1. Web service házi feladat

Dr. Simon Balázs (sbalazs@iit.bme.hu), BME IIT, 2023.

A továbbiakban a NEPTUN szó helyére a saját Neptun-kódot kell behelyettesíteni, csupa nagybetűkkel.

1 A feladat leírása

A feladat egy mozi jegyfoglalási rendszerének elkészítése, és ennek webszolgáltatásként való publikálása.

A szolgáltatás interfészét az alábbi pszeudo-kód írja le:

```
[Uri("http://www.iit.bme.hu/soi/hw/SeatReservation")]
namespace SeatReservation
{
    exception CinemaException
    {
        int ErrorCode;
        string ErrorMessage;
    }

    struct Seat
    {
        string Row;
        string Column;
    }

    enum SeatStatus
    {
        Free,
        Locked,
        Reserved,
        Sold
    }

    interface ICinema
    {
        void Init(int rows, int columns) throws CinemaException;
        Seat[] GetAllSeats()throws CinemaException;
        SeatStatus GetSeatStatus(Seat seat) throws CinemaException;
        string Lock(Seat seat, int count) throws CinemaException;
        void Unlock(string lockId) throws CinemaException;
        void Reserve(string lockId) throws CinemaException;
        void Buy(string lockId) throws CinemaException;
    }
}
```

A fenti pszeudo-kód csak a könnyebb áttekinthetőséget szolgálja. A valódi interfészt a csatolt WSDL és XSD fájl írja le, ezekből kiindulva kell implementálni a szervert és a klienst is.

Az egyes függvények feladata a következő:

- Init:
  - inicializálja a mozitermet a megadott számú sorokkal és oszlopokkal
  - a sorok száma: 1 <= rows <= 26
  - az oszlopok száma: 1 <= columns <= 100
  - minden szék szabad, és minden korábbi zárolás, foglalás törlődik
  - ha a sorok vagy oszlopok száma kívül esik a fent megadott tartományon, akkor CinemaException-t kell dobni
- GetAllSeats:
  - visszaadja a moziteremben lévő székeket
  - a sorokat az angol ABC nagy betűi jelölik 'A'-tól kezdve sorfolytonosan
  - az oszlopokat egész számok jelölik, 1-től kezdve sorfolytonosan
- GetSeatStatus:
  - megadja, hogy mi az adott helyen lévő szék státusza (szabad, zárolt, foglalt vagy eladva)
  - ha a megadott szék pozíciója hibás, akkor CinemaException-t kell dobni
- Lock:
  - az adott széktől kezdve az adott sorban count darab folytonosan egybefüggő székeket zárol előrefelé számolva
  - ha a zárolás valamilyen okból nem teljesíthető (pl. nincs annyi maradék szék a sorban, vagy az egybefüggő széksorozatból nem mindegyik szék szabad), akkor CinemaException-t kell dobni, és nem szabad semmit zárolni
  - a függvénynek vissza kell adnia egy egyedi azonosítót, ami alapján vissza tudja keresni a zárolt székeket
- Unlock:
  - feloldja az adott azonosítójú zárolást, és minden a zároláshoz tartozó szék szabad lesz
  - ha nincs ilyen azonosítójú zárolás, akkor CinemaException-t kell dobni
  - az Unlock foglalást nem old fel, csak zárolást
- Reserve:
  - lefoglalja az adott azonosítójú zárolást, és minden a zároláshoz tartozó szék foglalt lesz
  - ha nincs ilyen azonosítójú zárolás, akkor CinemaException-t kell dobni
- Buy:
  - eladja az adott azonosítójú zárolást (akkor is, ha már foglalt állapotban vannak a székek), és minden a zároláshoz tartozó szék eladott lesz
  - ha nincs ilyen azonosítójú zárolás, akkor CinemaException-t kell dobni

2 A szolgáltatás

A szolgáltatást egy Maven vagy egy .NET Core webalkalmazásban kell megvalósítani. A két technológia közül szabadon lehet választani.

