

Plan

2023-10-25

1. Practice
2. Arrays methods
3. Inner Classes

Theory

► English**▼ На русском**

класс Arrays

В Java класс `Arrays` из пакета `java.util` предоставляет набор статических методов для работы с массивами. Вот некоторые из них:

1. **Сортировка:** `Arrays.sort(array)` сортирует массив в порядке возрастания.
 - `Arrays.sort(int[] a)`: Сортирует целочисленный массив в порядке возрастания.
 - `Arrays.sort(int[] a, int fromIndex, int toIndex)`: Сортирует часть массива от `fromIndex` до `toIndex-1`.
 - `Arrays.sort(Object[] a)`: Сортирует объекты, реализующие интерфейс **Comparable**
 - `public static void sort(Object[] a, int fromIndex, int toIndex)`: Эта перегрузка сортирует часть массива объектов, реализующих интерфейс `Comparable`, от индекса `fromIndex` до `toIndex-1`. Объекты сравниваются на основе их естественного порядка.
 - `public static void sort(T[] a, int fromIndex, int toIndex, Comparator c)`: Эта версия позволяет сортировать часть массива с использованием специального компаратора. `Comparator c` определяет, как будут сравниваться объекты.
 - `public static void sort(T[] a, Comparator c)`: Эта версия сортирует весь массив объектов с использованием заданного компаратора. Это удобно, когда естественный порядок сортировки объектов вам не подходит.
2. **Поиск:** `Arrays.binarySearch(array, value)` выполняет бинарный поиск значения в отсортированном массиве.
3. **Копирование:** `Arrays.copyOf(array, newLength)` создаёт копию массива с новой длиной.

4. **Заполнение:** `Arrays.fill(array, value)` заполняет все элементы массива заданным значением.
5. **Сравнение:** `Arrays.equals(array1, array2)` проверяет, равны ли два массива.
6. **Преобразование в строку:** `Arrays.toString(array)` возвращает строковое представление массива.

Методы `Arrays.copyOf` и `System.arraycopy` оба предназначены для копирования массивов, но есть несколько ключевых различий:

`Arrays.copyOf`:

1. **Создание нового массива:** `Arrays.copyOf` возвращает новый массив, который может иметь другую длину.
2. **Тип возвращаемого массива:** Может быть изменён, если используется перегрузка с параметром типа.
3. **Простота использования:** Очень прост в использовании, так как вам нужно указать только исходный массив и новую длину.

```
int[] original={1,2,3};  
int[] copied=Arrays.copyOf(original,5); // [1, 2, 3, 0, 0]
```

`System.arraycopy`:

1. **Использует существующий массив:** Этот метод не создаёт новый массив, а копирует данные в уже существующий массив.
2. **Больше параметров:** Требуется указания исходного и целевого массивов, позиций в этих массивах и количества копируемых элементов.
3. **Быстродействие:** Обычно быстрее, так как работает напрямую с памятью.

```
int[] original={1,2,3};  
int[] destination=new int[5];  
System.arraycopy(original,0,destination,0,original.length); // dest
```

В общем, `Arrays.copyOf` удобнее и проще в использовании для создания новых массивов, тогда как `System.arraycopy` обычно используется для копирования данных в уже существующие массивы и может быть быстрее в некоторых сценариях.

Перегрузки методов `Arrays.copyOf` и `System.arraycopy`, детально

Методы `Arrays.copyOf` и `System.arraycopy` оба предназначены для копирования массивов, но есть несколько ключевых различий:

Arrays.copyOf:

1. **Создание нового массива:** `Arrays.copyOf` возвращает новый массив, который может иметь другую длину.
2. **Тип возвращаемого массива:** Может быть изменён, если используется перегрузка с параметром типа.
3. **Простота использования:** Очень прост в использовании, так как вам нужно указать только исходный массив и новую длину.

```
int[] original={1,2,3};  
int[] copied=Arrays.copyOf(original,5); // [1, 2, 3, 0, 0]
```

