[GUI] Ćwiczenia IV:

```
public enum E {
    VALUE1, VALUE2;
};
```

```
// https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Compar
able.html
class K implements Comparable<K> {

    // ...
    @Override
    public int compareTo(K ob)
    {
        // porównujemy 2 obiekty: this, ob
        // wywołanie: obiekt1.compareTo(obiekt2)

        // wynik < 0 kiedy uważamy, że this < ob
        // wynik = 0 kiedy uważamy, że this = ob
        // wynik > 0 kiedy uważamy, że this > ob
        return wynik;
    }
}
```

```
// https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/util/Compar
ator.html
class InnyPorzadek implements java.util.Comparator<K> {

    @Override
    public int compare(K ob1, K ob2)
    {
        // ...

        // wynik < 0 kiedy uważamy, że ob1 < ob2
        // wynik = 0 kiedy uważamy, że ob1 = ob2
        // wynik > 0 kiedy uważamy, że ob1 > ob2
        return wynik;
    }
}
```

```
public class Test {
  public static void main(String[] args) {

      java.util.List<K> lista;// = ...

      // sortowanie listy obiektów typu K zgodnie
      // z tzw. naturalnym porządkiem, określonym w Comparable<K>
```

```
      Collections.sort(lista);

      // sortowanie listy obiektów zgodnie
      // z innym podanym porządkiem
      Collections.sort(lista, new InnyPorzadek(...));

      // sortowanie listy obiektów zgodnie
      // porządkiem, podanym jako wyrażenie lambda
      Collections.sort(lista, (ob1, ob2) -> { kod / wynik metody compare });

      // ..
  }
}
```

```
// Jak działa rozszerzona instrukcja for?

// lista
List<String> lista = new ArrayList<String>();

// dodawanie elementów do listy
lista.add(...)
// ...

// pętla wydrukująca elementy listy
for (String e: lista) {
    System.out.println(e);
}

/*  // powyższa pętla działa ponieważ ArrayList implementuje interfejs Iterable
...
    // tak naprawdę:
Iterator<String> iter = lista.iterator();
while(iter.hasNext()) {
    String e = iter.next();
    System.out.println(e);
}
*/
// --------------------------------------
// Interfejs java.lang.Iterable
// https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Iterab
le.html

interface Iterable<T> {

    Iterator<T> iterator();

    // ...
}

// --------------------------------------
```

```
// Interfejs java.util.Iterator<E>
// https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Iterat
or.html
interface Iterator<E> {

    boolean hasNext();

    E next();

    // ...
}
```

```
// --------------------------------------
import java.util.Iterator;
// java.lang.Iterable
class IterNap implements Iterable<Character> {

    // ...
    public Iterator<Character> iterator() {

        // return new WlasnyIterator<Character>(...);
        return new Iterator<Character>(){

                // ...
            public boolean hasNext() {
                   // ...
            }

            public Character next() {
                    // ...
            }
        };
    }
}
/*
class WlasnyIterator implements Iterator<Character> {

            // ...

            public boolean hasNext() {
                   // ...
            }


            public Character next() {
                    // ...
            }

}
*/
// --------------------------------------
```

```
public class IterNapTest {

    public static void main(String [] args)
    {
        IterNap napis = new IterNap("...");

            // iteracja po znakach napisu...
        for (char z: napis)
            System.out.print(z + " ");

    }
}
```

```
import java.util.*;

enum Kryterium {
    imie, wiek;
}

public class Test {

    public static void main(String[] args) {

        List<Osoba> lista = Arrays.asList(
                                new Osoba("Anna", 23),
                                new Osoba("Maria", 22),
                                new Osoba("Anna", 20),
                                new Osoba("Wojciech", 21)
                            );

        Collections.sort(lista, new KomparatorOsob(Kryterium.imie));
        System.out.println(lista);

        Collections.sort(lista, new KomparatorOsob(Kryterium.wiek));

        Collections.sort(lista, Comparator.comparingInt(Osoba::getWiek));
        //Collections.sort(lista, (o1, o2) ->  o1.getWiek() - o2.getWiek());
        System.out.println(lista);

        Collections.sort(lista);
        System.out.println(lista);
    }
}
```

```
class Osoba implements Comparable<Osoba>{

    private String imie;
    private int wiek;

    public Osoba(String imie, int wiek) {
```

```
        this.imie = imie;
        this.wiek = wiek;
    }

    public int getWiek() {
        return this.wiek;
    }

    public String getImie() {
        return this.imie;
    }

    @Override
    public int compareTo(Osoba o) {
        if (this.imie.compareTo(o.imie) != 0)
            return this.imie.compareTo(o.imie);
        return wiek - o.wiek;
    }

    public String toString() {
        return "(" + this.imie + ", " + this.wiek + ")";
    }
}
```

```
class KomparatorOsob implements Comparator<Osoba> {

    private Kryterium kr;

    KomparatorOsob(Kryterium kr) {
        this.kr = kr;
    }

    @Override
    public int compare(Osoba o1, Osoba o2) {

        return switch (this.kr) {
            case wiek -> o1.getWiek() - o2.getWiek();
            case imie -> o1.getImie().compareTo(o2.getImie());
        };
    }
}
```