A feladatban a `rail.Railway` és `rail.RailMap` osztályokat fogjuk több lépésben megvalósítani.

Legyen minden nyilvános, kivéve ott, ahol a feladat mást határoz meg.

A városokat `String`-ként, a távolságot `int`-ként ábrázoljuk.

1. rész (5 + 1 pont)

Készítsük el a `rail.Railway` osztályt. Szerepeljenek benne az alábbi rejtett adattagok.

- Két `String` városnév (továbbiakban röviden csak város) és egy `int` távolság.

Valósítsuk meg továbbá az alábbi nyilvános metódusokat.

- Egy konstruktor, mely paraméterül vár két várost és egy távolságot, ezekkel inicializálja az adattagokat.

- Egy paraméter nélküli `getCities()`, mely egy (kételemű) tömbben adja vissza a két várost. A sorrend tetszőleges.

- Egy paraméter nélküli `getDistance()`, mely visszaadja a távolságot.

Tesztelő: `tests.Part1`

2. rész (5 + 1 pont)

A `rail.Railway` osztályba vegyük fel az alábbi nyilvános konstanst és metódusokat.

- Egy osztályszintű, `Railway` típusú konstanst `KESZTHELY_BUDAPEST` névvel. A két város legyen Keszthely és Budapest, a távolság 190.

- Egy osztályszintű `make()` metódust, mely egy `String`-et vár, és egy `Railway`-t ad vissza. A helyes paraméter két város és egy távolság szóközzel, például: `Budapest Eger 140`. A `make()` vizsgálja meg, hogy megvan-e mindhárom rész, a két városnév nem üres szöveg, és a harmadik rész valóban egy szám.

  Érdemes megnézni a String.split és a Character.isDigit metódusokat,

- Egy objektumszintű `hasEnd()` metódust, mely egy várost vár és egy logikai értéket ad vissza. Ha a paraméter város megegyezik valamelyik valamelyik város adattaggal, akkor igazat ad a metódus, különben hamisat.

- Egy objektumszintű `getOtherCity()` metódust, mely paraméterül is vár egy várost és vissza is ad egy várost. Ha a paraméter megegyezik a két város adattag közül valamelyikkel, akkor a másikat adja vissza a metódus. Ha a paraméter egyik adattaggal sem egyezik, akkor az eredmény `null`.

  Például Keszthely-Budapest viszonylatban a `getOtherCity("Budapest")` eredménye `"Keszthely"`, míg a `getOtherCity("Debrecen")` eredménye `null`.

Tesztelő: `tests.Part2`

3. rész (5 + 1 pont)

Valósítsuk meg a `rail.RailMap` osztályt is, mely egy ország vasúthálózatát ábrázolja. Legyenek benne az alábbi rejtett adattagok.

- Egy `String` országnév és egy `Railway` objektumokat tároló tömb.

Továbbá legyenek az osztályban az alábbi nyilvános metódusok.

- Egy konstruktor, mely paraméterül vár egy `String` országnevet és egy `String[]` vasútvonal tömböt. A konstruktor inicializálja az adattagokat és beolvassa a `város1 város2 távolság` szövegekként megadott vasútvonalakat. Ügyeljünk, hogy akkor se kerüljön `null` a `Railway` tömbbe, ha egy vasútvonal érvénytelen.

- Egy paraméter nélküli `getCities()` metódust, egy `String`-ekből álló tömbben adja vissza az összes várost. Ügyeljünk, hogy minden város csak egyszer szerepeljen a listában.

Tesztelő: `tests.Part3`

4. rész (5 + 1 pont)

- Bővítsük ki a `rail.Railway` osztályt egy nyilvános `toString()` metódussal. Példaként a Keszthely-Budapest viszonylatban az alábbi szöveget adja vissza: `Railway(Keszthely - Budapest 190)`.

- Bővítsük ki a `rail.RailMap` osztályt egy nyilvános `toString()` metódussal, mely a fentihez hasonlóan az országnevet és a vasútvonalakat adja vissza. Például `RailMap(Magyarorszag,[Railway(Keszthely - Budapest 190), Railway(Budapest - Salakszentmotoros 40)])`.

- Bővítsük ki a `rail.RailMap` osztályt egy nyilvános `getNeighbours()` metódussal, mely egy

várost vár paraméterül, és ennek városnak a szomszédos városait adja vissza egy láncolt listában.

Tesztelő: `tests.Part4`

5. rész (5 + 1 pont)

- Bővítsük ki a `rail.RailMap` osztályt egy paraméter nélküli, `String` visszatérési típusú `capitalCity()` metódussal, mely visszaadja azt a várost, melynek legtöbb szomszédja van.

Tesztelő: `tests.Part5`