A webalkalmazás neve a következő legyen: **WebService_NEPTUN**

A szolgáltatást a következő URL-en kell publikálni:

`http://localhost:8080/WebService_NEPTUN/Cinema`

A szolgáltatásnak adatokat kell tárolnia az egyes hívások között. Egy valódi alkalmazás esetén az adatok tárolására adatbázist kéne használni. A házi feladatban a könnyebbség kedvéért *statikus változókban* tároljuk a szükséges adatokat!

2.1 Maven választása esetén

A szolgáltatást egy ugyanolyan Maven webalkalmazásban kell megvalósítani, mint amilyen a tutorialban szerepel. A **pom.xml**-nek kötelezően a csatolt **service/pom.xml**-nek kell lennie, de a NEPTUN szót le kell cserélni benne a saját Neptun-kódra.

A **SeatReservation.wsdl** és **SeatReservation.xsd** fájlok az **src/main/webapp/WEB-INF/wsdl** könyvtárba kerüljenek, ahogy a tutorialban is történt.

A forrásfájlok készülhetnek Java vagy Kotlin nyelven is.

2.2 .NET Core választása esetén

A szolgáltatást egy ugyanolyan ASP.NET Core webalkalmazásban kell megvalósítani, mint amilyen a tutorialban szerepel. A projektfájlnak kötelezően a csatolt **service/WebService_NEPTUN.csproj**-nak kell lennie, de a fájl nevében a NEPTUN szót le kell cserélni a saját Neptun-kódra.

A **SeatReservation.wsdl** és **SeatReservation.xsd** fájlok a projekten belül a **wsdl** könyvtárba kerüljenek, ahogy a tutorialban is történt.

3 A kliens

A klienst egy Maven vagy egy .NET Core konzol alkalmazásban kell megvalósítani. A két technológia közül szabadon lehet választani.

Az alkalmazás neve a következő: **WebServiceClient_NEPTUN**

A kliens négy parancssori argumentumot kap: **`[url] [row] [column] [task]`**

Az argumentumok jelentése a következő:

- [url]: a meghívandó szolgáltatás URL-je (nem feltétlenül csak a saját szolgáltatással lesz tesztelve)
- [row]: a szék sorának azonosítója

- [column]: a szék oszlopának azonosítója
- [task]: hogy mit kell csinálni a székkel

A [task] lehetséges értékei:

- Lock: zárolni kell a széket – pontosan egy darab Lock() hívás a szerver felé
- Reserve: zárolni kell, majd le is kell foglalni a helyet – Lock()+Reserve() hívás a szerver felé
- Buy: zárolni kell, majd meg is kell vásárolni helyet – Lock()+Buy() hívás a szerver felé

Ez a kliens mindig pontosan 1 db széket kezel. A program csak a Lock(), Reserve() és Buy() függvényeket hívhatja a fent megadott sorrendben! A többi függvényt tilos meghívnia! (Más egyéb kliens programot lehet írni tesztelés céljából, de azt nem kell beadni.)

3.1 Maven választása esetén

A klienst egy ugyanolyan Maven konzol alkalmazásban kell megvalósítani, mint amilyen a tutorialban szerepel. A **pom.xml**-nek kötelezően a csatolt **client/pom.xml**-nek kell lennie, de a NEPTUN szót le kell cserélni benne a saját Neptun-kódra.

A **SeatReservation.wsdl** és **SeatReservation.xsd** fájlok az **src/main/resources/wsdl** könyvtárba kerüljenek, ahogy a tutorialban is történt.

Az alkalmazáson belül a következő osztály tartalmazza a **main()** függvényt: **cinema.Program**

A forrásfájlok készülhetnek Java vagy Kotlin nyelven is.

