Plan

2023-10-19

1. Generic
2. Comparator
3. Comparable

Theory

▶ **English**

▼ **На русском**

Генерики

Генерики (generics) в Java представляют собой механизм, который позволяет создавать **классы**, **интерфейсы** и **методы**, которые работают с параметризованными типами данных. Они позволяют написать код, который будет работать с разными типами данных, обеспечивая типовую безопасность. Типовая безопасность гарантирует, что вы не сможете вставить объект несовместимого типа данных.

Пример использования генериков для классов:

```
public class Box<T> {
    private T content;

    public Box(T content) {
        this.content = content;
    }

    public T getContent() {
        return content;
    }
}
```

В этом примере `T` является параметром типа (type parameter), который может быть заменен на конкретный тип данных при создании объекта класса `Box`. Например:

```
Box<Integer> integerBox=new Box<>(42);
Box<String> stringBox=new Box<>("Привет, мир!");
```

```
Integer intValue=integerBox.getContent();
String stringValue=stringBox.getContent();
```

- Использование обобщений делает код более гибким и безопасным.

Примеры использования гейнериков для методов:

```
public class GenericExample {
    public static <T> void printArray(T[] array) {
        for (T item : array) {
            System.out.print(item + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Integer[] intArray = {1, 2, 3, 4, 5};
        String[] stringArray = {"Привет", "Мир", "Java"};

        System.out.println("Массив целых чисел:");
        printArray(intArray);

        System.out.println("Массив строк:");
        printArray(stringArray);
    }
}
```

Практика:

- Напишите обобщенный интерфейс для калькулятора и несколько имплементаций с разными типами (Short, Double, Integer, Long) для интерфейса.

Пример:

```
public class CalculatorImplDouble implements ICalculator<Double, Float> {

    @Override
    public Double add(Float a, Float b) {
        return a + b;
    }

    //...
}
```

```
public class CalculatorImplDouble implements ICalculator<Long, Integer> {

    @Override
    public Long add(Integer a, Integer b) {
        return a + b;
    }

    //...
}
```

- Напишите обобщенный метод `printArray`, который принимает массив любого типа данных и выводит его содержимое на экран. Затем создайте два массива - один с целыми числами, другой с строками, и объектом Car и используйте `printArray` для их вывода

```
public class Car {

    /**
     * Статический счетчик. В этом примере используется для автоматической у
     */
    private static int carIdCounter = 0;

    private final Integer ID; // Константа/финальная переменная. Id не долж
    private String brand;
    private String model;

    public Car(String brand,
               String model) {
        carIdCounter++; // Добавляем +1 к каждому новому созданному объекту
        this.ID = carIdCounter; // присвоение значения для ID на основе сче
        this.brand = brand;
        this.model = model;
    }
}
```

```
public class GenericExample {

    public static void main(String[] args) {
        Integer[] intArray = {1, 2, 3, 4, 5};
        String[] stringArray = {"Привет", "Мир", "Java"};
```

```
        Car[] carArray = {new Car("Toyota", "Camry"), new Car("Honda", "Civ

        System.out.println("Массив целых чисел:");
        printArray(intArray); // вызов обобщенного метода

        System.out.println("Массив строк:");
        printArray(stringArray); // вызов обобщенного метода

        System.out.println("Массив машин:");
        printArray(carArray); // вызов обобщенного метода
    }
}
```

Интерфейсы Comparator и Comparable

`Comparable` и `Comparator` - это интерфейсы, которые позволяют **сравнивать** объекты в Java.

- `Comparable` позволяет определить **естественный порядок** сравнения для объектов данного класса.
- `Comparator` предоставляет возможность определить **несколько способов сравнения** для одного класса.

Использование Comparable

Чтобы использовать `Comparable`, класс должен реализовать этот интерфейс и переопределить метод `compareTo()`. Пример:

```
public class Person implements Comparable<Person> {
    private String name;
    private int age;

    // Конструкторы, геттеры, сеттеры

    @Override
    public int compareTo(Person other) {
        return this.age - other.age;
    }
}
```

Использование Comparator

`Comparator` позволяет создавать собственные правила сравнения для объектов. Пример:

```java
public class PersonAgeComparator implements Comparator<Person> {
    @Override
    public int compare(Person p1, Person p2) {
        return p1.getAge() - p2.getAge();
    }
}
```

Чем отличается в Java Comparator от Comparable?

