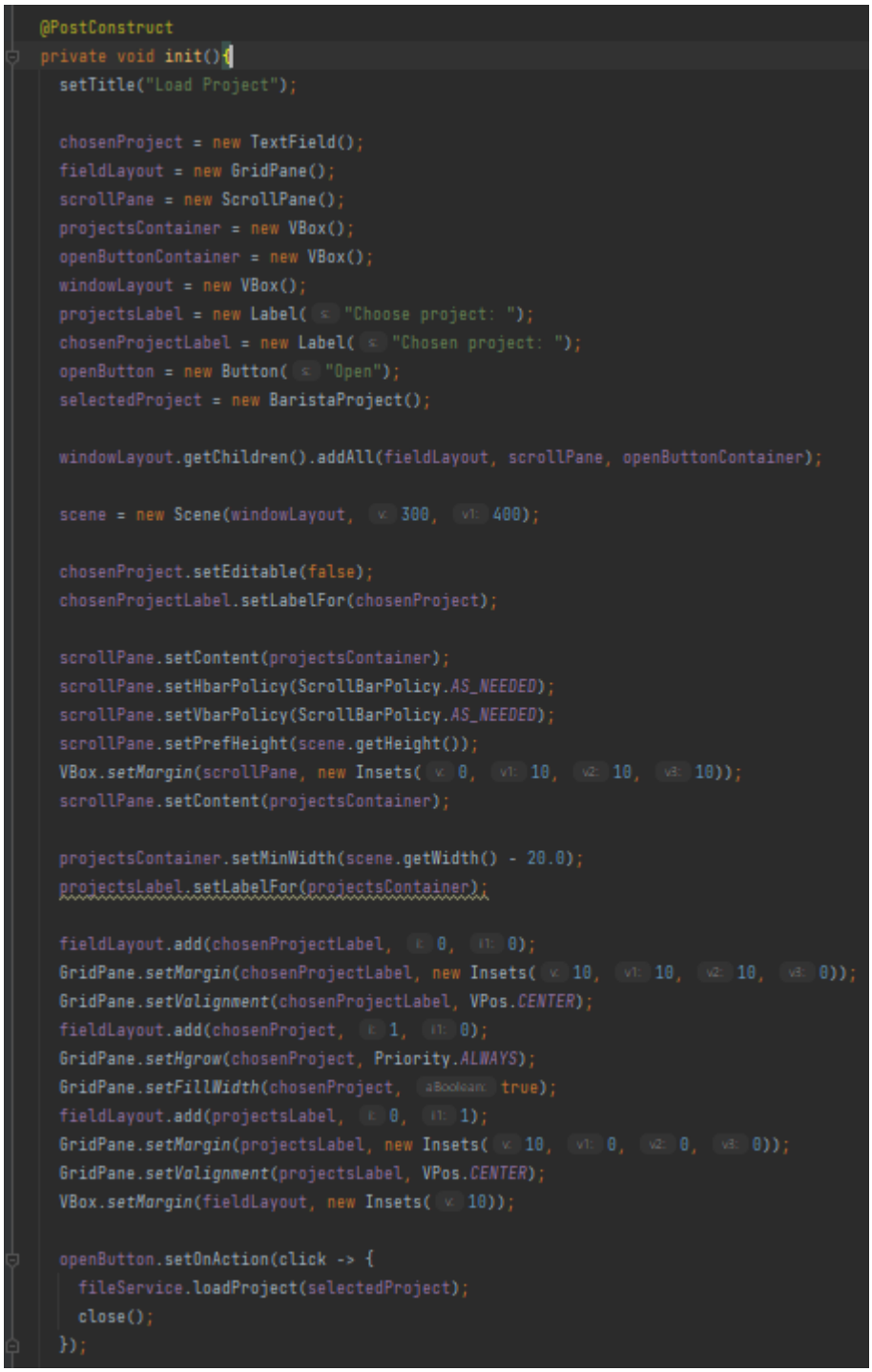
A program felépítése – a gui csomag

A **gui** csomag felosztása komponens-alapon történt. A kódolási felülettel kapcsolatos osztályok a **codinginterface**, az oldalsó menüvel kapcsolatos osztályok a **mainview.sidemenu**, a felső menüsorral kapcsolatos osztályok a **mainview.topmenu**, az egyéb, vagy több helyen előforduló komponensek pedig a **component** csomagban helyezkednek el. A **mainview** csomagban csak egy fájl található a **MainStage.java**.

Felugró ablakok

A felugró ablakok az egész programban egy egységes elképzelés, szerkezet alapján vannak elkészítve. Ahol lehetséges volt, tehát ahol eléggé hasonlítottak az osztályok egymáshoz, egy közös ősosztályból (pl.: **AbstractProjectPopup**) lettek leszármaztatva. Egy felugró ablak általános szerkezete a következő: rendelkezik egy **private void init()** metódussal, amely el van látva egy **PostConstruct** annotációval, aminek hatására a metódus rögtön az osztály konstruálásakor meghívódik. Ez a metódus tartalmazza az osztály „alapkövét” – itt lesznek a felületi elemek inicializálva, elhelyezve a képernyőn. Mivel ezek az osztályok mind **Spring bean**-ek, így csak egyszer lesznek megkonstruálva, így ez a metódus csak egyszer hívódik meg, tehát csomó erőforrást spórolok, ha a soha másként nem történő lépéseket itt valósítom meg. Ezen kívül rendelkezik még egy **private void onLoad** metódussal is, ami kezeli az ablak megjelenítésekor szükséges műveleteket. Végül van a **public void showWindow**, amely meghívja az **onLoad** metódust a megfelelő paraméterekkel (ezek általában ugyanazok, amelyeket ő is megkapott) és megjeleníti az ablakot.

Vizsgáljuk meg példának okáért a **LoadProjectWindow** metódusait:



```
@PostConstruct
private void init(){
    setTitle("Load Project");

    chosenProject = new TextField();
    fieldLayout = new GridPane();
    scrollPane = new ScrollPane();
    projectsContainer = new VBox();
    openButtonContainer = new VBox();
    windowLayout = new VBox();
    projectsLabel = new Label( s: "Choose project: ");
    chosenProjectLabel = new Label( s: "Chosen project: ");
    openButton = new Button( s: "Open");
    selectedProject = new BaristaProject();

    windowLayout.getChildren().addAll(fieldLayout, scrollPane, openButtonContainer);

    scene = new Scene(windowLayout,  v: 300,  v1: 400);

    chosenProject.setEditable(false);
    chosenProjectLabel.setLabelFor(chosenProject);

    scrollPane.setContent(projectsContainer);
    scrollPane.setHbarPolicy(ScrollBarPolicy.AS_NEEDED);
    scrollPane.setVbarPolicy(ScrollBarPolicy.AS_NEEDED);
    scrollPane.setPrefHeight(scene.getHeight());
    VBox.setMargin(scrollPane, new Insets( v: 0,  v1: 10,  v2: 10,  v3: 10));
    scrollPane.setContent(projectsContainer);

    projectsContainer.setMinWidth(scene.getWidth() - 20.0);
    projectsLabel.setLabelFor(projectsContainer);

    fieldLayout.add(chosenProjectLabel,  i: 0,  i1: 0);
    GridPane.setMargin(chosenProjectLabel, new Insets( v: 10,  v1: 10,  v2: 10,  v3: 0));
    GridPane.setValignment(chosenProjectLabel, VPos.CENTER);
    fieldLayout.add(chosenProject,  i: 1,  i1: 0);
    GridPane.setHgrow(chosenProject, Priority.ALWAYS);
    GridPane.setFillWidth(chosenProject,  aBoolean: true);
    fieldLayout.add(projectsLabel,  i: 0,  i1: 1);
    GridPane.setMargin(projectsLabel, new Insets( v: 10,  v1: 0,  v2: 0,  v3: 0));
    GridPane.setValignment(projectsLabel, VPos.CENTER);
    VBox.setMargin(fieldLayout, new Insets( v: 10));

    openButton.setOnAction(click -> {
        fileService.loadProject(selectedProject);
        close();
    });
```

