Zadaci za vježbu iz teme 11 (Ugniježđene i anonimne klase. Lambda izrazi.)

1. Krenite od klase `ImmutableContainer` koja predstavlja jednostavni nepromjenjivi spremnik za cijele brojeve koji se spremaju u polje:

```
public class ImmutableContainer {
	private Integer[] numbers;
	public ImmutableContainer (Integer... num) {
		numbers = new Integer[num.length];
		for(int i = 0; i<num.length; i++)
			numbers[i] = num[i];
	}
}
```

Klasu proširite na način da ona postane iterabilna, tako da se može koristit i u for-each petljama. Do spremljenih brojeva moći će se doći samo preko iteratora. Klasu iteratora ugradite u `ImmutableContainer` kao:
a) static nested klasu (iterira s kraja prema početku uz preskok - „svaki drugi“),
b) inner klasu (klasični iterator)
c) lokalnu (imenovanu) klasu (klasični iterator)
d) anonimnu klasu (iterira s kraja prema početku)
Za svaku inačicu klase `ImmutableContainer` napišite aplikaciju u kojoj isprobajte pristup brojevima. Pogledajte i rezultate prevođenja.

2. Napišite klasu **`Airplane`** koja ima privatne atribute: `String name` za spremanje naziva avio kompanije kojoj avion pripada, `int currentSpeed` koji bilježi trenutnu brzinu (inicijalno 0), listu tereta `List<Cargo> cargo` (inicijalno prazna), `String from` i `String to` (imena početnog i krajnjeg aerodroma; inicijalizira kapetan) . Klasa ima javni konstruktor za inicijalizaciju atributa `name` te javnu metodu `addCargo` za dodavanje tereta u listu. Klasa ima i privatne metode: `void increaseSpeed(int increment)` za povećanje brzine, `int getOverallCargoWeight()` koja vraća ukupnu težinu tereta te `void flyAround(Pair<Double, Double> ... airportsCoord)` koja prima varargs parova koji sadrže zemljopisnu širinu i dužinu aerodroma od kojih je prvi početni, a ostali su po redu oni koje avion posjećuje tijekom leta. Metoda ispisuje poruku o udaljenosti između početnog i krajnjeg aerodroma. (`Pair` je klasa koju ste koristili u zadacima iz teme 10.) Unutar metode morate definirati lokalnu finalnu klasu **`Distance`** čiji konstruktor ima dva argumenta tipa Pair<Double, Double> (širine i dužine dva aerodroma) i u njemu se na temelju haversine formule (vidi Google) računa udaljenost između aerodroma, a vrijednost se upisuje u vlastiti atribut `double distance`. Klasa `Distance` ima getter za atribut `distance`.
Klasa **`Captain`** koju trebate napisati ima privatni atribut `String name` i javni konstruktor za njegovu inicijalizaciju. Također, klasa ima javnu metodu `void sayWelcomeAndPilot(String from, String to)` unutar koje kapetan inicijalizira vrijednosti istoimenih atributa aviona, check-ira teret, povećava brzinu aviona, predstavlja se, želi dobrodošlicu i daje osnovne informacije o letu.
Klasa **`Cargo`** koju također trebate napisati ima privatne atribute `String type` i `int weight`, konstruktor za inicijalizaciju tih atributa te privatnu metodu `void checkCargo()`. Napomena: objekti iz klase `Cargo` pasivno postoje u avionu i nemaju nikakvu interakciju s avionom.
Napomena: pristup privatnim atributima i metodama objekta iz klase `Airplane` i `Cargo` ima samo kapetan. Jedino on kontrolira stanje aviona (ima pregled nad vrijednostima atributa) i upravlja njime pokretanjem metoda. Kako bi zadovoljili navedene zahtjeve vezane uz pristup, kako treba implementirati klase `Captain` i `Cargo`?
Napišite aplikaciju u klasi `Main` u kojoj se kreira avion Croatia Airlines-a, pridružuje mu se kapetan Sully, ukrcava se teret (50 kg pošte i 500 kg prtljage) i od kapetana traži da započne let između Rijeke i Venecije. Aplikacija treba dati sljedeći odziv:

```
mail, 50 kg, checked.
luggage, 500 kg, checked.
This is your captain Sully speaking.
The Croatia Airlines plane on flight from Rijeka to Venice is loaded with 550 kg.
Enjoy your flight. Currently, our speed is 200
We have approx  176,27 km to our end destination.
```

