Задание

11. Write a generic method to count the number of elements in a collection that have a specific property (for example, odd integers, prime numbers, palindromes).

**Способ 1.**

**// public class GenericAlgorithms.**

**public static <T> int countIfDefaultPredicate(Collection<T> collection, Predicate<T> condition) {**

**int count = 0;**

**for (T elem : collection)**

**if (condition.test(elem))**

**++count;**

**return count;**

**}**

**// вызов ф-и**

**GenericAlgorithms.countIfDefaultPredicate(array, integer -> integer% 2 != 0));**

**способ 2**

**// public class GenericAlgorithms.**

**public static <T> int countIf(Collection<T> collection, UnaryPredicate<T> condition) {**

**int count = 0;**

**for (T elem : collection)**

**if (condition.test(elem))**

**++count;**

**return count;**

**}.**

**class OddPredicate implements UnaryPredicate<Integer> {**

**public boolean test(Integer i) { return i % 2 != 0; }**

**}**

**// вызов ф-и**

**GenericAlgorithms.countIf(array, new OddPredicate()));**

12. Will the following class compile? If not, why?

public final class Algorithm {

public static <T> T max(T x, T y) {

return x > y ? x : y;

}

}

**Not, cause operator ‘>’ cannot be applied to ‘T’,’T’**

13. Write a generic method to exchange the positions of two different elements in an array.

**public static <T> void swap(T[] array, int i, int j) {**

**T temp = array[i];**

**array[i] = array[j];**

**array[j] = temp;**

**}**

14. If the compiler erases all type parameters at compile time, why should you use generics?

15. What is the following class converted to after type erasure?

public class Pair<K, V> {

public Pair(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey(); { return key; }

public V getValue(); { return value; }

public void setKey(K key) { this.key = key; }

public void setValue(V value) { this.value = value; }

private K key;

private V value;

}

**public class Pair {**

**public Pair(Object key, Object value) {**

**this.key = key;**

**this.value = value;**

**}**

**public Object getKey() { return key; }**

**public Object getValue() { return value; }**

**public void setKey(Object key) { this.key = key; }**

**public void setValue(Object value) { this.value = value; }**

**private Object key;**

**private Object value;**

**}**

16. What is the following method converted to after type erasure?

public static <T extends Comparable<T>>

int findFirstGreaterThan(T[] at, T elem) {

// ...

}

**public static int findFirstGreaterThan(Comparable[] at, Comparable elem) { // ...}**

17. Will the following method compile? If not, why?

public static void print(List<? extends Number> list) {

for (Number n : list)

System.out.print(n + " ");

System.out.println();

}

**Да**

Хорстманн, том 1

ограничение на переменную типа Т



https://www.baeldung.com/java-pairs

**2.1. The *Pair* Class**

We can find the *Pair* class in the *javafx.util* package. The constructor of this class takes two arguments, a key and its corresponding value:

Pair<Integer, String> pair = **new** Pair<>(1, "One");

Integer key = pair.getKey();

String value = pair.getValue();

This example illustrates a simple *Integer* to *String* mapping using the Pair concept.

As shown, the key in the *pair* object is retrieved by invoking a *getKey()* method, while the value is retrieved by calling *getValue().*

**2.2. *AbstractMap.SimpleEntry* and *AbstractMap.SimpleImmutableEntry***

*SimpleEntry* is defined as a nested class inside the *AbstractMap* class. To create an object of this type we can provide a key and value to the constructor:

AbstractMap.SimpleEntry<Integer, String> entry

= **new** AbstractMap.SimpleEntry<>(1, "one");

Integer key = entry.getKey();

String value = entry.getValue();

Задание

1. Использовать стандартные алгоритмы

2. Использовать stream

Хорстманн, том 1

Понятие nодстановочного типа

В подстановочном типе параметр типа может быть переменным . Например, следующий подстановочный тип обозначает любой обобщенный тип Pair, параметр типа которого представляет подкласс, производный от класса Employee, в частности, класс Pair<Manager>, но не класс Pai r<String>.

Pair<? extends Employee>

Задание

3. For example, say you want to write a method that works on List<Integer>, List<Double>, and List<Number>.

Написать метод суммирования и примеры использования метода

**public static double sum(List<? extends Number> list) {**

**double sum = 0.0;**

**for (Number n : list) {**

**sum += n.doubleValue();**

**}**

**return sum;**

**}**

**List<Integer> list1 = Arrays.asList(1, 2, 3);**

**System.out.println("sum of list1 = " + GenericAlgorithms.sum(list1));**

**List<Double> list2= Arrays.asList(1., 2.1, 4.3);**

**System.out.println("sum of list2 = " + GenericAlgorithms.sum(list2));**

4. Как использовать метод из 3 для суммирования элементов множества? Сделать!

