PLIKI

1.Korzystając z metod listFiles(FileFilter) i isDirectory z klasy java.io.File, napisz metodę zwracającą wszystkie podkatalogi wskazanego katalogu. Wykorzystaj wyrażenie lambda zamiast obiektu FileFilter. Wykonaj to samo za pomocą wyrażenia z metodą i anonimowej klasy wewnętrznej.

```
import java.io.File;
import java.util.Arrays;

public class ArrayUtil {
    public static String[] subdirectoryList(String path){

        File f = new File(path);
        return f.list((dir, name) -> dir.isDirectory());
    }

    public static void main(String[] args){
        System.out.println(Arrays.toString(subdirectoryList("src")));
        System.out.println(Arrays.toString(subdirectoryList("out")));
    }
}
```

2. Korzystając z metody list(FilenameFilter) klasy java.io.File, napisz metodę zwracającą wszystkie pliki ze wskazanego katalogu ze wskazanym rozszerzeniem. Użyj wyrażenia lambda, a nie FilenameFilter.

```
import java.io.File;
import java.util.List;
public class ArrayUtil {
    public static String[] printDir(String path, String extension){
        String[] result = {};
        try{
            File f = new File(path);
            result = f.list((dir, name) -> name.endsWith(extension));
        }
        catch (Exception e){

        }
        return result;
    }

    public static void main(String[] Args){
        List.of(printDir("src", ".txt")).forEach(System.out::println);
    }
}
```

3.Mając tablice obiektów file posortuj je w taki sposób by katalogi znalazły się przed plikami a w każdej grupie elementy zostały posortowane według nazwy . Uzyj lambda przy implemetowaniu interfejsu Comparator.

```
import java.io.File;
import java.util.Arrays;
```

```
public class ArrayUtil {
    public static void main(String[] args) {
        File[] files = new File("C:\\").listFiles();
        Arrays.sort(files, (a,b) -> {
            if(a.isDirectory() && !b.isDirectory())
                return -1;
            if(!a.isDirectory() && b.isDirectory())
                return 1;
            return a.compareTo(b);
        });

        for(File f: files){
            System.out.println(f);
        }
    }
}
```

4. pliki przed katalogami odwrocic tamto wyżej

```
import java.io.File;
import java.util.Arrays;

public class ArrayUtil {
    public static void main(String[] args) {
        File[] files = new File("C:\\").listFiles();
        Arrays.sort(files, (a,b) -> {
            if(!a.isDirectory() && b.isDirectory())
                return -1;
            if(a.isDirectory() && !b.isDirectory())
                return 1;
            return a.compareTo(b);
        });

        for(File f: files){
            System.out.println(f);
        }
    }
}
```

5. bez wskazanego rozszerzenia (2)

```
import java.io.File;
import java.util.List;
public class ArrayUtil {
    public static String[] printDir(String path, String extension){
        String[] result = {};
        try{
            File f = new File(path);
            result = f.list((dir, name) -> !name.endsWith(extension));
        }
        catch (Exception e){

        }
        return result;
    }
```

```
    public static void main(String[] Args){
        List.of(printDir("src", ".txt")).forEach(System.out::println);
    }
}
```

1 i 2 ZADANIA

1.Napisz statyczną generyczną metodę ArrayUtil.isSorted, która sprawdza czy podana jako argument tablica obiektów klasy T jest uporządkowana nierosnąco. W defnicji metody zadeklaruj, że klasa T musi implementować generyczny interfejs Comparable. W funkcji main przetestuj działanie metody isSorted dla posortowanych i nieposortowanych tablic obiektów klas Integer, LocalTime oraz String.

```
import java.time.LocalDate;

public class ArrayUtil {
    public static void main(String[] args) {
        Integer[] unsorted = new Integer[]{155,12,12};
        Integer[] sorted = new Integer[]{12,44,55};
        LocalDate[] unsortedDates = new LocalDate[]{
                LocalDate.of(2021,8,2),
                LocalDate.of(2019,11,3),
                LocalDate.of(2018,3,1),
        };
        LocalDate[] sortedDates = new LocalDate[]{
                LocalDate.of(2020,8,2),
                LocalDate.of(2020,10,6),
                LocalDate.of(2020,11,12),
        };
        String[] unsortedString = new String[]{"4","2","1"};
        String[] sortedString = new String[]{"1","2","3"};
        System.out.println(ArrayUtil.isSorted(sortedString));
        System.out.println(ArrayUtil.isSorted(unsortedString));
        System.out.println(ArrayUtil.isSorted(sortedDates));
        System.out.println(ArrayUtil.isSorted(unsortedDates));
        System.out.println(ArrayUtil.isSorted(sorted));
        System.out.println(ArrayUtil.isSorted(unsorted));


    }

    public static <T extends Comparable<T>> boolean isSorted(T[] ar){
        for (int i = 0; i< ar.length-1; i++)
            if (ar[i].compareTo(ar[i+1])<0) return false;
            return true;
    }

    }
```