System.arraycopy:

1. **Использует существующий массив:** Этот метод не создаёт новый массив, а копирует данные в уже существующий массив.
2. **Больше параметров:** Требуется указание исходного и целевого массивов, позиций в этих массивах и количества копируемых элементов.
3. **Быстродействие:** Обычно быстрее, так как работает напрямую с памятью.

```
int[] original={1,2,3};  
int[] destination=new int[5];  
System.arraycopy(original,0,destination,0,original.length); // dest
```

В общем, `Arrays.copyOf` удобнее и проще в использовании для создания новых массивов, тогда как `System.arraycopy` обычно используется для копирования данных в уже существующие массивы и может быть быстрее в некоторых сценариях.

Типы вложенных классов

1. Статические Вложенные Классы (Static Nested Classes)

Что это?

Это статический класс, объявленный внутри другого класса. Он имеет доступ только к статическим членам внешнего класса.

Пример кода

```
public class Car {  
    // Статический вложенный класс  
    public static class Wheel {  
        public void rotate() {
```

```
        System.out.println("Wheel is rotating...");  
    }  
}  
}
```

Пример из жизни

Представьте, что у вас есть класс `Car`, и у каждой машины есть колеса. Колеса (`Wheel`) не могут существовать без машины, но их поведение (вращение) не зависит от конкретного экземпляра машины. В этом случае `Wheel` может быть статическим вложенным классом.

Как использовать?

```
Car.Wheel wheel = new Car.Wheel();  
wheel.rotate();
```

2. Внутренние Классы (Inner Classes)

Что это?

Это класс, объявленный внутри другого класса без модификатора `static`. Он имеет доступ ко всем членам внешнего класса, даже к приватным.

Пример кода

```
public class OuterClass {  
    private int x = 10;  
  
    class InnerClass {  
        public void display() {  
            System.out.println("x = " + x);  
        }  
    }  
}
```

Пример из жизни

Представьте объект `Building` (здание), в котором есть объект `Elevator` (лифт). Лифт не может существовать без здания и имеет доступ к его внутренней структуре (например, количеству этажей).

Как использовать?

```
OuterClass outer = new OuterClass();  
OuterClass.InnerClass inner = outer.new InnerClass();
```

```
inner.display();
```

3. Локальные Вложенные Классы (Local Inner Classes)

Что это?

Это класс, объявленный внутри метода.

Пример кода

```
public class OuterClass {  
    public void myMethod() {  
        class LocalInnerClass {  
            public void display() {  
                System.out.println("Inside local inner class");  
            }  
        }  
  
        LocalInnerClass local = new LocalInnerClass();  
        local.display();  
    }  
}
```

Пример из жизни

Допустим, у вас есть метод, который выполняет сложный алгоритм, и вам нужно определить вспомогательный класс для промежуточных вычислений, который нигде больше не нужен. В этом случае этот класс можно определить прямо в методе.

4. Анонимные Вложенные Классы (Anonymous Inner Classes)

Что это?

Это класс без имени, объявленный "на лету", обычно для реализации интерфейсов или наследования классов.

Пример кода

```
new Thread(new Runnable() {  
    public void run() {  
        System.out.println("Thread is running...");  
    }  
}).start();
```

Пример из жизни

Если вам нужно быстро реализовать какой-либо интерфейс (например, для обработки событий в графическом интерфейсе), анонимные классы предлагают быстрый и удобный способ это сделать.

Homework

► English

▼ На русском

Задача 1: Использование статического вложенного класса

Описание:

Создайте класс `University`, внутри которого будет статический вложенный класс `Student`. Вложенный класс должен иметь поля `name`, `age` и `grade` (оценка), а также методы для вывода информации о студенте и изменения его оценки.

Дополнительные условия:

- Создайте во внешнем классе `University` метод для добавления студентов в массив.
- Создайте метод, который выводит информацию о всех студентах с оценкой выше заданной.

