Zadatak 1.

Napisati generičku klasu *Ntuple<T>* kojom se mogu opisati razne n-torke usporedivog tipa (npr. n-torka stringova, n-torka brojeva, …). Objekt klase *Ntuple* se može instancirati korištenjem konstruktora koji prima 2 ili više argumenata tipa T

Ntuple mora omogućiti dohvat (*get(index)*) i postavljanje (*set(index, element)*) vrijednosti na određenoj poziciji u n-torki (1 do n). Veličina n-torke se može dobiti pozivom metode *size*.

Dodatno, trebate omogućiti da se n-torke mogu ispravno dodavati u postojeće Javine kolekcije. Dvije n-torke su jednake ako su im vrijednosti na istim pozicijama jednake, a prilikom usporedbe prednost ima ona n-torka koja ima manju vrijednost na nekoj od ranijih pozicija. Npr. (4, 5, 8, 9) je ispred (4, 5, 9, 2), ali npr. iza (4, 4, 9, 5). Moguće je usporediti n-torke različite veličine, pa je tako npr. (3, 5) iza (3, 4, 1), ali ispred (3, 5, 1).

Nadjačati metodu *toString* tako da se n-torka ispiše unutar zagrada kao što je gore navedeno.

Primjer glavnog programa za 1. zadatak:

```
package hr.fer.oop.recap2.task1;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Main {

  public static void main(String[] args) {
    List<Ntuple<Integer>> list = new ArrayList<>();
    list.add(new Ntuple<>(4, 5, 8, 9));
    list.add(new Ntuple<>(4, 5, 9, 2));
    list.add(new Ntuple<>(3, 6, 2, 7));
    list.add(new Ntuple<>(3, 6, 2, 1));
    list.add(new Ntuple<>(4, 4, 6, 5));
    Ntuple<Integer> tuple = new Ntuple<>(0, 0, 0);
    tuple.set(1, 4);
    tuple.set(2, 4);
    tuple.set(3, 2);
    list.add(tuple);

    list.add(new Ntuple<>(3, 9));

    // list is now: [(4, 5, 8, 9), (4, 5, 9, 2), (3, 6, 2, 7), (3, 6, 2, 1), (4, 4,
    // 6, 5), (4, 4, 2), (3, 9)]
    System.out.println("Before: " + list);
    Collections.sort(list);
    // list is [(3, 6, 2, 1), (3, 6, 2, 7), (3, 9), (4, 4, 2), (4, 4, 6, 5), (4, 5,
    // 8, 9), (4, 5, 9, 2)]
    System.out.println("After: " + list);

    Ntuple<Integer> first = list.get(0);

    System.out.println(new Ntuple<>(3, 6, 2, 1).equals(first)); // true
    System.out.println(first.equals(list.get(1))); // false
    first.set(4, 7);
    System.out.println(first.equals(list.get(1))); // true
    first.set(1, 8); // what would happen in case of first = ...
    Collections.sort(list);
    // [(3, 6, 2, 7), (3, 9), (4, 4, 2), (4, 4, 6, 5), (4, 5, 8, 9), (4, 5, 9, 2),
    // (8, 6, 2, 7)]
    System.out.println("Sorted after the first element was changed:\n " + list);
  }
}
```

Zadatak 2.
Izvođenjem iz klase *Ntuple* napisati generičku klasu *Pair* koja predstavlja par elemenata istog tipa. Par se instancira pozivom kostruktora koji prima prvi i drugi element. Prvi i drugi element ćemo nazivati x i y, pa je potrebno napraviti odgovarajuće *gettere* i *settere (getX, getY, odnosno setX i setY)*.

U slučaju da se par parametrizirana po nekom numeričkom tipu, ova klasa može služiti kao način zapisivanja točaka u koordinatnom prostoru. Stoga se u posebnoj klasi *DistanceFromOrigin* može napisati implementaciju sučelja *Function<Pair<? extends Number>, Double>* koja će vratiti udaljenost točke od ishodišta.

Nakon toga je potrebno napisati predikat *QuadrantPredicate* koji će provjeravati pripada li neka točka nekom od odabranih kvadranata. Konstruktor ovog predikata prima 4 *boolean* vrijednosti kojima se određuje da je točka dobra ako pripada nekom od kvadranata. Npr. instancira se li se predikat s *new QuadrantPredicate(true, false, false, true)* točka će po tom predikatu biti „dobra" ako pripada prvom ili četvrtom kvadrantu (u ovom slučaju to znači da je vrijednost x-koordinate mora biti pozitivna, a vrijednost y-koordinata nije bitna). Za točke na nekoj od koordinatnih osi ćemo smatrati da ne pripadaju nijednom kvadrantu. U napisanom predikatu nadjačati metodu *toString()* tako da prikazuje postavljene parametre.

Nakon toga izvođenjem iz *Pair* napisati klasu *Point* koja predstavlja točku u koordinatnoj ravnini s cjelobrojnim koordinatama. Stvoriti listu nekih točaka i za svaku od točaka koja zadovoljava prethodno napisani predikat, ispisati njenu udaljenost od ishodišta.