3.2 .NET Core választása esetén

A szolgáltatást egy ugyanolyan .NET Core konzol alkalmazásban kell megvalósítani, mint amilyen a tutorialban szerepel. A projektfájlnak kötelezően a csatolt **client/WebServiceClient_NEPTUN.csproj**-nak kell lennie, de a fájl nevében a NEPTUN szót le kell cserélni a saját Neptun-kódra.

A **SeatReservation.wsdl** és **SeatReservation.xsd** fájlok a projekten belül a **wsdl** könyvtárba kerüljenek, ahogy a tutorialban is történt.

4 Beadandók

Beadandó egyetlen ZIP fájl. Más tömörítési eljárás használata tilos!

A ZIP fájl gyökerében két könyvtárnak kell lennie, az egyik a szolgáltatás, a másik a kliens alkalmazása:

- **WebService_NEPTUN:** a szolgáltatás Maven vagy .NET Core projektje teljes egészében
- **WebServiceClient_NEPTUN:** a kliens Maven vagy .NET Core projektje teljes egészében

A szolgáltatásnak a csatolt **wsdl/SeatReservation.wsdl** és **wsdl/SeatReservation.xsd** fájlokban specifikált interfészt kell implementálnia. Az interfész és a generált fájlok megváltoztatása tilos! A kliensnek bármely ilyen interfésznek megfelelő szolgáltatást meg kell tudnia hívni!

A megoldásnak a telepítési leírásban meghatározott környezetben kell fordulnia és futnia, a **pom.xml**-ek illetve a **csproj** fájlok tartalmát a Neptun-kód módosításán felül megváltoztatni tilos!

Fontos: a dokumentumban szereplő elnevezéseket és kikötéseket pontosan be kell tartani! Még egyszer kiemelve: a NEPTUN szó helyére mindig a saját Neptun-kódot kell behelyettesíteni, csupa nagybetűkkel!

4.1 Maven választása esetén

Mielőtt a projekteket becsomagolnánk a ZIP-be, mindkét projekten belül adjuk ki az **mvn clean** parancsot! Ez a parancs törli a **target** alkönyvtárat, amely elég nagy is lehet és egyébként sem lesz figyelembe véve a kiértékelés során.

A ZIP kibontása után a szolgáltatásnak parancssorból fordulnia kell a következő parancs kiadásával:

```
...\WebService_NEPTUN>mvn package
```

A szolgáltatásnak parancssorból települnie kell a szerverre következő parancs kiadásával:

```
...\WebService_NEPTUN>mvn wildfly:deploy
```

A kliensnek parancssorból fordulnia kell a következő parancs kiadásával:

```
...\WebServiceClient_NEPTUN>mvn package
```

A kliensnek parancssorból futnia kell a következő parancs kiadásával (az **exec.args** értéke csak egy példa):

```
...\WebServiceClient_NEPTUN>mvn exec:java -Dexec.mainClass=cinema.Program
            -Dexec.args="http://localhost:8080/WebService_NEPTUN/Cinema H 31 Reserve"
```

4.2 .NET Core választása esetén

Mielőtt a projekteket becsomagolnánk a ZIP-be, mindkét projekten belül adjuk ki a **dotnet clean** parancsot! Ez a parancs törli a **bin** és **obj** alkönyvtárakat, amelyek elég nagyok is lehetnek és egyébként sem lesznek figyelembe véve a kiértékelés során.

A ZIP kibontása után a szolgáltatásnak parancssorból fordulnia kell a következő parancs kiadásával:

```
...\WebService_NEPTUN>dotnet build
```

A szolgáltatásnak parancssorból futnia kell a következő parancs kiadásával:

```
...\WebService_NEPTUN>dotnet run
```

A kliensnek parancssorból fordulnia kell a következő parancs kiadásával:

```
...\WebServiceClient_NEPTUN>dotnet build
```

A kliensnek parancssorból futnia kell a következő parancs kiadásával (az utolsó négy paraméter értéke csak egy példa):

```
...\WebServiceClient_NEPTUN>dotnet run http://localhost:8080/WebService_NEPTUN/Cinema H 31
                                       Reserve
```