Цель:

Познакомиться с созданием и использованием статических вложенных классов, а также с манипуляцией данными во внешнем классе.

Пример использования:

```
public class University {  
    private Student[] students = new Student[10]; // храните студентов в этом массиве  
    private int studentCount = 0; // счетчик студентов. При добавлении студента  
    //code...  
  
    // напишите статический вложенный класс Student с полями name, age, grade и методами  
  
    /**  
     * метод для добавления студентов в массив  
     * @param student  
     */  
    public void addStudent(Student student) {
```

```
// code...
}

/**
 * метод, который выводит информацию о всех студентах с оценкой выше за,
 * @param minGrade
 */
public void printStudentsWithGradeAbove(int minGrade) {
    // code...
}

}

public class Main {

    public static void main(String[] args) {
        University university = new University();
        Student student1 = new Student("Alex", 20, 85);
        Student student2 = new Student("Maria", 21, 90);
        university.addStudent(student1);
        university.addStudent(student2);

        System.out.println("Students with grade above 80:");
        university.printStudentsWithGradeAbove(80);
    }
}
```

Задача 2: Внутренний класс для хранения состояния

Описание:

Создайте класс `BankAccount`, который имеет внутренний класс `Transaction`. Внутренний класс должен хранить информацию о транзакции (сумма, дата, тип транзакции).

Дополнительные условия:

- В классе `BankAccount` создайте методы для внесения и снятия денег, которые также создают объекты транзакций.
- Добавьте возможность просмотра истории транзакций.

Цель:

Углубить понимание внутренних классов и их взаимодействия с внешним классом.

Пример использования:

```
public class BankAccount {

    private Transaction[] transactions = new Transaction[10];
    private int transactionCount = 0;

    // создайте внутренний класс Transaction

    public void deposit(double amount) {
        //code...
        transactions[transactionCount++] = new Transaction(amount, "deposit");
        // code..
    }

    public void withdraw(double amount) {
        // code...
        Transaction transaction = new Transaction(amount, "withdraw");
        ArrayTools.add()
        // code...
    }

    public void printTransactionHistory() {

    }

}

public class Main {

    public static void main(String[] args) {
        BankAccount account = new BankAccount();
        account.deposit(200);
        account.withdraw(50);

        account.printTransactionHistory();
    }
}
```


Задача 3: Анонимные классы для обработки событий

Описание:

Создайте внутренний интерфейс `OnClickListener` с методами `onClick` и `onDoubleClick`.

Создайте класс `Button`, у которого будет метод `setOnClickListener`, принимающий объект типа `OnClickListener`.

Дополнительные условия:

- Реализуйте обработку события "клик" и "двойной клик" с использованием анонимного класса.
- Добавьте возможность отключать слушатель событий.

Цель:

Научиться создавать и использовать анонимные классы для реализации интерфейсов и управления событиями.

```
public class Button {
    private OnClickListener[] listeners = new OnClickListener[10];
    private int listenerCount = 0;

    // создайте внутренний интерфейс OnClickListener с двумя методами onClick

    public void setClickListener(OnClickListener listener) {
        if (listenerCount < listeners.length) {
            listeners[listenerCount++] = listener;
        }
    }

    public void removeClickListener() {
        listenerCount = 0;
    }

    public void simulateClick() {
        for (int i = 0; i < listenerCount; i++) {
            listeners[i].onClick();
        }
    }

    public void simulateDoubleClick() {
        for (int i = 0; i < listenerCount; i++) {
            listeners[i].onDoubleClick();
        }
    }
}
```

```
}

public class Main{
public static void main(String[] args) {
    Button button = new Button();

    button.setOnClickListener(
        // code here...
    );

    button.simulateClick();
    button.simulateDoubleClick();
    button.removeClickListener();
}
}
```