**Set<Number> set=new HashSet<>(list2);**

**System.out.println("sum of set = " + GenericAlgorithms.sum(List.copyOf(set)));**

5. Вариант stream.

**System.out.println("(stream) sum = " + set.stream().mapToDouble(Number::doubleValue).sum());**

**System.out.println("(stream) sum = " +list2.stream().mapToDouble(f -> f.doubleValue()).sum());**

6. Обобщенные алгоритмы. Найти min и max TreeSet

**Set<Double> treeSet=new TreeSet<>(list2);**

**System.out.println("min of treeSet = " + Collections.min(treeSet));**

**System.out.println("max of treeSet= " + Collections.max(treeSet));**

7. Имеем List<Student>. Написать метод, который возвращает двух лучших (максимальный средний балл).

**public List<Student> findBestStudents(List<Student> list) {**

**// using correct compareTo**

**//List<Student> sortedList = new ArrayList<>(list.stream().sorted().toList());**

**List<Student> sortedList = list.stream().sorted(Comparator.comparing(Student::getAverageScore)).toList();**

**List<Student> result = new ArrayList<>();**

**result.add(sortedList.get(0));**

**result.add(sortedList.get(1));**

**return result;**

**}**

8. Имеем List<Student>. Написать метод, который возвращает частоту встречаемости студентов с заданным средним баллом.

**public int frequencyByAverageScore(List<Student> students,double averageScore) {**

**return Collections.frequency(students.stream()**

**.map(Student::getAverageScore)**

**.toList(),averageScore);**

**}**

9. Имеем List<Student>. Написать метод, который возвращает частоту встречаемости студентов с заданным свойством. Продемонстрировать на задаче 8.

**public int frequencyByPredicate(List<Student> students,Predicate<Student> predicate) {**

**int count = students.stream().filter(predicate).toList().size();**

**return count;**

**}**

10. Имеем List<Student>. Написать метод, который ищет студента, используя binarySearch().

**public Student FindByBinarySearch(List<Student> students,Student student) {**

**return students.get(Collections.binarySearch(students, student));**

**}**

Хорстманн, том 1

9.6.6. Взаимное преобразование коллекций и массивов

если имеется массив, который требуется преобразовать в коллекцию, для этой цели служит оболочка метода List.of()

9.6.7. Написание собственных алгоритмов

Собираясь писать свой собственный алгоритм (по существу, любой метод, принимающий коллекцию в качестве параметра), старайтесь по возможности оперировать интерфейсами, а не конкретными их реализациями. Допустим, требуется обработать элементы коллекции. Для этого можно, конечно, реализовать метод, аналогичный следующему:

puЫic void processitems(ArrayList<Item> items)

{

for (Item item : items)

сделать что-нибудь с элементом item

Но в этом случае ограничиваются возможности той части кода, где вызывается приведенный выше метод, поскольку ему должны передаваться пункты в списочном массиве типа ArrayList. Если это окажется другой контейнер, его придется сначала перепаковать. Намного лучше принимать в качестве параметра обобщенную коллекцию.

A [Collection](https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html) represents a group of objects known as its elements. The Collection interface is used to pass around collections of objects where maximum generality is desired. For example, by convention all general-purpose collection implementations have a constructor that takes a Collection argument. This constructor, known as a *conversion constructor*, initializes the new collection to contain all of the elements in the specified collection, whatever the given collection's subinterface or implementation type. In other words, it allows you to *convert* the collection's type.

18. Имеем Collection<Type >. You want to create another Collection containing the same elements but with all duplicates eliminated.