1.Napisz statyczną generyczną metodę ArrayUtil.isSorted, która sprawdza czy podana jako argument tablica obiektów klasy T jest uporządkowana niemalejąco. W defnicji metody zadeklaruj, że klasa T musi implementować generyczny interfejs Comparable. W funkcji main przetestuj działanie metody isSorted dla posortowanych i nieposortowanych tablic obiektów klas Integer, LocalTime oraz String.

```
import java.time.LocalDate;

public class ArrayUtil {
    public static void main(String[] args) {
        Integer[] unsorted = new Integer[]{155,12,44};
        Integer[] sorted = new Integer[]{12,44,55};
        LocalDate[] unsortedDates = new LocalDate[]{
                LocalDate.of(2021,8,2),
                LocalDate.of(2019,11,3),
                LocalDate.of(2020,3,1),
        };
        LocalDate[] sortedDates = new LocalDate[]{
                LocalDate.of(2020,8,2),
                LocalDate.of(2020,10,6),
                LocalDate.of(2020,11,12),
        };
        String[] unsortedString = new String[]{"4","2","3"};
        String[] sortedString = new String[]{"1","2","3"};
        System.out.println(ArrayUtil.isSorted(sortedString));
        System.out.println(ArrayUtil.isSorted(unsortedString));
        System.out.println(ArrayUtil.isSorted(sortedDates));
        System.out.println(ArrayUtil.isSorted(unsortedDates));
        System.out.println(ArrayUtil.isSorted(sorted));
        System.out.println(ArrayUtil.isSorted(unsorted));


    }

    public static <T extends Comparable<T>> boolean isSorted(T[] ar){
        for (int i = 0; i< ar.length-1; i++)
            if (ar[i+1].compareTo(ar[i])<0) return false;
            return true;
    }

    }
```

2. Napisz generyczną statyczną metodę print, której argumentem jest dowolny obiekt implementujący interfejs Iterable. Metoda print wypisuje elementy swojego argumentu oddzielając je przecinkami. W funkcji main przetestuj działanie metody print dla obiektów kilku różnych klas implementujących interfejs Iterable.

```
import java.util.Iterator;
import java.util.*;
import java.util.LinkedList;
import java.util.List;
import java.util.*;
import java.lang.Character;

public class ArrayUtil {
    public static <E extends Iterable<?>> void print(E obiekt) {
        Iterator<?> iter =  obiekt.iterator();
        if (iter.hasNext())
            System.out.print("[" + iter.next() + "],");
        while (iter.hasNext())
            System.out.print("[" + iter.next() + "],");

    }

    public static void main(String[] args) {
        LinkedList<Integer> linkedList1 = new
LinkedList<>(List.of(1,2,3,5));
        System.out.println(linkedList1);
        ArrayUtil.print(linkedList1);

        TreeSet<Character>treeMap = new TreeSet<>(List.of('A','B'));
        System.out.println("");
        System.out.println(treeMap);
        ArrayUtil.print(treeMap);
        System.out.println("");

        ArrayList<String>arrayList = new ArrayList<>(List.of("aaa","bbb"));
        System.out.println(arrayList);
        ArrayUtil.print(arrayList);
    }
}
```

3. Napisz statyczną generyczną metodą ArrayUtil.removeRepeatedElements, która zwraca obiekt klasy ArrayList zawierający niepowtarzające się elementy tablicy obiektów klasy T podanej jako argument metody (zwracany obiekt stanowi zbiór reprezentujący tablicę elementów podaną jako argument metody). Klasa T może implementować generyczny interefejs Comparable. W funkcji main przetestuj działanie metody removeRepeatedElements dla tablic obiektów Character oraz LocalTime.

```
import java.time.LocalTime;
import java.util.ArrayList;

public class ArrayUtil {
    public static <T extends Comparable<T>> ArrayList<T>
removeRepeatedElements(T[] tab)
    {
        ArrayList<T> noRepeated = new ArrayList<>();
        for (int i=0; i<tab.length; i++)
        {
            if (!noRepeated.contains(tab[i]))
                noRepeated.add(tab[i]);
        }


        return noRepeated;
    }
    public static void main(String[] args) {
        LocalTime[] tab1 = {LocalTime.of(5,20, 4), LocalTime.of(6,14,2),
LocalTime.of(1,15,7), LocalTime.of(6,14,2)};
        System.out.println(ArrayUtil.removeRepeatedElements(tab1));

        Character[] tab2 = {'a','b','c','d','b','e'};
        System.out.println(ArrayUtil.removeRepeatedElements(tab2));
    }
}
```