*21. ábra: a **LoadProjectWindow init** metódusa*

A 21. ábrán látható kódrészlet a **LoadProjectWindow init** metódusa. Először inicializálom az összes helyi változót, majd ezek pozíciójá beállítom az ablakon belül, valamint ekkor adom hozzá gombok akcióit is. Látható, hogy az init-ben csak **változókat állítok be, amelyek az ablak élettartama alatt biztosan nem fognak változni.**

```
private void onLoad(){
  chosenProject.setText("");
  List<BaristaProject> projects = fileService.getProjects();
  projectsContainer.getChildren().clear();
  for(BaristaProject project : projects){
    Button openProject = new Button(project.getProjectName());
    openProject.setId("folder-dropdown__item");
    openProject.setGraphic(new FontIcon( iconCode: "mdi-folder"));
    openProject.setOnAction(click -> {
      chosenProject.setText(project.getProjectName());
      selectedProject = project;
    });
    openProject.setMinWidth(projectsContainer.getMinWidth());
    openProject.setMaxWidth(Double.MAX_VALUE);
    openProject.setMaxHeight(Double.MAX_VALUE);
    projectsContainer.getChildren().add(openProject);
  }
}
```

*22. ábra: a **LoadProjectWindow onLoad** metódusa*

A 22. ábrán látható kódrészlet a **LoadProjectWindow onLoad** metódusa. Itt kerül beállításra, hogy milyen projektek lesznek láthatók a menüben. Ezt **minden alkalommal újra kell megadni, amikor megnyitjuk az ablakot**, ezért az onLoad-ban végezzük el.

```
public void showWindow(){
  onLoad();
  setScene(scene);
  show();
}
```

*23. ábra: a **LoadProjectWindow showWindow** metódusa*

A 23. ábrán látható kódrészlet a **LoadProjectWindow showWindow** metódusa. Ezzel a metódussal lehet **megjeleníteni az ablakot**, **és innen hívódik meg az onLoad is**.

A codinginterface csomag

Ebben a csomagban helyeztem el a kódolási felülettel kapcsolatos osztályokat. Most ezeket az osztályokat fogom röviden bemutatni.

A **CodingInterfaceContainer** osztály **tárolja magában a kódolási felületet/ felületeket**. Ez az osztály kezeli az ***interfacek* megnyitását, bezárását,** valamint itt döntöm el, **hogy drag-and-drop esetén hol nyíljon meg az új fájl**.

A **CodingInterface** a kódolási felületet megvalósító osztály. Tartalmaz egy **SwitchMenu**-t, ami felelős a fájlok közötti váltogatásért, valamint itt található az a **WebView elem, aminek segítségével szerkeszthetem a megnyitott fájlt**. Ezen a WebView elemen keresztül jelenítem meg **a JavaScript által kiszínezett, karbantartott kódolási felületet**.

A **SwitchMenu** segítségével **tudok fájlok között váltogatni** a CodingInterface-en. Számon tartja az éppen szerkesztett fájlt, és a benne definiált **SwitchMenuItem**-ek **tartalmazzák az éppen megnyitott, de nem látható fájlokat**. A **SwitchMenuItem-**ek **drag-and-drop segítségével** behúzhatók a CodingInterfaceContainer-be, hogy kettéosszák azt két külön CodingInterface-re.

A **GenerateWindow** a CodingInterface jobbklikkmenüjéből nyitható meg. Ezzel az osztállyal valósítom meg azt az ablakot, aminek segítségével lehet *gettert*, *settert* és *konstruktort* generálni.

A sidemenu csomag

Ebben a csomagban találhatóak az oldalsó menüvel kapcsolatos osztályok. Most ezeket az osztályokat fogom röviden bemutatni.

- **AbstractProjectPopup** -> absztrakt osztály, amelyből az ebben a csomagban lévő felugró ablakok nagyrésze leszármazik.
- **NewFilePopup** -> ezzel implementálom a megnyitott projektben egy fájl létrehozásáért felelős felugró ablakot.
- **NewFolderPopup** -> ezzel implementálom a megnyitott projektben egy mappa létrehozásáért felelős felugró ablakot.
- **RenameFilePopup** -> ezzel implementálom a megnyitott projektben egy fájl átnevezéséért felelős felugró ablakot.
- **RenameFolderPopup** -> ezzel implementálom a megnyitott projektben egy mappa átnevezéséért felelős felugró ablakot.

- **RenameProjectPopup** -> ezzel implementálom az éppen megnyitott projekt átnevezéséért felelős felugró ablakot.

- **RunConfigWindow** -> ezzel implementálom a futási konfigurációk beállításáért felelős felugró ablakot.

A **SideMenu-vel** implementálom magát az oldalsó menü kinézetét és működését. Itt kapják meg funkcionalitásukat az oldalsó menün lévő gombok, valamint a lenyíló projekt elemei itt kapják meg a jobbklikkmenüs elemeiket. A lenyíló **projekt** egy **FolderDropDown** osztállyal oldottam meg, míg **az éppen megnyitott és bezárt fájlok** (ha nincs aktív projekt) a **SimpleDropdown** osztály segítségével vannak valósítottam meg.

A topmenu csomag

A **topmenu** csomag tartalmazza felső menüvel kapcsolatos osztályokat. Most ezeket az osztályokat fogom röviden bemutatni.

- **LoadProjectWindow** -> ezzel implementálom a projektek betöltéséért felelős felugró ablakot.

- **NewFileWindow** -> ezzel implementálom az új fájlok létrehozásáért felelős felugró ablakot.

- **NewProjectWindow** -> ezzel implementálom a projektek létrehozásáért felelős felugró ablakot.

- **OpenFileWindow** -> ezzel implementálom a fájlok megnyitásáért felelős felugró ablakot

A **TopMenu-vel** valósítom meg a felső menünek az elrendezését. Itt hozom létre a **főbb menüpontokat**, és ezek itt **töltöm fel menüelemekkel**. Jelenleg 2 menüelemet tartalmaz: ***File*** és ***Help.***

A component csomag

A **component** csomag tartalmazza azokat az osztályokat, amelyek olyan felületi elemeket implementálnak, amelyek több helyen szereplő, általános komponensek.

Az **ErrorPopup** osztállyal implementálom azt a felugró ablakot, amely akkor jelenik meg, ha a program futása során valami hiba lép fel. Szerepel rajta egy hibaüzenet,

egy **OK** gomb, és egy ***Error Log*** gomb, aminek megnyomásával a felhasználónak megnyílik a Windows Intézőben az **…\AppData\Roaming\BaristaIDE\logs** mappa, ahol a létrejött **error_log_[mai dátum]** fájlban szerepel a felmerült hiba részletes *stack trace*-je.