В Java интерфейсы `Comparator` и `Comparable` используются для сравнения объектов, но они имеют разные назначения:

1. `Comparable`:

- Интерфейс `Comparable` позволяет объекту сравнивать себя с другим объектом.
- Класс, реализующий `Comparable`, предоставляет метод `compareTo`, который определяет, как один объект сравнивается с другим.
- Объекты, реализующие `Comparable`, могут быть естественно упорядочены, например, числа или строки, их можно сравнивать с использованием метода `compareTo`.

1. `Comparator`:

- Интерфейс `Comparator` представляет собой внешний компаратор, который можно использовать для сравнения объектов, даже если класс сам по себе не реализует `Comparable`.
- Вы можете создать различные `Comparator` для сортировки объектов по разным критериям.
- `Comparator` реализует метод `compare`, который принимает два объекта и определяет, какой из них должен идти первым.

Важно помнить, что `Comparable` определяет естественный порядок сравнения для объектов данного класса, в то время как `Comparator` позволяет создавать пользовательские способы сравнения объектов.

Практические задачи

1. Создайте класс `Student` с полями `name` и `grade`. Реализуйте для него интерфейс `Comparable` так, чтобы студенты сортировались по оценкам.
2. Создайте список студентов и отсортируйте его с использованием `Comparator` и `Comparable`.

Homework

▶ **English**

▼ **На русском**

Задача 1

Напишите интерфейс для CRUD (C - Creat, R - Read, U - Update, D - Delete) операций. Интерфейс должен быть обобщенным

Пример не обобщенного интерфейса:

```
public interface ICrudService { // не обобщенный интерфейс с CRUD операциям

    Car add();

    Car get(Long id);

    Car[] getAll();

    Car edit(Long id);

    boolean remove(Long id);

}
```

- Создайте package entity и внутри этого package создайте эти классы:
  - Book с полями: int id, String title
  - Animal, с полями: long id, String color
- Создайте несколько имплементаций обобщенного интерфейса для классов: Book, Animal.
  - **!!! Опционально** - реализуйте методы в классах имплементаций

В итоге у вас должны получится следующие методы (*Achtung - это читы*):

▼ **Для класса Book**

```
private final Book[] SOURCE = new Book[10]; // SOURCE объявите явно без обобщения в к

public Book add(){
     // code hier
 }

public Book get(Integer id){
     // code hier
 }
```

```
public Book[] getAll(){
     // code hier
 }

public Book edit(Integer id){
     // code hier
 }

public boolean remove(Integer id){
     // code hier
```

▼ **Для класса Animal**

```
private final Animal[] SOURCE = new Animal[10]; // SOURCE объявите явно без обобщения

public Animal add(){
     // code hier
 }

public Animal get(Long id){
     // code hier
 }

public Animal[] getAll(){
     // code hier
 }

public Animal edit(Long id){
     // code hier
 }

public boolean remove(Long id){
     // code hier
 }
```

Важно, используйте **один** интерфейс для разных классов и используйте **гейнерики** для обобщения!

Задача 2:

Создайте класс `Student` с полями `name` и `grade`. Реализуйте для него интерфейс `Comparable` так, чтобы студенты сортировались по оценкам.

- создайте массив студентов и отсортируйте используя `Arrays.sort()`

▼ **Напоминалка**

```
// Сортируем массив студентов, для сортировки используется имлементация интерфейс
 Arrays.sort(studends);
```

Задача 3:

Домашнее задание: Создайте класс `Product`, представляющий товар с полями `id`, `name` и `price`.

- Напишите `Comparator`, который сравнивает товары по цене.
- Напишите `Comparator`, который сравнивает товары по id.
- создайте массив товаров и отсортируйте его по цене используя `Arrays.sort()`
- отсортируйте массив его по id `Arrays.sort()`

Для сортировки используйте свои компараторы.

▼ **Напоминалка**

```
// Сортируем массив товаров по цене
 Arrays.sort(products, priceComparator);

// Сортируем массив товаров по id
 Arrays.sort(products, idComparator);
```

Задача 4** (Очень сложная!!!)