3. Napišite klasu `Person` s atributima `int id`, `String name`, `String surname`, `LocalDate birthday` i `Gender gender`, gdje je `Gender` enumeracija s vrijednostima `MALE` i `FEMALE`. Klasa ima konstruktor za inicijalizaciju svih atributa, getter-e, metodu `int getAge()` koja iz datuma rođenja računa dob koju vraća, metodu `toString` te statičku metodu `List<Person> loadPersons()` koja vraća listu osoba. Klasa neka pripada posebnom paketu jer će vam trebati i u drugim zadacima.
U aplikaciji (klasa `Main`) želimo ispisati podatke o osobama koje generira metoda `loadPersons`, ali u različitim formatima. Najprije po formatu koji obuhvaća sve podatke: *ime prezime god-mj-dan spol dob*, zatim u formatu: *ime PREZIME dob*, zatim bi željeli ispisivati podatke uvjetno, za osobe mlađe od 55 godina, po formatu: *ime spol dob*, … .
Za realizaciju svakog ispisa trebate definirati posebnu statičku metodu u klasi `Main`. Da li postoji efikasniji pristup?

4. Rješenje prethodnog problema je definiranje takve metode za ispis u koju će se preko argumenata, pored informacije o osobama čije podatke treba ispisati, unijeti i metoda (kod) za formatiranje ispisa. Napišite definiciju sučelja `PersonDataFormatter` koje predstavlja apstraktnu funkcionalnost formatiranja – sadrži apstraktnu metodu `String format(Person p)`, a koje neka bude dio paketa kojem pripada klasa `Person`. Također, u sučelje ugradite i default metodu za ispis osobe `default void print(Person p)` po formatu kojeg definira klasa koja implementira sučelje, te statičku metodu `static void printAll(Iterable<Person> persons, PersonDataFormatter formatter)` koja predstavlja željeno, efikasnije rješenje za ispis osoba po odabranom formatu. U aplikaciji (klasa `Main`) pokažite kako se sučelje `PersonDataFormatter` može implementirati: klasom, anonimnom klasom, ali i lambda izrazom odnosno referencom na metodu, budući da se radi o funkcijskom sučelju.

5. Najčešće i nije potrebno definirati posebna sučelja kako je to napravljeno u prethodnom primjeru. Java 8 API nudi definicije standardnih, parametriziranih, funkcijskih sučelja `Function<T>`, `Predicate<T>` i `Consumer<T>` uz pomoć kojih je moguće napisati univerzalnu metodu za ispis osoba, ali i općenitije metode za „obradu“ osoba čime bi se obuhvatio ispis, ali i druge akcije nad osobama. Upotrebom generics-a možemo rješenje dalje generalizirati te napisati metodu za obradu podataka koja će moći raditi s osobama, ali i s bilo kojim drugim iterabilnim tipovima.
Trebate napisati sljedeće statičke metode i isprobati njihov rad u aplikaciji (klasa `Main`) pozivanjem s lambda izrazima i referencama na metode:

- `public static void printPersons(Iterable<Person> persons, Predicate<Person> criteria, Function<Person, String> formatter)`
  (metoda ispisuje osobe koje zadovoljavaju određeni kriterij u zadanom formatu)
- `public static void processPersons(Iterable<Person> persons, Predicate<Person> criteria, Consumer<Person> action)`
  (metoda omogućava izvođenje specificirane akcije nad odabranim osobama)
- `public static <T, R> void processData(Iterable<T> data, Predicate<T> criteria, Function <T, R> mapper, Consumer<R> action)`
  (generička metoda prihvaća proizvoljni iterabilni tip T, selektira podatke, preslikava podatke tipa T u tip R te izvodi specificiranu akciju nad podacima tipa R)