A **WarningPopup** osztállyal valósítom meg azt a felugró ablakot, ami esetleges nem megengedett *inputoknál* jelenik meg, vagy amikor valami fontos információt szeretnék a felhasználóval megosztani. Szerepel rajta a felhasználóval megosztott üzenet, egy **OK** gomb és szükség esetén egy ***Cancel*** gomb.

A **SimpleDropdown** osztállyal valósítom meg azt a felületi elemet, ahol van egy szülő-elem, aminek több gyermek eleme van, amik a szülőre kattintva megjelennek és eltűnnek. A SimpleDropdown elemei fájlok, amikre kattintva azok megnyithatók. A **SideMenu**-ben szereplő **openFiles** és **recentlyClosed** elemeket ezzel implementálom.

A **FolderDropdown** implementálja azt a felületi elemet, ahol van (akár több) szülő elem, aminek több gyermeke van, és ezeknek a gyermek-elemeknek is lehetnek gyermekei, és így tovább. Ennek a komponensnek a programban be lehet állítani a design-ját, jobbklikkmenüjeit, illetve azt, hogy bizonyos elemekre kattintáskor mi történjen. Többek között a SideMenuben a **projectDropdown** elemet implementálom ezzel.

A MainStage osztály

A **MainStage** osztály a program felületének a fő eleme. Itt helyezem el az oldalsó menü, a felső menüsáv és a kódolási felületet is. Ez az osztály szolgál az IDE legfőbb ablakaként, ez lesz elindítva a **JavaFxApp** *main* metódusában.

A program felépítése – a util csomag

A **util** csomag tartalmazza az általam létrehozott segédosztályokat. Ezek az osztályok általában egy specifikus feladat elvégzésre lettek létrehozva, hatáskörük meglehetősen szűk.

Az enums csomag

Az **enums** csomag tartalmazza a programban felhasznált felsorolási típusokat.

Ezeket alább felsorolom.

- **FileExtensionEnum** -> támogatott fájlkiterjesztéseket sorolok fel benne. Tagjai: **JAVA, TXT, XML, OTHER**.

- **GenerateEnum** -> felsorolom, hogy kód generálásakor milyen típusú kódrészlet generálódjon. Tagjai: **GETTER, SETTER, GETTER_AND_SETTER, CONSTRUCTOR.**

- **JavacEnum** -> a javac parancshoz tartozó paramétereket sorolom fel benne. Tagjai: **D, CLASSPATH, SOURCEPATH**

- **JavaClassTypesEnum** -> a Java osztálytípusait sorolom fel benne. Tagjai: **CLASS, ENUM, INTERFACE, ANNOTATION, RECORD.**

- **ProjectTypeENum** -> különböző java projekttípusokat sorolom fel benne. Tagjai: **BASIC, MAVEN, GRADLE.** Jelenleg csak a **BASIC** van használatban.

- **ResultTypeEnum** -> A **Result** osztályhoz tartozó eredménytípusokat sorolom fel benne. Tagjai: **OK, FAIL, ERROR.**

A settings csomag

A **settings** csomag tartalmazza az IDE különböző beállítástípusait megvalósító osztályokat. Jelenleg csak az **AbstractSetting** és az ebből leszármazó **RunSetting** található benne, de a program fejlesztésével ez a csomag bővülni fog. Az **AbstractSetting**-ből származik le az összes itt található osztály. Maga az **AbstractSetting** üres, létezésének célja, hogy ezeket az osztályokat könnyebben lehessen egy listában tárolni. Az eddig egyetlen „éleseben is használt” osztály a **RunSetting,** amiben egy projekt futási beállításait tárolom.

A BaristaDragBoard osztály

Ez az osztály felelős a ***drag-and-drop*** metódusok rendeltetésszerű működéséért. Ha egy elemen *drag-and-drop* akció lesz elkezdve, akkor az az elem bekerül a BaristaDragBoard osztály **draggedItem** változójába. Ebből a változóból lehet aztán kiolvasni egy sikeres *drag-and-drop* esetén a szükséges adatokat. Az osztályban tárolom a ***drag-and-drop*** **célpontját** is a **dragTarget** változóban, mert vannak esetek, amikor a draggedItem-nek ezt el kell érnie **validálás szempontjából**.

A BaristaProject osztály

A BaristaProject osztály a program által használt projektfelépítés absztrakciója. A **ProjectConfig.json**-ból kiolvasott adatokat ebben az osztályban tárolom, és ettől az osztálytól kérem le. A ProjectConfig.json osztályból való olvasás és az abba való írás érdekében tartalmaz egy **toJsonString** és egy **fromJsonString** metódust. A **toJsonString** metódussal visszaadok az osztály alapján egy JSON formátumú Stringet, a **fromJsonString** metódussal beállítom az osztály adatait egy JSON formátumú Stringből.

A FileTemplates osztály

Ebben az osztályban találhatók a különböző automatikusan generált fájlrészletek. Az alább felsorolt metódusokkal a megadott adatok alapján generálom a szükséges fájlrészleteket.

- **mainTemplate** -> az automatikusan létrehozott *main* fájl sablonja.
- **classTemplate** -> az automatikusan generált *class* java fájl sablonja.
- **enumTemplate** -> az automatikusan generált *enum* java fájl sablonja.
- **interfaceTemplate** -> az automatikusan generált *interface* java fájl sablonja.
- **annotationTemplate** -> az automatikusan generált *annotation* java fájl sablonja.
- **recordTemplate** -> az automatikusan generált *record* java fájl sablonja.
- **createSetter** -> az automatikusan generált *setter* metódus sablonja
- **createGetter** -> az automatikusan generált *getter* metódus sablonja
- **createConstructor** -> az automatikusan generált konstruktor sablonja
- **createPackage** -> az automatikusan generált csomag sablonja
- **createImport** -> az automatikusan generált *import* sablonja.

A Result osztály

Ennek az osztálynak a segítségével implementáltam azt, hogy a metódusok „érvénytelenül” is vissza tudjanak térni. Található benne egy **String message** adattag, ami egy üzenetet hordoz magával, ami pl. egy hiba kiíratásánál fontos lehet. Található benne egy **ResultTypeEnum result** adattag, ami alapján eldönthető,

hogy a Result **OK, FAIL** vagy **ERROR** állapotban van. Található benne egy **File logFile** adattag, ami egy **ERROR** típusú Result esetén a hibaüzenetre mutató fájl reprezentációját tartalmazza. Található benne egy **Object returnValue** adattag, ami magát a metódus által visszatérített értéket tartalmazza, ha létrehozott Result **OK** típusú.

A Result-nak **privát a konstruktora**. Result-ot csak a nyilvános és statikus **OK, FAIL** és **ERROR** metódusokkal lehet létrehozni.

A TreeNode osztály

A TreeNode osztállyal egy faszerkezetet valósítok meg rekurzió segítségével. Ennek az osztálynak a segítségével van a **FolderDropdown** egy része megvalósítva. Egy TreeNode tartalmazza a szülő *node*-ját, a gyermekei listáját, és egy tetszőleges típusú értéket. Fontosabb metódusok:

- A **getHeight** rekurzívan bejárom a fát, és visszaadom annak magasságát.
- A **cutBelow** levágom a *node*, így levelet csinálva végül belőle.
- A **doActionPreorder** végrehajtom a megadott akciót a fa egészén, preorder módon bejárva.
- A **doActionPostorder** végrehajtom a megadott akciót a fa egészén, postorder módon bejárva.
- A **removeNode** eltávolítom a megadott *node-ot* a fából. Ezt rekurzívan keresem meg, és az első megtalált ilyen *node-nál* megállok.