Code

code/Hw_Solution_35/src/nested/inner_class/Library.java

```
package nested.inner_class;

import java.util.Objects;

/**
 * @author Andrej Reutow
 * created on 25.10.2023
 */

// Здесь Book является внутренним классом Shelf, Shelf является внутренним
public class Library {

    private String name;

    public Library(String name) {
        this.name = name;
    }

    private String getLName() {
        return this.name;
    }
}
```

```
}

// Shelf является внутренним классом в Library
public class Shelf {
    private String name;

    public Shelf(String name) {
        this.name = name;
    }

    // Book является внутренним классом Shelf
    public class Book {
        private String title;

        public Book(String title) {
            this.title = title;
        }

        public void printInfo() {
            System.out.println("Book title: " + this.title + ", Shelf:
        }
    }
}

@Override
public boolean equals(Object object) {
    if (this == object) return true;
    if (object == null || getClass() != object.getClass()) return false

    Library library = (Library) object;

    return Objects.equals(name, library.name);
}

@Override
public int hashCode() {
    return name != null ? name.hashCode() : 0;
}

@Override
public String toString() {
    return "Library{" +
```

```
        "name='" + name + '\\ ' +  
        '}'';  
    }  
}
```

code/Hw_Solution_35/src/nested/inner_class/Main.java

```
package nested.inner_class;  
  
/**  
 * @author Andrej Reutow  
 * created on 25.10.2023  
 */  
public class Main {  
  
    public static void main(String[] args) {  
        Library library = new Library("Humboldt");  
        Library.Shelf shelf1 = library.new Shelf("ID-1");  
        Library.Shelf.Book book1 = shelf1.new Book("Some title");  
  
        book1.printInfo();  
    }  
}
```


code/Hw_Solution_35/src/nested/local_inner_classes/TaskManager.java

```
package nested.local_inner_classes;  
  
/**  
 * @author Andrej Reutow  
 * created on 25.10.2023  
 */  
// Здесь Task является локальным вложенным классом в методе executeTask().  
public class TaskManager {  
    public void executeTask(String taskName) {  
        class Task {  
            void run() {  
                System.out.println(taskName + " is running.");  
            }  
        }  
  
        new Task().run();  
    }  
}
```

```
    public static void main(String[] args) {  
        new TaskManager().executeTask("FileDownload");  
    }  
}
```

code/Hw_Solution_35/src/nested/static_nested_classes/Calculator.java

```
package nested.static_nested_classes;  
  
import tools.Id;  
  
/**  
 * @author Andrej Reutow  
 * created on 25.10.2023  
 */  
// Здесь Operation является статическим вложенным классом в Calculator, кото  
public class Calculator {  
  
    // Operation - статический вложенный класс  
    public static class Operation {  
  
        public static int add(int a, int b) {  
            return a + b;  
        }  
  
        public static int subtract(int a, int b) {  
            return a - b;  
        }  
    }  
}
```



code/Hw_Solution_35/src/nested/static_nested_classes/Main.java

```
package nested.static_nested_classes;  
  
/**  
 * @author Andrej Reutow  
 * created on 25.10.2023  
 */  
public class Main {  
  
    public static void main(String[] args) {
```

```
int add = Calculator.Operation.add(1, 2);
int subtract = Calculator.Operation.subtract(2, 1);

System.out.println(add);
System.out.println(subtract);
}
}
```

code/Hw_Solution_35/src/tests/ArrayToolsTest.java

```
package tests;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import tools.ArrayTools;
import tools.Id;

import static org.junit.jupiter.api.Assertions.*;

/**
 * @author Andrej Reutow
 * created on 25.10.2023
 */
class ArrayToolsTest {

    class Car implements Id {
        private int id;

        public Car(int id) {
            this.id = id;
        }

        @Override
        public long getId() {
            return this.id;
        }

        @Override
        public boolean equals(Object object) {
            if (this == object) return true;
            if (object == null || getClass() != object.getClass()) return false;