6. U sklopu klase aplikacije `Main` napišite statičke metode:

- `static Map<Integer, String> convertIterablePersonsToMap(Iterable<? extends Person> persons)`
  (metoda iterabilni slijed podataka tipa `Person` pretvara u mapu u kojoj ključ predstavlja `id` osobe, a vrijednost je `String` sastavljen od imena i prezimena dotične osobe)
- `static <K, V, E> Map<K, V> convertIterableToMap(Iterable<? extends E> elements, Function<E, K> keyFunction, Function<E, V> valueFunction)`
  (generička varijanta gornje metode koja pored reference na iterabilni tip, kao argumente prima funkcije za pretvorbu iterabilnog tipa u ključ, odnosno vrijednost mape)

Napišite aplikaciju u kojoj ćete koristeći listu osoba (`loadPersons`) isprobati rad gore navedenih metoda, ali i upotrebu sljedećih default metoda koje su ugrađene u sučelja iz Java Collection Frameworks-a:

`default void forEach(Consumer<? super T> action)`	FROM `Iterable`
`default void sort(Comparator<? super E> c)`	FROM `List`
`default boolean removeIf(Predicate<? super E> filter)`	FROM `Collection`
`default void forEach(BiConsumer<? super K,? super V> action)`	FROM `Map`

Kod poziva svih metoda koje kao tipove argumenata imaju funkcijska sučelja koristite lambda izraze.

7. U sklopu klase aplikacije `Main` napišite statičke metode:

- `wordCount(List<String> words, String ... lookingFor)`
  (Metoda u listi riječi broji pojavu onih riječi koji su navedeni u varargs-u metode. Riječ i broj pojave riječi se spremaju u mapu koja se vraća kao rezultat. Pretpostavite da su sve riječi pisane malim slovima.)
- `charactersFrequency(String str)`
  (Metoda u stringu broji pojavu znakova. Znak i broj pojavljivanja znaka se spremaju u mapu, a mapa se vraća kao rezultat.)

Napomena: prilikom pisanja koda koriste default metode `computeIfPresent` i `computeIfAbsent` iz sučelja `Map` koje primaju argumente tipa `BiFunction`, odnosno `Function`. Isprobajte rad napisanih metoda.

8. Napišite parametriziranu klasu `Sequence<T>` koja predstavlja slijed podataka tipa `T`, a podatke čuva u listi. Klasa ima atribut i metode kako je prikazano u tablici. Uočite metode koje vraćaju sekvencu i one koje ne vraćaju. Isprobajte upotrebu objekta iz klase `Sequence` u aplikaciji.

`class Sequence <T>`	
`private List<T> values`	
`public Sequence(T... values)`	Konstruktor s varargs
`public Sequence(List<T> values)`	Konstruktor s List
`public Sequence<T> filter(Predicate<T> filter)`	Filtrira vrijednosti iz sekvence, a filtrirane vrijednosti se **vraćaju kao nova sekvenca**.
`public <R> Sequence<R> map(Function<? super T,? extends R> mapper)`	Preslikava svaku vrijednost u sekvenci u skladu s mapper funkcijom. Preslikane vrijednosti se **vraćaju kao nova sekvenca**.
`public void forEach(Consumer<T> action)`	Izvodi zadanu akciju nad svim vrijednostima sekvence.
`public List<T> toList()`	Pretvara sekvencu u listu
`public Boolean all(Predicate<T> predicate)`	Vraća `true` ako svi elementi zadovoljavaju zadani predikat.

Rješenja zadataka dostupna su na:

https://github.com/FER-OOP/Lectures/tree/master/Exercises/Homework-11