A program felépítése – syntaxhighlight.js és codearea.html

A codearea.html

A **codearea.html** egyetlen egy elemet tartalmaz: Egy **codearea** *id*-vel rendelkező **textarea** *tag*-et. Ebben a fájlban lesz továbbá belinkelve a CodeMirror **codemirror.js, clike.js és codemirror.css** fájljai, valamint a saját **barista-style.css és syntaxhighlight.js** fájljaim.

A syntaxhighlight.js

A **syntaxhighlight.js** segítségével lesz a CodeMirror által kezelt kódolási felület

bekonfigurálva, valamint **ez a JavaScript fájl kapcsolódik a Java**-hoz és kéri le a tőle a szükséges információkat (pl. az épp beillesztendő kódrészletet.)

A **CodeMirror.fromTextArea-val** inicializálom a CodeMirror által kezelt kódolási felületet. A codearea.html-ben található textArea-t használom fel erre a célra. Ebben a metódusban állítom be, hogy milyen nyelvet ismejen fel, milyen stílust használjon, mutassa a sorok számát, legyen *autofocus* és a tabulátorok mérete 2 szóköz legyen.

A **loadContent**-tel betöltöm a **PersistenceService.getContentToOpen**-ben található szöveget a kódolási felületre. **Ezt a metódust a program Javas részéből hívom meg.**

A **switchContent**-tel betöltöm a **PersistenceService.getContentToSwitch**-ben található szöveget a kódolási felületre. **Ezt a metódust a program Javas részéből hívom meg.**

A **getContent**-tel visszaadom a kódolási felületnek a szöveges tartalmát. **Ezt a metódust a program Javas részéből hívom meg.**

A **getCursorLine**-nal visszaadom azt a sort, ahol a kurzor éppen elhelyezkedik. **Ezt a metódust a program Javas részéből hívom meg.**

Az **insertGeneratedContent**-tel beillesztem a **PersistenceService.getGenerateInsertPosition** által megadott helyre a **PersistenceService.getGeneratedContent** tartalmát. **Ezt a metódust a program Javas részéből hívom meg.**

Tesztelési terv

A kétféleképpen teszteltem a programom működését. Készítettem automatikus *unit teszteket* (egység teszteket) a Java **JUnit** könyvtárának[12] és a Java **Mockito** könyvtárának[13], a testfx könyvtárnak[14] és az assertj könyvtárnak[15] segítségével, amivel a *services* osztály fontos és nyilvános metódusait teszteltem. Emellett készítettem egy leírást felület helyességének kézi tesztelésére, amit alább fogok részletesen kifejteni.

Unit tesztek

A *unit* tesztek a forráskódban az **src\test** mappában helyezkednek el. Az ezen belül elhelyezkedő **java** mappában található **tests** csomag tartalmazza a tesztfájlokat, valamint a tesztelés könnyítését szolgáló **TestHelper.java** osztályt. Az **src\test** mappában található még egy **resources** mappa is, ami a különböző tesztekhez (pl. fájlok törlésének tesztelése) tartalmaz fájlokat, mappákat.

A unit tesztek felépítése

Minden *unit* teszt osztály az **ApplicationTest**[hivatkozás] osztályból származik le, ez a JavaFx-hez elengedhetetlen ***JavaFX Application Thread*** elindításához szükséges. A ***JavaFX Application Thread***-et a **start** metódus indítja el, amit minden osztálynak felül kell írnia. Ez a metódus minden teszteset előtt lefut, ezért itt inicializálom a több helyen szükséges elemeket.

A tesztelést a JUnit 4.13.2-es verziójával végzem. A JUnit a Java legelterjedtebb tesztelési keretrendszere. A teszteket tartalmazó osztályok metódusait a **@Test** annotációval jelöltem meg, jelezve a JUnit-nak hogy ezek tesztesetek. Az összehasonlításokat a JUnit **Assert** osztályában található metódusok segítségével végzem. Egy ilyen metódus pl. az **assertTrue(boolean value)**, ami ha **true** értéket kap, akkor tovább lép, viszont, ha **false** értéket kap **elbuktatja a tesztet.**

Az egység teszteknél szükséges volt néhány **függőséget kiváltanom** a **hatékonyság érdekében**, vagy mert **nem azt az osztályt teszteltem**. Ehhez a **Mockito** 4.5.1-es verzióját használtam. A Mockito segítségével dublőröket, „álosztályokat” vagy ***mock***-okat lehet létrehozni. Ezeknek a *mock*-oknak **meg lehet szabni a**

viselkedésüket, ha pl. egy metódus meghívódik, mit adjanak vissza. Az alábbi ábrán (24. ábra) található erre egy példa.

```
@Test
public void getGenerateInsertPositionTest(){
    File testFile = new File( pathname: "src\\test\\resources\\FileServiceTest\\ClassLevelVariables.java").getAbsoluteFile();
    File noMethodTestFile = new File( pathname: "src\\test\\resources\\FileServiceTest\\NoMethodTest.java").getAbsoluteFile();
    int randomNumber = (int)(Math.random() * 100);
    Mockito.doReturn(randomNumber).when(javaScriptService).getCursorLine(Mockito.any());
    CodingInterface codingInterface = Mockito.mock(CodingInterface.class);

    int result = fileService.getGenerateInsertPosition(TestHelper.copyFileContent(testFile), codingInterface);
    Assert.assertEquals( expected: 4, result);

    result = fileService.getGenerateInsertPosition(TestHelper.copyFileContent(noMethodTestFile), codingInterface);
    Assert.assertEquals( expected: randomNumber - 1, result);
}
```

*24. ábra: a **getGenerateInsertPositionTest** teszteset*

A 24. ábrán látható, hogy (a start metódusban *mock*-ként létrehozott) **javaScriptService** osztályon hogyha meghívom a **getCursorLine** metódust, a metódus nem lesz lefuttatva, hanem helyette a **randomNumber** lesz a visszatérési érték. Ez itt azért fontos, **mert ezzel biztosítom, hogy a teszteset sikeressége csak az éppen tesztelni kívánt osztálytól függ**, és nem egy másiktól.

A FileService osztály tesztelése

A **FileService** osztályt a **tests.fileservice** csomagban lévő fájlok segítségével tesztelem.

A **JarConfigOperationsTest** osztályban ellenőrzöm, hogy az **…\AppData\Roaming\BaristaIDE\config\JarConfig.json** fájl manipulálásával kapcsolatos metódusok rendeltetésszerűen működnek-e. Ebben az osztályban a **FileService** osztály **addNewJarConfig**, **updateNameInJarConfig**, **UpdateJarsInJarConfig** és **getJarsForFile** metódusait tesztelem.

Az **OutsideProjectOperationsTest** osztályban tesztelem azokat a fájlokkal kapcsolatos műveleteket, amik a projekten kívüli fájlokkal foglalkoznak. Ezek a metódusok a **FileService** osztály **createFile**, **deleteFile** és **renameFile** metódusainak azon esetei, amikor nincs épp megnyitva projekt.

