Important -- use Stream API in your solutions
all other (non Stream API) solutions will be IGNORED during review

Tests are given to give you detailed information about the requirement

1 - Convert elements of a collection to upper case.

```
@Test
public void transformShouldConvertCollectionElementsToUpperCase() {
    List<String> collection = asList("My", "name", "is", "John", "Doe");
    List<String> expected = asList("MY", "NAME", "IS", "JOHN", "DOE");
    assertThat(transform(collection)).hasSameElementsAs(expected);
}
```

2 - Filter collection so that only elements with less than 4 characters are returned.

```
@Test
public void transformShouldFilterCollection() {
    List<String> collection = asList("My", "name", "is", "John", "Doe");
    List<String> expected = asList("My", "is", "Doe");
    assertThat(transform(collection)).hasSameElementsAs(expected);
}
```

3 - Flatten multidimensional collection (read about .flatMap non-terminal operation and use it)

```
@Test
public void transformShouldFlattenCollection() {
    List<List<String>> collection = asList(asList("Viktor", "Farcic"), asList("John", "Doe", "Third"));
    List<String> expected = asList("Viktor", "Farcic", "John", "Doe", "Third");
    assertThat(transform(collection)).hasSameElementsAs(expected);
}
```

4 -Get oldest person from the collection

5 - Sum all elements of a numeric collection

6 - Get names of all kids (under age of 18)

```
@Test
public void getKidNameShouldReturnNamesOfAllKidsFromNorway() {
    Person sara = new Person("Sara", 4);
    Person viktor = new Person("Viktor", 40);
    Person eva = new Person("Eva", 42);
    Person anna = new Person("Anna", 5);
    List<Person> collection = asList(sara, eva, viktor, anna);
    assertThat(getKidNames(collection))
            .contains("Sara", "Anna")
            .doesNotContain("Viktor", "Eva");
}
```

7 Partition adults and kids

```
@Test
public void partitionAdultsShouldSeparateKidsFromAdults() {
    Person sara = new Person("Sara", 4);
    Person viktor = new Person("Viktor", 40);
    Person eva = new Person("Eva", 42);
    List<Person> collection = asList(sara, eva, viktor);
    Map<Boolean, List<Person>> result = partitionAdults(collection);
    assertThat(result.get(true)).hasSameElementsAs(asList(viktor, eva));
    assertThat(result.get(false)).hasSameElementsAs(asList(sara));
}
```

8 - Group people by nationality

```
@Test
public void partitionAdultsShouldSeparateKidsFromAdults() {
    Person sara = new Person("Sara", 4, "Norwegian");
    Person viktor = new Person("Viktor", 40, "Serbian");
    Person eva = new Person("Eva", 42, "Norwegian");
    List<Person> collection = asList(sara, eva, viktor);
    Map<String, List<Person>> result = groupByNationality(collection);
    assertThat(result.get("Norwegian")).hasSameElementsAs(asList(sara, eva));
    assertThat(result.get("Serbian")).hasSameElementsAs(asList(viktor));
}
```

9 - Return people names separated by comma

```
@Test
public void toStringShouldReturnPeopleNamesSeparatedByComma() {
    Person sara = new Person("Sara", 4);
    Person viktor = new Person("Viktor", 40);
    Person eva = new Person("Eva", 42);
    List<Person> collection = asList(sara, viktor, eva);
    assertThat(namesToString(collection))
            .isEqualTo("Names: Sara, Viktor, Eva.");
}
```