Primjer glavnog programa za 2. zadatak:

```
package hr.fer.oop.recap2.task2;

import java.util.LinkedList;
import java.util.List;
import java.util.function.Function;
import java.util.function.Predicate;

public class Main {

  public static void main(String[] args) {
    Pair<String> p = new Pair<>("first", "second");
    System.out.println(p.toString()); // (first, second)
    p.setX(p.getY().replace("cond", "rious"));
    System.out.println(p); // (serious, second)

    List<Pair<Double>> points = new LinkedList<>();
    points.add(new Pair<>(1.5, 5.0));
    points.add(new Pair<>(3.0, -4.0));
    points.add(new Pair<>(-5.0, -12.0));
    points.add(new Pair<>(-1.0, 3.5));

    DistanceFromOrigin distance = new DistanceFromOrigin();
    for (var point : points) {
      System.out.format("%s dist = %.2f%n", point, distance.apply(point));
    }
    // print should be
    // (1.5, 5.0) dist = 5.22
    // (3.0, -4.0) dist = 5.00
    // (-5.0, -12.0) dist = 13.00
    // (-1.0, 3.5) dist = 3.64

    List<Point> intpoints = new LinkedList<>();
    intpoints.add(new Point(1, 5));
    intpoints.add(new Point(3, -4));
    intpoints.add(new Point(-5, -12));
    intpoints.add(new Point(-9, 12));

    // the next print should be
//              Predicate: Take points from quadrants 1-4? (false, true, true, true)
//              (3, -4) dist = 5.00
//              (-5, -12) dist = 13.00
//              (-9, 12) dist = 15.00
    Predicate<Pair<? extends Number>> predicate = new QuadrantPredicate(false, true, true, true);
    System.out.println("Predicate: " + predicate.toString());
    for (var point : intpoints) {
      if (predicate.test(point))
        System.out.format("%s dist = %.2f%n", point, distance.apply(point));
    }
  }
}
```

Zadatak 3.
Napisati komparator cjelobrojnih točaka na način da se točke uspoređuju po udaljenosti od ishodišta.

Pitanja za razmišljanje:

a) Možemo li i kako s ovim komparatorom sortirati listu cjelobrojnih točaka?
b) Što će se dogoditi ako ovaj komparator koristimo kao komparator za TreeSet?
c) Što će se dogoditi ako u TreeSetu koristimo prethodno napisani prirodni komparator iz klase Point?

Korištenjem vlastitog ili ugrađenog kompozitnog komparatora sortirati listu cjelobrojnih točaka na način da se točke sortiraju po udaljenosti od ishodišta, a ako su jednako udaljene sortirati ih po prirodnom poretku.

Varijacije zadatka s korištenjem kolekcijskih tokova:

a) Umjesto sortiranja liste, koristeći kolekcijske tokove stvoriti listu stringova čiji je sadržaj nastao odabirom točaka iz određenih kvadranata (upotrijebiti prethodno napisane predikate), silazno sortiranih po udaljenosti od ishodišta. Svaki string iz liste je oblika dist(x, y)=vrijednost

b) Koristeći kolekcijske tokove izračunati prosječnu udaljenost točaka (iz nekog od kvadranata) od ishodišta

Zadatak 4.

Napisati program koji će za neku listu cjelobrojnih točaka kreirati mapu tipa *Map<Integer, Set<Point>>* koja će za ključeve imati kvadrante 1-4, a vrijednost će biti skup točaka koje pripadaju tom kvadrantu. Iteriranje po skupu mora vraćati točke uzlazno po udaljenosti od ishodišta. Ako su dvije točke jednako udaljene od ishodišta, njihov međusobni poredak prilikom iteriranja nije bitan.

Zadatak 5.

Napisati klasu *PythagoreanTriangles* koja u konstruktoru prima 2 pozitivna cijela broja a i b (baciti iznimku *llegalArgumentException* ako argumenti nisu ispravni) te implementira sučelje *Iterable<Ntuple<Integer>>* tako da vraća sve Pitagorine trojke oblika (x, y, z) takve da je:

$0 < x \leq a$, $0 < y \leq b$, $x < y$ te $z^2 = x^2 + y^2$.

Primjer glavnog programa:

```
package hr.fer.oop.recap2.task5;

import hr.fer.oop.recap2.task1.Ntuple;

public class Main {

	public static void main(String[] args) {
		PythagoreanTriangles triples = new PythagoreanTriangles(50, 80);
		for (Ntuple<Integer> p : triples)
			System.out.println(p);
	}
}
```

Ispis:

```
(3, 4, 5)
(5, 12, 13)
(6, 8, 10)
(7, 24, 25)
(8, 15, 17)
(9, 12, 15)
(9, 40, 41)
(10, 24, 26)
(11, 60, 61)
(12, 16, 20)
(12, 35, 37)
(14, 48, 50)
(15, 20, 25)
(15, 36, 39)
(16, 30, 34)
(16, 63, 65)
(18, 24, 30)
(18, 80, 82)
(20, 21, 29)
(20, 48, 52)
(21, 28, 35)
(21, 72, 75)
(24, 32, 40)
(24, 45, 51)
(24, 70, 74)
(25, 60, 65)
(27, 36, 45)
(28, 45, 53)
(30, 40, 50)
(30, 72, 78)
(32, 60, 68)
(33, 44, 55)
(33, 56, 65)
(36, 48, 60)
(36, 77, 85)
(39, 52, 65)
(39, 80, 89)
(40, 42, 58)
(40, 75, 85)
(42, 56, 70)
(45, 60, 75)
(48, 55, 73)
(48, 64, 80)
```

Dodatne ideje za vježbanje:
U primjerima koji koriste *DistanceFromOrigin i QuandrantPredicate* definirajte neke druge funkcije i predikate koristeći lambda izraze i/ili reference na metode.