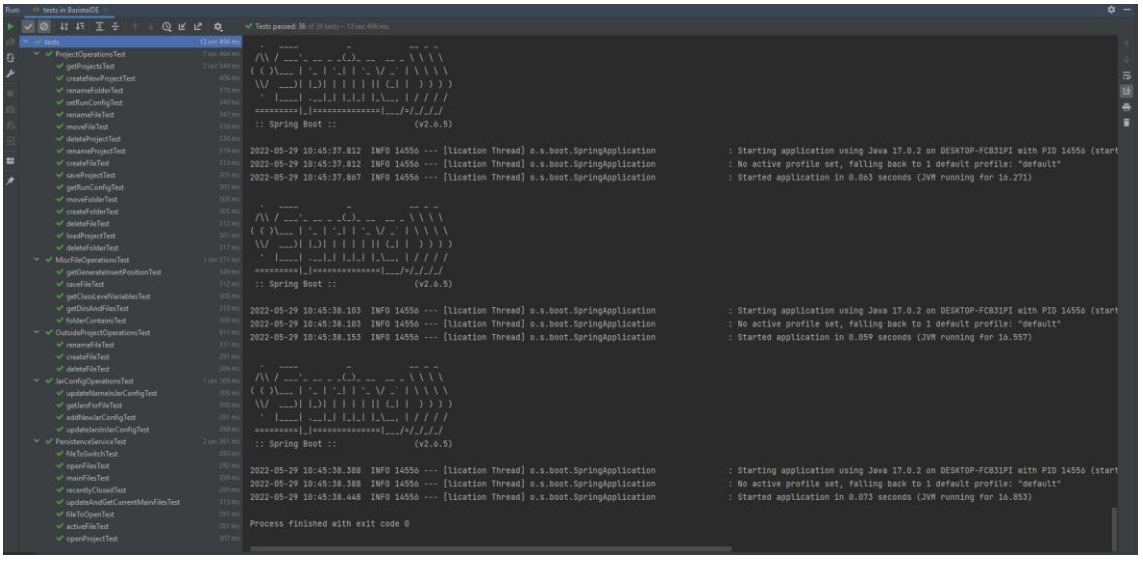
A **ProjectOperationsTest** osztályban tesztelem azokat a fájlokkal kapcsolatos műveleteket, amikor az adott fájl része egy projektnek. Ezek a metódusok a **FileService** osztály **createFile**, **createFolder**, **deleteFile**, **deleteFolder**, **createNewProject**, **loadProject**, **saveProject**, **deleteProject**, **renameProject**,

getProjects, **getRunConfig**, **setRunConfig**, **renameFile**, **renameFolder**, **moveFile** és **moveFolder** metódusai.

A **MiscFileOperationsTest** osztályban a **FileService** osztály azon metódusait tesztelem, amelyek a fentiek közül felsorolt egyik kategóriába sem tartoznak bele. Ezek a metódusok a **saveFile**, a **getDirsAndFiles**, a **getClassLevelVariables**, a **getGenerateInsertPosition** és a **folderContains**.

A PersistenceService osztály tesztelése

A **PersistenceService** osztály a **tests.persistenceservice** csomagban lévő **PersistenceServiceTest** osztállyal tesztelem. Az itteni tesztek azt hivatottak ellenőrizni, hogy a PersistenceService-től kért változtatások valóban megtörténnek-e. Ez azért fontos, mert a program *perzisztenciája*, vagyis a program állandósága ettől az osztálytól függ. Ha a program egyik részében beállítom az **openPoject** értékét, annak a program többi részében is tükröződnie kell. Ez az osztály teszteli a **PersistenceService setActiveFile**, **setFileToOpen**, **setFileToSwitch**, **addOpenFile**, **removeOpenFile**, **addMainFile**, **removeMainFile**, **addRecentlyClosed**, **removeRecentlyClosed**, **setOpenProject** és **updateAndGetCurrentMainFiles** metódusait.



25. ábra: A program unit tesztjeinek sikeres lefutása az IntelliJ IDEA-ben.

Felületi tesztek

A felületi tesztekkel ellenőrzöm, hogy a program gombjai és egyéb *interface* elemei rendeltetésszerűen működnek-e. A felületi teszteket kézzel kell elvégezni. Ezek

menetét alább részletezem.

A New File ablak felületi tesztje

1. Felső menüben a *File* gombra kattintva megjelenik a leugró menü.

2. Leugró menüben a *New File*-ra kattintva megjelenik a felugró ablak

3. A *File Name* mezőbe és a *Folder Path* mezőbe lehet írni

4. A *Choose destination folder* alatti területen az elemekre kattintva azok lenyílnak, és a legutoljára kattintott elem bekerül a *Folder path* mezőbe.

5a. A *Create* gomb megnyomása után, ha a létrehozni kívánt fájl már létezik egy erre figyelmeztető ablak jelenik meg.

5b. A *Create* gomb megnyomása után, ha a *File name* vagy *Folder Path* mező üres, akkor egy erre figyelmeztető ablak jelenik meg.

5c. A *Create* gomb megnyomása után, ha a létrehozni kívánt fájl még nem létezik az ablak bezáródik, és a fájl megnyílik a kódolási felületen.

5d. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik.

A New Project ablak felületi tesztje

1. Felső menüben *File* a gombra kattintva megjelenik a leugró menü.

2. Leugró menüben a *New File*-ra kattintva megjelenik a felugró ablak

3. A *Project name* mezőbe és a *Project creation folder* mezőbe lehet írni

4. A *Choose destination folder* alatti területen az elemekre kattintva azok lenyílnak, és a legutoljára kattintott elem bekerül a *Project creation folder* mezőbe.

5a. A *Create* gomb megnyomása után, ha már van ilyen nevű projekt, akkor egy erre figyelmeztető ablak jelenik meg.

5b. A *Create* gomb megnyomása után, ha a *Project name* vagy *Project creation folder* mező üres, akkor egy erre figyelmeztető ablak jelenik meg.

5c. A *Create* gomb megnyomása után, ha még nincs ilyen nevű projekt az ablak bezáródik és az oldalsó menüben megnyílik az új projekt.

5d. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik.

Az Open File ablak felületi tesztje

1. Felső menüben a *File* gombra kattintva megjelenik a leugró menü.

2. Leugró menüben a *Open File*-ra kattintva megjelenik a felugró ablak

3. A *Chosen File* mezőbe nem lehet írni.

4. A *Choose a file* alatti területen az elemekre kattintva azok lenyílnak.

5. A legutoljára kattintott fájl neve megjelenik a *Chosen File* mezőben. Mappára kattintáskor ez nem történik meg.

6a. Az *Open* gomb megnyomásakor a kiválasztott fájl megnyílik a kódolási felületen.

6b Az Open gomb megnyomásakor, ha a *Chosen File* mező üres, akkor egy erre figyelmeztető ablak jelenik meg.

6c. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik.

A Load Project ablak felületi tesztje

1. Felső menüben a *File* gombra kattintva megjelenik a leugró menü.

2. Leugró menüben a *Load Project*-re kattintva megjelenik a felugró ablak

3. A *Chosen Project* mezőbe nem lehet írni.

4. A *Choose project* alatti területen az elemekre kattintva azok neve megjelenik a *Chosen project* mezőben.

5a. Az *Open* gomb megnyomásakor a kiválasztott projekt megnyílik az oldalsó menüben

5b. Az Open gomb megnyomásakor, ha a *Chosen project* mező üres, akkor egy erre figyelmeztető ablak jelenik meg.

5c. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik.

