Zadaci za vježbu iz kolekcija

1. Napišite klasu `Ladder` koja će služiti za evidentiranje pozicija u nekoj igri pri čemu svaka igra između 2 igrača završava pobjedom jednog od igrača. Igrači su predstavljeni stringovima. Ovisno o rezultatu i poziciji na ljestvici nakon svake igre se mijenja poredak na sljedeći način:

    - Poraženi igrač se spušta za jednu poziciju niže na ljestvici
    - Ako je pobjednik bio više rangiran od poraženog, tada se pobjednik pomiče za jednu poziciju više
    - Ako je pobjednik bio iza poraženog, tada se njegova pozicija mijenja za pola udaljenosti od poraženog. Primjerice, ako je pobjednik bio 11., a poraženi 4., tada se pobjednik treba pomaknuti za (11-4)/2 (tj. 3 mjesta), pa su nove pozicije 8, odnosno 5.

    Igrači se u igru mogu dodati u konstruktoru ili naknadno pozivom metode `join`. U oba slučaju moguće je dodati varijabilni broj igrača pri čemu prilikom dodavanja treba provjeriti je li igrač već bio dodan te izbjeći dupliciranje igrača. Klasa treba imati

    - metodu `count` koja vraća broj igrača
    - metodu `standings` kojom će se omogućiti iteriranje po igračima u poretku koji odgovara pozicijama u igri
    - metodu `public void gameFinished(String winner, String looser).`

2. Definirati generičku klasu `Pair` za pohranu para vrijednosti istog tipa. Par se inicijalizira konstruktorom i ne može se naknadno mijenjati.

    Napisati klasu `LadderUtil` sa statičkom metodom

    ```
    Iterable<Pair<String>> randomDraw(Ladder ladder)
    ```

    koja će nasumično generirati parove za sljedeći krug igre. U slučaju neparnog broja igrača, za igrača koji je slobodan u tom krugu napraviti par oblika *(ime igrača, "FREE").*
    Uputa: Upotrijebiti metodu `shuffle` iz klase `Collections`.

3. Proširite prethodni primjer i napišite klasu `LadderEnhanced` tako da se za svakog igrača evidentira podatak o broju pobjeda i poraza. Rješenje izvedite nasljeđivanjem klase `Ladder`.
    Klasa mora imati metode `wins` i `losses` koje vraćaju broj pobjeda, odnosno poraza nekog igrača.

4. Napisati klasu `LadderEnhancedByLossesOrder` izvedenu iz `LadderEnhanced` tako da ima metodu `Iterable<String> orderByLosses()` koja će omogućiti iteriranje igrača ne po poretku na tablici, nego po broju poraza. U slučaju jednakog broja poraza, međusobni poredak igrača je nebitan.

5. Definirati klasu `Track` koja predstavlja pjesmu. Svaka pjesma sastoji se od naslova pjesme, naziva izvođača i trajanja u sekundama. Svi argumenti se postavljaju koristeći argumente konstruktora i nepromjenjivi su. Nadjačati metodu `toString` tako da vraća s*tring* u obliku *naziv izvođača: naslov pjesme* što predstavlja puni naziv pjesme.
   Definirati klasu `Playlist` koja predstavlja popis za reprodukciju. Popis ima svoje ime i kolekciju pjesama, te može biti ograničen trajanjem u sekundama. Opcionalno ograničenje se postavlja ovisno o korištenom konstruktoru i ne može se naknadno dodati ili ukloniti. Za pohranu informacije o ograničenju trajanja upotrijebiti klasu `OptionalInt`. Ime za popis se zadaje konstruktorom, ali se može naknadno promijeniti.
   Za razmisliti: Hoćemo li za kolekciju u kojoj pohranjujemo pjesme definirati getter i setter?
   Pjesme je moguće dodavati (`add`) u popis pri čemu se ista pjesma može dodati više puta. Ako u popisu više nema mjesta zbog ograničenja trajanja, ova metoda mora vratiti *false*. Klasa mora omogućiti dohvat pjesme na određenoj poziciji (`trackAt`), uklanjanje pjesme s određene pozicije (`deleteAt`) ili pomak pjesme na određenoj poziciji prema naprijed ili natrag (`move`) ovisno o drugom argumentu metode (pozitivna vrijednost pomiče pjesmu prema kraju popisa, negativna prema naprijed). Pozicije su numerirane tako da je prva pjesma na poziciji 1 te nije potrebno provjeravati je li korisnik zadao ispravne pozicije za dohvat, uklanjanje i pomak. Dodatno, u klasi `Playlist` definirati metode koje vraćaju veličinu (`count`) i ukupno trajanje popisa (`duration`) te metodu `invert` koja će obrnuti postojeći popis za reprodukciju.

6. Napisati klasu `PlaylistUtil` i nekoliko metoda koje primaju varijabilni broj popisa za reprodukciju, a vraćaju redom
   - metoda `words`: skup svih riječi koje se nalaze u nazivima pjesama.
   - metoda `wordsOccurance`: informacije o broju pojavljivanja pojedinih riječi iz naziva pjesama
   - metoda `perLength`: za svaku moguću duljinu riječi, evidentirati riječi te duljine i broj njihovih pojavljivanja

7. Napisati klasu `SortedPlaylist` izvedenu iz klase `Playlist` u kojoj nije moguće mijenjati poziciju neke pjesme (metoda treba baciti neprovjeravanu iznimku), a dodavanje se vrši umetanjem u popis tako da pjesme budu složene po trajanju pjesme ovisno o aktivnom sortiranju. Inicijalno su složene ulazno, ali u slučaju invertiranja poredak je silazan. Sljedeće invertiranje vraća popis uzlazno itd…

8. Napisati klasu `Polynomial` koja omogućuje zbrajanje polinoma. Polinom se inicijalizira parnim brojem varijabilnih argumenata gdje se redom izmjenjuju koeficijenti i potencije (vidi primjer). Klasa mora imati metodu `plus` za sumu polinom. Nadjačati metodu `toString()` tako da se polinom ispiše od najveće do najmanje potencije. Obratiti pažnju na negativne brojeve, koeficijent 1 (ne ispisuje se osim ako se ne radi o slobodnom članu) i potenciju 1.

```
Polynomial a = new Polynomial(-2, 4, 2, 2, 3, 1, 5, 0);
System.out.println(a); // -2x^4+2x^2+3x+5

Polynomial b = new Polynomial(-2, 1, -3, 6, 1, 3, -2, 2);
System.out.println(b); // -3x^6+x^3-2x^2-2x

Polynomial c = a.plus(b);
System.out.println(c); // -3x^6-2x^4+x^3+x+5
```

Rješenje zadataka dostupna su na

1. https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t01/Ladder.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t01/Main.java
2. https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t02/Pair.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t02/LadderUtil.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t02/Main.java
3. https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t03/PlayerInfo.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t03/LadderEnhanced.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t03/Main.java
4. https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t04/LadderEnhancedByLossesOrder.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t04/Main.java
5. https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t05/Track.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t05/Playlist.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t05/PlaylistDataLoader.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t05/Main.java
6. https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t06/PlaylistUtil.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t06/Main.java
7. https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t07/SortedPlaylist.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t07/Main.java
8. https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t08/Polynomial.java
   https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-09/src/main/java/hr/fer/oop/homework_09/t08/Main.java

Svaki od zadataka ima već pripremljen kostur glavnog programa za demonstraciju pojedinog primjera, ali ne sadrži sve primjere kojim se provjerava ispravnost rješenja te se stoga preporuča dopuniti ih po potrebi.