4. Napisz generyczną statyczną metodę printWithSemicolon, której argumentem jest dowolny obiekt implementujący interfejs Iterable. Metoda printWithSemicolon wypisuje elementy swojego argumentu oddzielając je średnikami. W funkcji main przetestuj działanie metody printWithSemicolon dla obiektów czterech różnych klas implementujących interfejs Iterable.

```
import java.util.Iterator;
import java.util.*;
import java.util.LinkedList;
import java.util.List;
import java.lang.Character;

public class ArrayUtil {
    public static <E extends Iterable<?>> void printWithSemicolon(E obiekt)
{
        Iterator<?> iter =  obiekt.iterator();
        if (iter.hasNext())
            System.out.print(iter.next() + ";");
        while (iter.hasNext())
            System.out.print(iter.next() + ";");

    }


    public static void main(String[] args) {
        LinkedList<Integer> linkedList1 = new
LinkedList<>(List.of(1,2,3,5));
        ArrayUtil.printWithSemicolon(linkedList1);

        TreeSet<Character>treeMap = new TreeSet<>(List.of('A','B'));
        System.out.println("");
        ArrayUtil.printWithSemicolon(treeMap);
        System.out.println("");

        ArrayList<String>arrayList = new ArrayList<>(List.of("aaa","bbb"));
        ArrayUtil.printWithSemicolon(arrayList);
    }
}
```

5.Napisz generyczną statyczną metodę printMarginal, której argumentem jest dowolny obiekt implementujący interfejs Iterable<E>. Metoda printMarginal wypisuje krańcowe elemety swojego argumetu(pierwszy i ostatni elemet). W funkcji main przetestuj dxiałanie metody printMArginal dla obiektów 2 róznych klas implemtujacuch interfrjs Iterable.

```
import java.util.*;

public class ArrayUtil {
    public static <E, T extends Iterable<E>> void printMarginal(T a){
        Iterator<E> it = a.iterator();
        System.out.println("Pierwszy: " + it.next());

        int ile = 0;
        while(it.hasNext()){
            ile++;
            it.next();
        }
        it = a.iterator();
        for(int i = 0; i < ile; i++){
            it.next();
        }
        System.out.println("Ostatni: " + it.next());
    }


    public static void main(String[] args) {
        ArrayList<Integer> A = new ArrayList<>();
        A.add(1);
        A.add(2);
        A.add(3);
        A.add(23);
        A.add(22);
        A.add(522);

        printMarginal(A);

        Stack<String> B = new Stack<>();
        B.add("a");
        B.add("b");
        B.add("c");
        B.add("d");

        printMarginal(B);

        Vector<String> C = new Vector<>();

        C.add("q");
        C.add("w");
        C.add("e");
        C.add("r");

        printMarginal(C);

        LinkedList<Integer> D = new LinkedList<>();
        D.add(1);
        D.add(2);
        D.add(3);
        D.add(4);

        printMarginal(D);
```

```
    }
}
```

6.Napisz statyczną generyczną metodę ArrayUtil.jestpalindromem, która sprawdza czy podana jaka argument tablica obiektów klasy T ma tę własnośc , że wypisanie tej tablicy od początku do końca daje taki sam wynik jak jej wypisanie od końca do początku . W definicji metody zadeklaruj ze klasa T musi implemtować generyczny interfejs Comparable. W funkcji main przetetuj dla posortowanych i nieposortowanych tablic obuketo klas integer i localtime.

```
import java.time.LocalTime;


public class ArrayUtil {
    public static <T extends Comparable<T>> boolean jestPalindromem(T[] a){

        for (int i = 0; i < a.length - 1; i ++) {
            if (a[i].compareTo(a[a.length - 1 - i]) < 0) {
                return false;
            }
        }
        return true;
    }
    public static void main(String[] args) {
        Integer[] integers1 = {1, 3, 6, 2, 1};
        Integer[] integers2 = {1, 5, 1, 5, 1};
        LocalTime[] localTimes1 = {LocalTime.of(10, 20, 10),
LocalTime.of(10, 20, 50), LocalTime.of(11,15,26)};
        LocalTime[] localTimes2 = {LocalTime.of(10, 20, 30),
LocalTime.of(10, 20, 30)};
        System.out.println(jestPalindromem(integers1));
        System.out.println(jestPalindromem(integers2));
        System.out.println(jestPalindromem(localTimes1));
        System.out.println(jestPalindromem(localTimes2));
    }
}
```