Az Open Documentation felületi tesztje

1. Felső menüben a *Help* gombra kattintva megjelenik a leugró menü.

2. Az *Open Documentation*-re kattintva megnyílik a dokumentáció pdf formátumban.

Az Open File lenyíló menü felületi tesztje

1a. Ha nincs megnyitva projekt, akkor látható a *Recently Closed* alatt.

1b. Ha van megnyitva projekt, akkor nem látható.

2a. Ha van megnyitva fájl, akkor rákattintva megjelenik az összes éppen megnyitott fájl.

2b. Ha nincs megnyitva fájl, akkor rákattintva nem jelenik meg semmi.

3. Ha a lenyitott fájlok közül az egyikre rákattintunk, akkor az fog láthatóvá válni a kódolási felületen.

4 Ha egy fájlt megnyitunk, az megjelenik a lenyíló fájlok között.

5. Ha egy fájlt bezárunk, az eltűnik a lenyíló fájlok közül.

6. Ha ismét rákattintunk, akkor a megjelenített fájlok eltűnnek.

7. A kódolási felület kettéosztás *drag-and-drop*-pal nem változtat semmit az *Open File*-on.

8. Az eddig felsorolt funkciók kettéosztott kódolási felület mellett is működnek.

A Recently Closed lenyíló menü felületi tesztje

1a. Ha nincs megnyitva projekt, akkor látható a *Open Files* felett.

1b. Ha van megnyitva projekt, akkor nem látható.

2a. Ha még nem lett egy fájl sem bezárva, akkor rákattintva nem jelenik meg semmi.

2b. Ha volt bezárva fájl, akkor rákattintva megjelenik az összes bezárt fájl.

3a. Ha a lenyitott fájlok közül az egyikre rákattintunk és az nincs megnyitva, akkor megnyílik a kódolási felületen.

3b. Ha a lenyitott fájlok közül az egyikre rákattintunk és az meg van nyitva, akkor az fog láthatóvá válni a kódolási felületen.

4. Ha egy fájlt bezárunk, az megjelenik a lenyíló fájlok között.

5. Ha ismét rákattintunk, akkor a megjelenített fájlok eltűnnek.

6. A kódolási felület kettéosztás *drag-and-drop*-pal nem változtat semmit az *Recently Closed*-on.

7. Az eddig felsorolt funkciók kettéosztott kódolási felület mellett is működnek.

A Compile gomb felületi tesztje (projekten kívül)

1a. Ha nincs kiválasztva *main* fájl, akkor inaktív, nem kattintható.

1b. Ha ki van választva *main* fájl, akkor kattintható.

2a. Kattintásra lefordítja a kiválasztott *main* fájlt. Ha sikeres, megjelenik egy felugró ablak, ami tájékoztatja a felhasználót a futtatás sikerességéről.

2b. Kattintásra lefordítja a kiválasztott *main* fájlt. Ha hiba lép fel, megnyílik egy Windows Parancssor a hibaüzenettel.

A Run gomb felületi tesztje (projekten kívül)

1a. Ha nincs kiválasztva *main* fájl, akkor inaktív, nem kattintható.

1b. Ha ki van választva *main* fájl, akkor kattintható.

2. Kattintásra lefordítja és futtatja a kiválasztott *main* fájlt, majd ennek eredményével megjelenít egy Windows Parancssort.

Az Open Command Line gomb felületi tesztje (projekten kívül)

1a. Ha van kiválasztott *main* fájl, akkor kattintásra megnyílik egy Windows Parancssor a *main* fájlt tartalmazó mappában.

1b. Ha nincs kiválasztott *main* fájl, akkor kattintásra megnyílik egy Windows Parancssor a felhasználó *home* mappájában.

A Main fájl választó felületi tesztje (projekten kívül)

1a. Ha nincs megnyitva a program által *main* fájlként felismert fájl, akkor üres és kattintásra egy üres lenyíló menü jelenik meg.

1b. Ha meg van nyitva a program által *main* fájlként felismert fájl, akkor üres és kattintásra a lenyíló menüben megjelenik a megnyitott *main* fájlok elérési útvonala.

2. A lenyíló menüben egy elemre kattintva, a menü bezáródik és a kiválasztott útvonal megjelenik a lenyíló menü tetején. A *Compile, Run* és *Settings* gombok kattinthatóvá válnak.

3. Új *main* fájl megnyitásakor az hozzáadódik a lenyíló menühöz.

4. *Main* fájl bezárásakor az eltávolítódik a menüből.

5. Ha az éppen kiválasztott *main* fájl lesz bezárva, akkor a kiválasztás kiüresedik, és a *Compile, Run* és *Settings* gombok nem lesznek többé kattinthatóak.

A Settings ablak felületi tesztje (projekten kívül)

1a. Ha van kiválasztott *main* fájl, akkor kattintásra megnyílik a *Settings* ablak.

1b. Ha nincs kiválasztott *main* fájl, akkor a *Settings* gomb nem kattintható.

2. A *Settings* ablakban csak az *Add Dependencies* kattintható a felső menüsorban.

3. A *Select jar dependencies* alatti területen megjelennek a hozzáadott *jar* függőségek.

4. Ha a *jar* függőség neve mellett található mínusz gombra kattintunk a függőség eltűnik a listából.

5. A *Browse…* gombra kattintva megnyílik a Windows Intéző.

6. A Windows Intézőben csak mappák és *jar* fájlok jelennek meg, és csak *jar* fájlokat lehet kiválasztani.

7. A kiválasztott *jar* fájlok megjelennek a felületen.

8. Az *Apply* gombra kattintva a megtett változások mentésre kerülnek, és az ablak nyitva marad.

9a. Az OK gombra kattintva a megtett változások mentésre kerülnek, és az ablak bezáródik.

9b. A *Cancel* gombra kattintva a megtett változások el lesznek vetve, és az ablak bezáródik.

9c. A jobb felső sarokban lévő X-re kattintva a megtett változások el lesznek vetve, és az ablak bezáródik.

A lenyíló projekt menü felületi tesztje

1. Projekt megnyitásakor csak a legfelső mappa látszódik, aminek a neve a megegyezik a projektével.

2. Mappára való balkattintáskor az lenyílik, és megjelenik a tartalma.

3a. A gyökérmappára való jobbkattintáskor megjelenő menü tartalma a következő: *Rename Project, Create New File, Create New Folder, Close Project, Delete Project*.

3b. Egyéb mappára való jobbkattintáskor a megjelenő menü tartalma a következő: *Rename, Create New File, Create New Folder, Delete.*

4. Fájlra való balkattintáskar, az megnyílik a kódolási felületen.

5. Fájlra való jobbkattintáskor a megjelenő menü tartalma: *Rename, Set as Main File, Delete.*

6. Mappára való balkattintáskor, ha az már meg van nyitva, akkor bezáródik, és a tartalma eltűnik.

A Rename Project menüpont felületi tesztje

1. A menüpontra kattintáskor megjelenik a *Rename Project* ablak.

2. A *Project Name* mezőbe lehet írni.

3a. Az *Apply* gomb megnyomásakor, ha a *Project Name* mező üres, akkor egy erre figyelmeztető ablak jelenik meg.

3b. Az *Apply* gomb megnyomásakor, ha már létezik projekt a megadott névvel, akkor egy erre figyelmeztető ablak jelenik meg.

3c. Az *Apply* gomb megnyomásakor a projekt át lesz nevezve a megadott névre és az ablak bezáródik. Ez a lenyíló projekt menüben is megtörténik.