```

```
        Car car = (Car) object;

        return id == car.id;
    }

    @Override
    public int hashCode() {
        return id;
    }

    @Override
    public String toString() {
        return "Car{" +
            "id=" + id +
            '}';
    }
}

class Dog implements Id {
    private int id;

    public Dog(int id) {
        this.id = id;
    }

    @Override
    public long getId() {
        return this.id;
    }

    @Override
    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null || getClass() != object.getClass()) return false;

        Dog dog = (Dog) object;

        return id == dog.id;
    }

    @Override
    public int hashCode() {
```

```
        return id;
    }

    @Override
    public String toString() {
        return "Dog{" +
            "id=" + id +
            '}';
    }
}

@Nested
@DisplayName("Tests search entity by id")
class TestSearchById {
    @Test
    public void test_searchById_findEntityCar_entityFound() {
        Car entity1 = new Car(1);
        Car entity2 = new Car(2);
        Car[] cars = {entity1, entity2};

        Car resultFindCarById1 = ArrayTools.searchById(cars, 1);

        assertEquals(entity1, resultFindCarById1);
    }

    @Test
    public void test_searchById_findEntityDog_entityFound() {
        Dog entity1 = new Dog(1);
        Dog entity2 = new Dog(2);

        Dog[] dogs = {entity1, entity2};

        Dog resultFindCarById1 = ArrayTools.searchById(dogs, 1);

        assertEquals(entity1, resultFindCarById1);
    }

    @Test
    public void test_searchById_findEntityDog_entityNotFound() {
        Dog entity1 = new Dog(1);
        Dog entity2 = new Dog(2);

        Dog[] dogs = {entity1, entity2};
```



```
        Dog resultFindCarById1 = ArrayTools.searchById(dogs, 10);

        assertNull(resultFindCarById1);
    }

    @Test
    public void test_searchById_findEntityDogWhenArrayEmpty_entityNotFo
        Dog[] dogs = new Dog[0];

        Dog resultFindCarById1 = ArrayTools.searchById(dogs, 10);

        assertNull(resultFindCarById1);
    }
}

@Nested
@DisplayName("Tests search entity by entity")
class TestSearchByEntity {
    @Test
    public void test_search_findEntityCar_entityFound() {
        Car entity1 = new Car(1);
        Car entity2 = new Car(2);
        Car[] cars = {entity1, entity2};

        Car resultFindCarById1 = ArrayTools.search(cars, entity1);

        assertEquals(entity1, resultFindCarById1);
    }

    @Test
    public void test_search_findEntityDog_entityFound() {
        Dog entity1 = new Dog(1);
        Dog entity2 = new Dog(2);

        Dog[] dogs = {entity1, entity2};

        Dog resultFindCarById1 = ArrayTools.search(dogs, entity1);

        assertEquals(entity1, resultFindCarById1);
    }

    @Test
```

```
public void test_search_findEntityDog_entityNotFound() {
    Dog entity1 = new Dog(1);
    Dog entity2 = new Dog(2);
    Dog entity3 = new Dog(3);

    Dog[] dogs = {entity1, entity2};

    Dog resultFindCarById1 = ArrayTools.search(dogs, entity3);

    assertNull(resultFindCarById1);
}

@Test
public void test_search_findEntityDogWhenArrayEmpty_entityNotFound(
    Dog entity1 = new Dog(1);

    Dog[] dogs = new Dog[0];

    Dog resultFindCarById1 = ArrayTools.search(dogs, entity1);

    assertNull(resultFindCarById1);
}

}

@Nested
@DisplayName("Tests remove entity by id")
class TestRemoveById {
    @Test
    public void test_removeById_removeCar_entityFound() {
        Car entity1 = new Car(1);
        Car entity2 = new Car(2);
        Car[] cars = {entity1, entity2};

        boolean resultRemovedCarById1 = ArrayTools.removeById(cars, 1);

        assertTrue(resultRemovedCarById1);
        assertNull(cars[0]);
    }

    @Test
    public void test_removeById_removeDog_entityFound() {
        Dog entity1 = new Dog(1);
        Dog entity2 = new Dog(2);
```

```
Dog[] dogs = {entity1, entity2};

boolean resultRemovedDogById1 = ArrayTools.removeById(dogs, 1);

assertTrue(resultRemovedDogById1);
assertNull(dogs[0]);
}

@Test
public void test_removeById_removeDog_entityNotFound() {
    Dog entity1 = new Dog(1);
    Dog entity2 = new Dog(2);

    Dog[] dogs = {entity1, entity2};

    boolean resultRemoveDogById1 = ArrayTools.removeById(dogs, 10);

    assertFalse(resultRemoveDogById1);
}

@Test
public void test_removeById_removeDogWhenArrayEmpty_entityNotFound(
    Dog[] dogs = new Dog[0];

    boolean resultRemoveDogById1 = ArrayTools.removeById(dogs, 1);

    assertFalse(resultRemoveDogById1);
}

@Test
public void test_removeById_removeElementWhenArrayHasNullElement_en
    Dog entity1 = new Dog(1);
    Dog entity2 = new Dog(2);

    Dog[] dogs = {null, entity1, entity2};

    boolean resultRemoveDogById1 = ArrayTools.removeById(dogs, 2);

    assertTrue(resultRemoveDogById1);
    assertNull(dogs[2]);
}
```

```
}
```

```
@Nested
```

```
@DisplayName("Tests remove entity by entity")
```

```
class TestRemove {
```

```
    @Test
```

```
    public void test_remove_removeCar_entityFound() {
```

```
        Car entity1 = new Car(1);
```

```
        Car entity2 = new Car(2);
```

```
        Car[] cars = {entity1, entity2};
```

```
        boolean resultRemovedCarById1 = ArrayTools.remove(cars, entity1
```

```
        assertTrue(resultRemovedCarById1);
```

```
        assertNull(cars[0]);
```

```
    }
```

```
    @Test
```

```
    public void test_remove_removeDog_entityFound() {
```

```
        Dog entity1 = new Dog(1);
```

```
        Dog entity2 = new Dog(2);
```

```
        Dog[] dogs = {entity1, entity2};
```

```
        boolean resultRemovedDogById1 = ArrayTools.remove(dogs, entity1
```

```
        assertTrue(resultRemovedDogById1);
```

```
        assertNull(dogs[0]);
```

```
    }
```

```
    @Test
```

```
    public void test_remove_removeDog_entityNotFound() {
```

```
        Dog entity1 = new Dog(1);
```

```
        Dog entity2 = new Dog(2);
```

```
        Dog entity3 = new Dog(3);
```

```
        Dog[] dogs = {entity1, entity2};
```

```
        boolean resultRemoveDogById1 = ArrayTools.remove(dogs, entity3)
```

```
        assertFalse(resultRemoveDogById1);
```

```
    }
```

```
@Test
public void test_remove_removeDogWhenArrayEmpty_entityNotFound() {
    Dog entity1 = new Dog(1);

    Dog[] dogs = new Dog[0];

    boolean resultRemoveDogById1 = ArrayTools.remove(dogs, entity1)

    assertFalse(resultRemoveDogById1);
}

@Test
public void test_remove_removeNullElement_entityNotFound() {
    Dog entity1 = new Dog(1);
    Dog entity2 = new Dog(2);

    Dog[] dogs = {entity1, entity2};

    boolean resultRemoveDogById1 = ArrayTools.remove(dogs, null);

    assertFalse(resultRemoveDogById1);
}

@Test
public void test_remove_removeElementWhenArrayHasNullElement_entity
    Dog entity1 = new Dog(1);
    Dog entity2 = new Dog(2);

    Dog[] dogs = {null, entity1, entity2};

    boolean resultRemoveDogById1 = ArrayTools.remove(dogs, entity1)

    assertTrue(resultRemoveDogById1);
    assertNull(dogs[1]);
}
}
}
```