▶ **Описание задачи:**

Code

code/ClassWork_32/src/class_work/Box.java

```
package class_work;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */

// обобобщенный класс
public class Box<T> { // <T, R, Q> - generics

    private T value; // Box<CAR> -> T -> Car
```

```
    public Box(T value) { // Box<CAR> -> T -> Car
        this.value = value;
    }

    public T getContent() { // Box<CAR> ->  T -> Car
        return this.value;
    }

}
```

code/ClassWork_32/src/class_work/BoxAppRunner.java

```
package class_work;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class BoxAppRunner {

    public static void main(String[] args) {
        BoxSimple boxSimpleInteger = new BoxSimple(1);
        BoxSimpleString boxSimpleString = new BoxSimpleString("Hello World!

        Integer boxIntResult = boxSimpleInteger.getContent();
        String boxStringResult = boxSimpleString.getContent();

        Box boxO = new Box(5); // обобщенный класс Box без использования ge
        Box boxS = new Box("Hello");
        Box boxC = new Box('C');

        System.out.println(boxO.getContent() instanceof Integer);
        System.out.println(boxS.getContent() instanceof String);
        System.out.println(boxC.getContent() instanceof Character);

        Box<Car> boxCar = new Box<>(new Car("TT"));

        System.out.println(boxCar.getContent() instanceof Car);

        Car unboxedCar = boxCar.getContent();
        unboxedCar.getModel();

        // перепишете BoxSimple, BoxSimpleString используя generic и класс
```

```
        Box<Integer> integerBox = new Box<>(9);
        Box<String> stringBox = new Box<>("String");
        Integer intResult = integerBox.getContent();
        String resStr = stringBox.getContent();
    }
}
```

code/ClassWork_32/src/class_work/BoxSimple.java

```
package class_work;

public class BoxSimple {

    private Integer value;

    public BoxSimple(Integer value) {
        this.value = value;
    }

    public Integer getContent() {
        return this.value;
    }
}
```

code/ClassWork_32/src/class_work/BoxSimpleString.java

```
package class_work;

public class BoxSimpleString {

    private String value;

    public BoxSimpleString(String value) {
        this.value = value;
    }

    public String getContent() {
        return this.value;
    }
}
```

code/ClassWork_32/src/class_work/Car.java

```java
package class_work;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class Car {

    private String model;

    public Car(String model) {
        this.model = model;
    }

    public String getModel() {
        return model;
    }
}
```

code/ClassWork_32/src/interface_generic/ConcatIntegerAsString.java

```java
package interface_generic;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */

// эта реализация интерфейса ICalculator, должна принимать тип Integer, воз
public class ConcatIntegerAsString implements ICalculator<Integer, String>

    @Override
    public String add(Integer value1, Integer value2) {
        return value1.toString() + value2.toString();
    }
}
```

code/ClassWork_32/src/interface_generic/ConcatStrings.java

```java
package interface_generic;
```

```java
import comparator.entity.Car;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class ConcatStrings implements ICalculator<String, String> {
    @Override
    public String add(String value1, String value2) {
        return value1 + value2;
    }


}
```

code/ClassWork_32/src/interface_generic/DoubleCalculator.java

```java
package interface_generic;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class DoubleCalculator implements ICalculator<Double, Integer> {


    @Override
    public Integer add(Double value1, Double value2) {
        return (int) (value1 + value1);
    }
}
```

code/ClassWork_32/src/interface_generic/ICalculator.java

```java
package interface_generic;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public interface ICalculator<T, R> {
```

```
    R add(T value1, T value2);
}
```

code/ClassWork_32/src/interface_generic/ShortCalculator.java

```
package interface_generic;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class ShortCalculator implements ICalculator<Short, Integer> {
    @Override
    public Integer add(Short value1, Short value2) {
        return value1 + value2;
    }
}
```

code/ClassWork_32/src/interface_generic/ICalculator.java

```
package interface_generic;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public interface ICalculator<T, R> {

    R add(T value1, T value2);
}
```

code/ClassWork_32/src/comparable/entity/Person.java

```
package comparable.entity;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class Person implements Comparable<Person> {

    private String name;
    private int age;
```

```
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public int compareTo(Person other) {
        // Сравнение объектов происходит по одному их из полей. В нашем при
        // если текущий объект больше other то вернуть положительное число
        // если текущий объект меньше other то вернуть отрицательное число
        // если объект равны то вернуть 0
        return this.age - other.age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

code/ClassWork_32/src/comparable/PersonApp.java