3d. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik.

A Create New File menüpont felületi tesztje

1. A menüpontra kattintáskor megjelenik a *Create New File* ablak.

2. *File Name* mezőbe lehet írni.

3a. Ha a kiterjesztést módosító lenyíló menüben nem a „.java” szerepel, akkor a *Class Preset* lenyíló menü nem látszódik.

3b. Ha a kiterjesztést módosító lenyíló menüben a „.java” szerepel, akkor a *Class Preset* lenyíló menü látszódik.

4a. A *Create* gomb megnyomásakor, ha a *File Name* mező üres, akkor egy erre

figyelmeztető ablak jelenik meg.

4b. A *Create* gomb megnyomásakor, ha a *File Name* mező üres, akkor egy erre figyelmeztető ablak jelenik meg.

4c. A *Create* gomb megnyomásakor, az új fájl létrejön a megfelelő mappában, a beállított névvel és kiterjesztéssel, java fájl esetén a kiválasztott sablon tartalommal. Az ablak bezáródik, és a létrejött fájl megjelenik a lenyíló projekt menüben, valamint megnyílik a kódolási felületen.

4d. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik.

A Create New Folder menüpont felületi tesztje

1. A menüpontra kattintáskor megjelenik a *Create Folder* ablak.

2. *Folder name* mezőbe lehet írni.

3a. A *Create* gomb megnyomásakor, ha a *Folder name* mező üres, akkor egy erre figyelmeztető ablak jelenik meg.

3b. A *Create* gomb megnyomásakor, ha már létezik mappa a megadott névvel, akkor egy erre figyelmeztető ablak jelenik meg.

3c. A *Create* gomb megnyomásakor, az új mappa létrejön a megfelelő mappában, a beállított névvel. Az ablak bezáródik, és a létrejött mappa megjelenik a lenyíló projekt menüben.

3d. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik.

A Close Project menüpont felületi tesztje

1. A *Close Project* menüpontra kattintva a megnyitott projekt bezáródik. A lenyíló projekt menü eltűnik, és megjelennek a helyén az *Open Files* és a *Recently Closed* menük. Ha voltak megnyitva fájlok, azok bezáródnak. A *Compile, Run* és *Settings* gombok inaktívvá válnak.

A Delete Project menüpont felületi tesztje

1. A *Delete Project* menüpontra kattintva megjelenik egy felugró ablak, ami megkéri a felhasználót, hogy erősítse meg a törlést.

2a. A felugró ablakon az OK gombra kattintva a felugró ablak bezáródik és a projekt

törlésre kerül. A lenyíló projekt menü eltűnik, és megjelennek a helyén az *Open Files* és a *Recently Closed* menük. Ha voltak megnyitva fájlok, azok bezáródnak. A *Compile, Run* és *Settings* gombok inaktívvá válnak.

2b. A felugró ablakon a *Cancel* gombra kattintva a felugró ablak bezáródik, és a törlés nem történik meg.

2c. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik, és a törlés nem történik meg.

A Rename menüpont felületi tesztje (mappánál)

1. A menüpontra kattintáskor megjelenik a *Rename Folder* ablak.

2. A *Folder Name* mezőbe lehet írni

3b. Az *Apply* gomb megnyomásakor, ha már létezik mappa a megadott névvel, akkor egy erre figyelmeztető ablak jelenik meg.

3c. Az *Apply* gomb megnyomásakor a mappa át lesz nevezve a megadott névre és az ablak bezáródik. Ez a lenyíló projekt menüben is megtörténik. Ha a mappa egy csomag, akkor őt tartalmazó fájlok csomagja is megváltozik.

3d. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik.

A Delete menüpont felületi tesztje (mappánál)

1. A *Delete* menüpontra kattintva megjelenik egy felugró ablak, ami megkéri a felhasználót, hogy erősítse meg a törlést.

2a. A felugró ablakon az OK gombra kattintva a felugró ablak bezáródik és a mappa törlésre kerül. A lenyíló projekt menüből eltűnik a törölt mappa. A mappának a fájljai is törlődnek. Ha volt megnyitva fájl a mappából, azok bezáródnak.

2b. A felugró ablakon a *Cancel* gombra kattintva a felugró ablak bezáródik, és a törlés nem történik meg.

2c. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik. és a törlés nem történik meg.

A Rename menüpont felületi tesztje (fájlnál)

1. A menüpontra kattintáskor megjelenik a *Rename File* ablak.

2. A *File Name* mezőbe lehet írni

3b. Az *Apply* gomb megnyomásakor, ha már létezik fájl a megadott névvel, akkor egy erre figyelmeztető ablak jelenik meg.

3c. Az *Apply* gomb megnyomásakor a fájl át lesz nevezve a megadott névre és az ablak bezáródik. Ez a lenyíló projekt menüben is megtörténik. Java fájl esetén az osztály neve, és a rá hivatkozó referenciák neve is megváltozik.

3d. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik.

A Delete menüpont felületi tesztje (fájlnál)

1. A *Delete* menüpontra kattintva megjelenik egy felugró ablak, ami megkéri a felhasználót, hogy erősítse meg a törlést.

2a. A felugró ablakon az OK gombra kattintva a felugró ablak bezáródik és a fájl törlésre kerül. A lenyíló projekt menüből eltűnik a törölt fájl. Ha afájl meg volt nyitva a kódolási felületen, akkor bezáródik.

2b. A felugró ablakon a *Cancel* gombra kattintva a felugró ablak bezáródik, és a törlés nem történik meg.

2c. A jobb felső sarokban lévő X-re kattintva az ablak bezáródik. és a törlés nem történik meg.

A Set as Main File felületi tesztje

1. A menüpontra kattintva megjelenik egy felugró ablak, ami megerősíti, hogy a kiválasztott fájl lett az új *main* fájl.

2. A futtatás és a fordítás az új *main* fájlt használja.

A Compile gomb felületi tesztje (projekten belül)

1a. Ha nincs beállítva *main* fájl, akkor inaktív, nem kattintható.

1b. Ha ki van beállítva *main* fájl, akkor kattintható.

2a. Kattintásra lefordítja a projektet. Ha sikeres, megjelenik egy felugró ablak, ami tájékoztatja a felhasználót a futtatás sikerességéről.

2b. Kattintásra lefordítja a projektet. Ha hiba lép fel, megnyílik egy Windows Parancssor a hibaüzenettel.

A Run gomb felületi tesztje (projekten belül)

1a. Ha nincs beállytva *main* fájl, akkor inaktív, nem kattintható.

1b. Ha ki van beállítva *main* fájl, akkor kattintható.

2. Kattintásra lefordítja és futtatja a kiválasztott futási konfiguráció alapján a projektet, majd ennek eredményével megjelenít egy Windows Parancssort.

Az Open Command Line gomb felületi tesztje (projekten belül)

1. Kattintáskor megnyílik egy Windows Parancssor a projekt forrásmappájában.

A Futási konfiguráció választó felületi tesztje (projekten kívül)

1. Alapjáraton a projekt nevével megegyező futási konfiguráció van kiválasztva.

2. Kattintásra felsorolja a projekthez beállított futási konfigurációkat.

3. A *Compile* és *Run* gombok az aktuálisan kiválasztott futási konfigurációt használják.

4. Új futási konfiguráció felvételekor az megjelenik a leugró menüben.