code/Hw_Solution_35/src/tools/ArrayTools.java

```
package tools;
```

```
/**
 * @author Andrej Reutow
 * created on 24.10.2023
 */
public class ArrayTools {

    private ArrayTools() {
    }

    public static <T> void print(T[] array) {
        for (int i = 0; i < array.length; i++) {
            System.out.println(array[i]);
        }
    }

    public static <T> T search(T[] source, T value) {
        for (int i = 0; i < source.length; i++) {
            if (source[i].equals(value)) {
                return source[i];
            }
        }
        return null;
    }

    public static <T extends Id> T searchById(T[] source, long id) {
        for (int i = 0; i < source.length; i++) {
            if (id == source[i].getId()) {
                return source[i];
            }
        }
        return null;
    }

    public static <T> boolean remove(T[] source, T value) {
        for (int i = 0; i < source.length; i++) {
            if (value != null && value.equals(source[i])) {
                if (source[i] != null && source[i].equals(value)) {
                    source[i] = null;
                    return true;
                }
            }
        }
        return false;
    }
}
```

```

    public static <T extends Id> boolean removeById(T[] source, long id) {
        for (int i = 0; i < source.length; i++) {
            if (source[i] != null && source[i].getId() == id) {
                source[i] = null;
                return true;
            }
        }
        return false;
    }
}