```
package comparable;

import comparable.entity.Person;

import java.util.Arrays;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class PersonApp {

    public static void main(String[] args) {
        Person[] peoples = {
                new Person("Andrej", 20),
                new Person("Vasja", 18),
                new Person("Petja", 34),
                new Person("Nastja", 18)
        };
```

```
        // {Vasja, Nastja, Andrej, Petja}

        for (int i = 0; i < peoples.length; i++) {
            System.out.println(peoples[i].toString());
        }
        Arrays.sort(peoples);
        for (int i = 0; i < peoples.length; i++) {
            System.out.println(peoples[i]);
        }
    }
}
```

code/ClassWork_32/src/comparator/car_comparator/CarIdComparator.java

```
package comparator.car_comparator;

import comparator.entity.Car;

import java.util.Comparator;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class CarIdComparator implements Comparator<Car> {

    @Override
    public int compare(Car o1, Car o2) {
        // Сравнение объектов происходит по одному из их полей. В нашем при
        // если o1 объект больше o2 то вернуть положительное число
        // если o1 объект меньше o2 то вернуть отрицательное число
        // если объект равны то вернуть 0
        return o1.getId() - o2.getId();
    }
}
```

code/ClassWork_32/src/comparator/car_comparator/CarModelComparator.java

```
package comparator.car_comparator;
```

```java
import comparator.entity.Car;

import java.util.Comparator;


/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class CarModelComparator implements Comparator<Car> {
    @Override
    public int compare(Car o1, Car o2) {
        String modelObj1 = o1.getModel();
        String modelObj2 = o2.getModel();
        return modelObj1.compareTo(modelObj2);
    }
}
```

code/ClassWork_32/src/comparator/car_comparator/CarPowerComparator.java

```java
package comparator.car_comparator;

import comparator.entity.Car;

import java.util.Comparator;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class CarPowerComparator implements Comparator<Car> {

    @Override
    public int compare(Car o1, Car o2) {
        // Сравнение объектов происходит по одному из их полей. В нашем при
        // если o1 объект больше o2 то вернуть положительное число
        // если o1 объект меньше o2 то вернуть отрицательное число
        // если объект равны то вернуть 0
        return o1.getPower() - o2.getPower();
    }
}
```

code/ClassWork_32/src/comparator/entity/Car.java

```java
package comparator.entity;

import java.util.Comparator;
import java.util.concurrent.Callable;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class Car {

    private final int id;
    private final String model;
    private final int power;


    public Car(int id, String model, int power) {
        this.id = id;
        this.model = model;
        this.power = power;
    }

    public int getId() {
        return this.id;
    }

    public String getModel() {
        return model;
    }

    public int getPower() {
        return power;
    }

    @Override
    public String toString() {
        return "Car{" +
                "id=" + id +
                ", model='" + model + '\'' +
                ", power=" + power +
                '}';
```

```
    }
}
```

code/ClassWork_32/src/comparator/ComparatorRunner.java

```
package comparator;

import comparator.car_comparator.CarIdComparator;
import comparator.car_comparator.CarModelComparator;
import comparator.car_comparator.CarPowerComparator;
import comparator.entity.Car;

import java.util.Arrays;
import java.util.Comparator;

/**
 * @author Andrej Reutow
 * created on 19.10.2023
 */
public class ComparatorRunner {

    public static void main(String[] args) {
        Car[] cars = {
                new Car(4, "Mondeo", 124),
                new Car(3, "Vesta", 120),
                new Car(1, "S-500", 500),
                new Car(5, "TT", 221),
                new Car(2, "ID-4", 345)
        };

        Comparator<Car> carIdComparator = new CarIdComparator();
        Arrays.sort(cars, carIdComparator);

        System.out.println("Sort Car Array by ID:");
        for (int i = 0; i < cars.length; i++) {
            System.out.print(cars[i] + " ");
        }
        System.out.println();

        Arrays.sort(cars, new CarPowerComparator());
        System.out.println("Sort Car Array by Power:");
        for (int i = 0; i < cars.length; i++) {
            System.out.print(cars[i] + " ");
        }
```

```
        System.out.println();


        CarModelComparator carModelComparator = new CarModelComparator();
        Arrays.sort(cars, carModelComparator);
        System.out.println("Sort Car Array by Model:");
        for (int i = 0; i < cars.length; i++) {
            System.out.print(cars[i] + " ");
        }
        System.out.println();
    }
}
```