5. Futási konfiguráció eltávolítása után az eltűnik a leugró menüből.

A Settings ablak felületi tesztje (projekten belül)

1a. Ha van kiválasztott *main* fájl, akkor kattintásra megnyílik a *Settings* ablak.

1b. Ha nincs kiválasztott *main* fájl, akkor a *Settings* gomb nem kattintható.

2. A *Settings* ablakban az *Add Dependencies, Add Run-Configuration* és *Edit Configuration* fülek is kattintható a felső menüsorban.

3. A *Select jar dependencies* alatti területen megjelennek a hozzáadott *jar* függőségek.

4. Ha a *jar* függőség neve mellett található mínusz gombra kattintunk a függőség eltűnik a listából.

5. A *Browse…* gombra kattintva megnyílik a Windows Intéző.

6. A Windows Intézőben csak mappák és *jar* fájlok jelennek meg, és csak *jar* fájlokat lehet kiválasztani.

7. A kiválasztott *jar* fájlok megjelennek a felületen.

8. Az *Add Run-Configuration* fül alatt a *Name* és a *Command* mezőkbe lehet írni.

9a. Az *Add* gombra kattintva, ha valamelyik mező üres, akkor megjelenik egy felugró ablak, ami erre figyelmezteti a felhasználót.

9b. Az *Add* gombra kattintva, ha már van a megadott névvel rendelkező futási konfiguráció, akkor megjelenik egy felugró ablak, ami erre figyelmezteti a felhasználót.

9c. Az *Add* gombra kattintva, ha nincs a megadott névvel rendelkező futási konfiguráció akkor ez felvételre kerül a futási konfigurációk közé. Erről egy felugró ablak tájékoztatja a felhasználót, és a mezők kiüresednek.

10. Az *Edit Run-Configuration* fül alatt a *Name* ás *Command* mezőkbe lehet írni.

11. A *Choose run config* lenyíló menüben jelen van (az elepértelmezett futási konfiguráción kívül) az összes létrehozott futási konfiguráció.

12. A kiválasztott futási konfiguráció adatai megjelennek a hozzájuk tartozó mezőkben.

13a. A *Save* gombra kattintva, ha valamelyik mező üres, akkor megjelenik egy felugró ablak, ami erre figyelmezteti a felhasználót.

13b. A *Save* gombra kattintva, ha már van a megadott névvel rendelkező futási konfiguráció, akkor megjelenik egy felugró ablak, ami erre figyelmezteti a felhasználót.

13c. A *Save* gombra kattintva, ha nincs a megadott névvel rendelkező futási konfiguráció akkor a kiválasztott futási konfiguráció adati módosulnak. Erről egy felugró ablak tájékoztatja a felhasználót.

13d. A *Save* gombra kattintva, ha nincs kiválasztott futási konfiguráció, akkor megjelenik egy felugró ablak, ami erről tájékoztatja a felhasználót.

14a. A *Delete* gombra kattintva a kiválasztott futási konfiguráció törlődik. Errőr egy felugró ablak tájékoztatja a felhasználót.

15. Az *Apply* gombra kattintva a megtett változások mentésre kerülnek, és az ablak nyitva marad.

16a. Az OK gombra kattintva a megtett változások mentésre kerülnek, és az ablak bezáródik.

16b. A *Cancel* gombra kattintva a megtett változások el lesznek vetve, és az ablak bezáródik.

16c. A jobb felső sarokban lévő X-re kattintva a megtett változások el lesznek vetve, és az ablak bezáródik.

A Kódolási felület felületi tesztje

1a. Ha java fájl van megnyitva, akkor a különböző szintaktikai elemek ki vannak színezve: a Java kulcsszavai kék, a primitív típusok türkíz, az annotációk barna, a szövegek zöld, a kommentek szürke és az egyéb szavak feketével vannak kiszínezve.

1b. Ha nem java fájl van megnyitva, akkor nincs színezés, a karakterek feketék.

2. A fenti menüben megjelennek a megnyitott fájlok, az éppen aktív szürkével kiemelve.

3. Ha több fájl van megnyitva, akkor köztük váltani kattintással lehet.

4. Ha *drag-and-drop*-pal a felső fület behúzzuk a kódolási felületre, akkor az vizuálisan jelzi, hogy hol lesz megjelenítve a fájl.

5a. Ha a fájl a saját „felére” lesz behúzva, akkor ott fog megjelenni.

5b. Ha a fájl a másik „félre” lesz húzva, akkor ott fog megjelenni. Ha a másik fél még nincs megnyitva, akkor a felület kettéoszlik.

6. Ha egy fájlt bezárunk a fülön lévő apró X-re kattintva, akkor a fájl bezáródik, és a tőle jobbra lévő fájl kerül megnyitásra. Ha nincs tőle jobbra fájl, akkor tőle balra lévő nyílik meg.

Végszó

Célom egy olyan IDE készítése volt, amely megkönnyíti a programozás folyamatát Java nyelven, kis terjedelmű projekteknél. A hangsúly egy koncepció kidolgozásán, és ennek a megvalósításán, mintsem egy teljesen forradalmi, soha nem látott fejlesztői környezet elkészítésén volt, amelynek teljesen egyedi funkciói vannak. A szakdolgozat fejlesztésének ideje alatt elkészült egy stabil keret, amit ezek után fejleszteni, kiegészíteni lehet, még hiányzó, majd teljesen egyedi funkciókkal.

A program elkészítése számos kihívás elé állított. Ilyen volt például a Spring és a JavaFX közös működése, vagy magának a kódfelületnek a létrehozása. Az utóbbiba sok időt öltem bele, egy teljesen saját felület létrehozására, de végül jobbnak láttam egy 3. féltől származó API használatát.

A témabejelentőben felvázolt program elkészült, egy funkció híján. A verziókezelés kezdetleges formáját a program többi funkciójának teljessége érdekében kivezettem, és az ezáltal nyert időt inkább QA-ra és hibák javítására szántam.

A szakdolgozat eredménye számomra kevésbé nyilvánul meg a konkrét elkészített programban, hanem inkább a megismert technológiákban, folyamatokban, illetve abban, hogy megadott idő alatt reálisan mennyit tudok elkészíteni. Ebből a szempontból a szakdolgozatot átfogóan sikeresnek találom.

Irodalomjegyzék

[1] - https://projectlombok.org/ - 2022.05.29

[2] - https://www.jetbrains.com/idea/ - 2022.05.29

[3] - https://github.com/fangyidong/json-simple - 2022.05.29

[4] - https://docs.oracle.com/javase/8/docs/api/javax/annotation - 2022.05.29

[5] - https://guava.dev/ - 2022.05.29

[6] - https://maven.apache.org/ - 2022.05.29

[7] - https://github.com/FarkaschZoltan/BaristaIDE - 2022.05.29

[8] - https://openjfx.io/ - 2022.05.29

[9] - https://spring.io/projects/spring-framework - 2022.05.29

[10] - https://spring.io/projects/spring-boot#overview - 2022.05.29

[11] - https://codemirror.net/ - 2022.05.29

[12] - https://junit.org/junit4/ - 2022.05.29

[13] - https://site.mockito.org/ - 2022.05.29

[14] - http://testfx.github.io/TestFX/ - 2022.05.29

[15] - https://assertj.github.io/doc/ - 2022.05.29

Az UML diagrammokat és kapcsolati ábrákat a class-visualizer programmal készítettem:

http://www.class-visualizer.net/ - 2022.05.29