```

code/Hw_Solution_35/src/tools/Id.java

```

package tools;

/**
 * @author Andrej Reutow
 * created on 24.10.2023
 */
public interface Id {

    long getId();
}

```

code/Hw_Solution_35/src/Main.java

```

import tools.ArrayTools;
import java.util.Arrays;
import java.util.function.Predicate;

import static tools.ArrayTools.*;

/**
 * @author Andrej Reutow
 * created on 25.10.2023
 */

//Задание 4: Обобщенный метод с предикатами
//      Создайте обобщенный метод filterArray, который принимает массив и
//      в котором останутся только элементы, удовлетворяющие условию пред
//      Сигнатура метода может выглядеть так: <T> T[] filterArray(T[] arr;
//      Создайте два разных массива для тестирования: один с целыми числами

```

```
//      Например, массив целых чисел [1, 2, 3, 4, 5] и массив строк ["app
//      Определите предикаты для каждого типа массивов:
//      Для массива чисел предикат, который фильтрует четные числа.
//      Для массива строк предикат, который фильтрует строки, начинающиес
//      Примените filterArray к каждому из массивов, используя определенн

//      Выведите результаты на экран.
//      Ожидаемый результат
//      На экране должны быть выведены отфильтрованные массивы: один с чи
//      и один со строками, которые удовлетворяют предикату.
//
//      Будьте внимательны с типами данных. Обобщенный метод должен работ
public class Main {

    public static void main(String[] args) {
        Integer[] integers = {6, 1, 2, 3, 4, 5};
        String[] strings = {"apple", "banana", "cherry"};

        Predicate<Integer> integerPredicate = number -> number % 2 == 0;
        Predicate<String> stringPredicate = text -> text.startsWith("a");

        Integer[] resultIntegers = filterArray(integers, integerPredicate);
        String[] resultStrings = filterArray(strings, stringPredicate);

        print(resultIntegers);
        System.out.println();
        print(resultStrings);

        System.out.println(search(strings, "wien"));
    }

    private static <T> T[] filterArray(T[] array, Predicate<T> predicate) {
        int counter = 0;

        for (int i = 0; i < array.length; i++) {
            if (predicate.test(array[i])) {
                counter++;
            }
        }

        // вернуть отфильтрованный массив только с теми элементами которые
        T[] result = Arrays.copyOf(array, counter);
```



```
    for (int i = 0, j = 0; j < result.length; i++) {  
        if (predicate.test(array[i])) {  
            result[j++] = array[i];  
        }  
    }  
    return result;  
}
```