7.Napisz generyczną statyczną mettodę wypiszCoDrugi, której argumetem jest dowolny obiekt implemetujący interfejs Iterable<E>. Metoda wypiszCoDrugi wypisuje(Począwszy od pierwszego eleemtu) co drugi, oddzielając przecinkami. W funkcji main przetestuj Działanie metody wypiszCoDrugi dla obiektów kilku różnych klas implementujących interfejs Iterable<E>.

```
import java.util.*;


public class Main {
    public static void main(String[] args){
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(123);
        lista.add(234);
        lista.add(345);
        lista.add(456);
        lista.add(352);
```

```
        ArrayList<String> lista2= new ArrayList<>();
        lista2.add("A");
        lista2.add("B");
        lista2.add("C");
        lista2.add("D");
        lista2.add("E");

        TreeSet<String> lista3 = new TreeSet<>();
        lista3.add("ala");
        lista3.add("ola");
        lista3.add("pola");
        lista3.add("tola");
        lista3.add("wiola");

        LinkedList<Integer> lista1 = new LinkedList<>();
        lista1.add(1);
        lista1.add(2);
        lista1.add(3);
        lista1.add(4);
        lista1.add(5);

        Stack<String> B = new Stack<>();
        B.add("a");
        B.add("b");
        B.add("c");
        B.add("d");
        B.add("vhuj");


        wypiszCoDrugi(B);
        wypiszCoDrugi(lista1);
        wypiszCoDrugi(lista);
        wypiszCoDrugi(lista2);
        wypiszCoDrugi(lista3);
    }
    public static <E extends Iterable<?>> void wypiszCoDrugi(E t){
        Iterator<?> iterator = t.iterator();
        while(iterator.hasNext()){
            System.out.print(iterator.next());
            if(iterator.hasNext())
                iterator.next();
            if(iterator.hasNext())
                System.out.print(",");
        }
        System.out.println();

    }

}
```

Kontener linked hashset i usuwanie przedostatniego elelemtu::

```
LinkedHashSet<Integer> set = new LinkedHashSet<>();
 set.add(1);
 set.add(2);
 set.add(3);
 Iterator<Integer> iterator = set.iterator();
 while (iterator.hasNext())
 { iterator.next();
 }
iterator.remove();
```

Redukuj usuwa co n eleemt:

```
public class druga {
    public static void redukuj(LinkedList<String> pracownicy, int n) {
        pracownicy.remove(n);
    }

//    public static <T> void redukuj(LinkedList<T> pracownicy, int n) {
//        pracownicy.remove(n);
//    }

    public static void main(String[] args) {
        LinkedList<String> pracownicy = new LinkedList<String>();
        pracownicy.add("Jan Kowalski");
        pracownicy.add("Anna Nowak");
        pracownicy.add("Piotr Wiśniewski");
        pracownicy.add("Katarzyna Lewandowska");

        System.out.println("Lista pracowników przed redukcją:");
        System.out.println(pracownicy);

        int n = 2;
        redukuj(pracownicy, n);

        System.out.println("Lista pracowników po redukcji:");
        System.out.println(pracownicy);
    }
}
```

```
        @Override
        public int compareTo(DownloadFile other) {
            return CharSequence.compare(this.name, other.name);
        }

//        @Override
//        public int compareTo(Osoba o) {
//            int porownanieNazwisk = this.nazwisko.compareTo(o.nazwisko);
//            if (porownanieNazwisk != 0) {
//                return porownanieNazwisk;
//            }
//            return this.dataUrodzenia.compareTo(o.dataUrodzenia);
//        }
//        @Override
//        public DownloadFile clone() throws CloneNotSupportedException{
//            DownloadFile cloned = (DownloadFile) super.clone();
//            cloned.size = (double) size.clone();
//            return cloned;
        @Override
        public DownloadFile clone() throws CloneNotSupportedException {
            return (DownloadFile) super.clone();
        }
    }}
//
//    // Od wersji Javy 1.5 metoda clone może zwracać obiekt klasy, w
której jest zdefiniowana
//    public Pracownik clone() throws CloneNotSupportedException
//    {
//        // wywołanie metody Object.clone()
//        Pracownik cloned = (Pracownik) super.clone();
//
//        // klonowanie modyfikowalnego pola
//        cloned.dataZatrudnienia = (Date)dataZatrudnienia.clone();
//
//        return cloned;
//    }
```