**public static <Type> Set<Type> removeDuplicates(Collection<Type> c) {**

**return new HashSet<Type>(c);**

**}**

Проверить для разных типов данных и коллекций

**List<Integer> list = Arrays.asList(1, 2, 3, 3, 1);**

**System.out.println(DataManager.removeDuplicates(list));**

**Deque<Integer> deque = new ArrayDeque<>(list);**

**System.out.println(DataManager.removeDuplicates(deque));**

19. 18 – вариант stream

**public static <Type> Set<Type> removeDuplicatesUsingStream(Collection<Type> c) {**

**return c.stream().collect(Collectors.toSet());**

**}**

20. List<String> Найти слова, которые встречаются более 1 раза

**List<String> listMoreOneTime = new ArrayList<>(Arrays.asList("a", "b", "b", "c", "a","d", "a"));**

**System.out.println("listMoreOneTime: ");**

**System.out.println(listMoreOneTime**

**.stream()**

**.collect(Collectors.groupingBy(Function.identity()))**

**.entrySet()**

**.stream()**

**.filter(stringListEntry -> stringListEntry.getValue().size()>1)**

**.map(Map.Entry::getKey)**

**.toList());**

21. Collection<Type>

Traversing Collections

There are three ways to traverse collections:

1. using aggregate operations (stream)

**collection.stream().forEach(elem -> System.out.println(elem));**

1. with the for-each construct and

**for(Object o:collection){**

**System.out.println(o);**

**}**

1. by using Iterators.

**Collection<Integer> collection=new ArrayList<>();**

**for(Iterator<Integer> it=collection.iterator();it.hasNext();){**

**System.out.println(it.next());**

**}**

Написать метод вывода в трех вариантах!!!

Продемонстрировать для различных типов

22.

Department/Faculty – перечисление

Employee/Student – имеет свойство типа Department

List<Employee>

// Group employees by department

Map<Department, List<Employee>>

23.

Write a method that takes a List<String> and applies [String.trim](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html" \l "trim--" \t "_blank) to each element.

**static void trimListString(List<String> strings) {**

**strings.forEach(s -> s.trim());**

**}**

24. List<String>

to use a SortedSet Specify a Comparator so that case is ignored when sorting and identifying set elements.

**//не работает!!!**

**SortedSet<String> treeSetIgnoreCase = new TreeSet<String>(new Comparator<String>() {**

**@Override**

**public int compare(String o1, String o2) {**

**return o1.compareToIgnoreCase(o2);**

**}**

**});**

public class Person {

    public enum Sex {

        MALE, FEMALE

    }

    String name;

    LocalDate birthday;

    Sex gender;

    String emailAddress;

    Person(String nameArg, LocalDate birthdayArg,

        Sex genderArg, String emailArg) {

        name = nameArg;

        birthday = birthdayArg;

        gender = genderArg;

        emailAddress = emailArg;

    }

    public int getAge() {

        return birthday

            .until(IsoChronology.INSTANCE.dateNow())

            .getYears();

    }

    public void printPerson() {

      System.out.println(name + ", " + this.getAge());

    }

    public Sex getGender() {

        return gender;

    }

    public String getName() {

        return name;

    }

    public String getEmailAddress() {

        return emailAddress;

    }

    public LocalDate getBirthday() {

        return birthday;

    }

    public static int compareByAge(Person a, Person b) {

        return a.birthday.compareTo(b.birthday);

    }

    public static List<Person> createRoster() {

        List<Person> roster = new ArrayList<>();

        roster.add(

            new Person(

            "Fred",

            IsoChronology.INSTANCE.date(1980, 6, 20),

            Person.Sex.MALE,

            "fred@example.com"));

        roster.add(

            new Person(

            "Jane",

            IsoChronology.INSTANCE.date(1990, 7, 15),

            Person.Sex.FEMALE, "jane@example.com"));

        roster.add(

            new Person(

            "George",

            IsoChronology.INSTANCE.date(1991, 8, 13),

            Person.Sex.MALE, "george@example.com"));

        roster.add(

            new Person(

            "Bob",

            IsoChronology.INSTANCE.date(2000, 9, 12),

            Person.Sex.MALE, "bob@example.com"));

        return roster;

    }

}

МЕТОДЫ

25. Напечатать всех

**public void printAll() {**

**for (Person person : persons) {**

**person.printPerson();**

**}**

**}**

26. Напечатать Sex.MALE

**public void printMale() {**

**persons.stream().filter(person -> person.gender == Person.Sex.MALE).forEach(person -> person.printPerson());**

**}**

27. Напечатать по признаку

**public void printByPredicate(Predicate<Person> condition) {**

**persons.stream().filter(condition).forEach(Person::printPerson);**

**}**

28. Средний возраст

**public int countAverageAge() {**

**return persons.stream().mapToInt(Person::getAge).sum() / persons.size();**

**}**

29. Процент Sex.MALE

**public double countPercentOfMale() {**

**return (double) persons.stream().filter(p -> p.gender == Person.Sex.MALE).count() / persons.size();**

**}**