Plan

2023-10-18

1. Homework Review
2. Generic

---

1. Разбор домашнего задания
2. Generic

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.

Вот простой пример использования генериков:

```
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();
```

Теперь давайте создадим задачу для тренировки:

Задача: 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 используйте `printArray` для их вывода.

```
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; // коснтанта/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", "Civic")

      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
    }
}
```

Homework

▶ **English**

▼ **На русском**

Задача

Шаг 1 - повторите самостоятельно все те шаги, которые мы делали в классе на примере класса `Book` и интерфейса `Library`.

- создаем класс `Book` с полями `ISBN`, `автор`, `название книги`, `год издания`;
- создаем интерфейс `Library` с методами:
- добавить книгу;
- удалить книгу;
- найти книгу;
- кол-во книг;
- напечатать список книг.
- создаем класс `LibraryImpl`, который `implements Library`;
- создаем класс `LibraryImplTest`, в котором создаем тесты для вышеперечисленных методов.

Code

../lesson_30/code/Classwork_30/src/entity/BaseEmployee.java

```java
package entity;

import java.util.Calendar;

// Абстрактный класс BaseEmployee
public abstract class BaseEmployee implements Employee {
    private String name;
    private Integer id; // null
    private int hireYear;
    private double salary;

    public BaseEmployee(String name, int hireYear) {
        this.name = name;
        this.hireYear = hireYear;
    }

    @Override
    public String getName() {
        return name;
    }

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

    @Override
    public void setId(Integer id) {
        this.id = id;
    }

    public int getHireYear() {
        return hireYear;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
```

```
    public void adjustSalaryByExperience(int minExperience, int maxExperien
        // todo
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("BaseEmployee{");
        sb.append("name='").append(name).append('\'');
        sb.append(", id=").append(id);
        sb.append(", hireYear=").append(hireYear);
        sb.append(", salary=").append(salary);
        sb.append('}');
        return sb.toString() + " ";
    }
}
```

../lesson_30/code/Classwork_30/src/entity/Developer.java

```
package entity;

// Класс Developer
public class Developer extends BaseEmployee {
    private double hourlyRate;
    private int hoursWorked;

    public Developer(String name, double hourlyRate, int hoursWorked, int h
        super(name, hireYear);
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }

    @Override
    public double calculateSalary() {
        return hourlyRate * hoursWorked;
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Developer{");
        sb.append("hourlyRate=").append(hourlyRate);
        sb.append(", hoursWorked=").append(hoursWorked);
        sb.append('}');
        return super.toString() + sb;
```

```
    }
}
```

../lesson_30/code/Classwork_30/src/entity/Employee.java

```
package entity;

// Интерфейс Employee
public interface Employee {
    String getName();

    int getId();

    double calculateSalary();

    void setId(Integer id);
}
```

../lesson_30/code/Classwork_30/src/entity/Manager.java

```
package entity;

// Класс Manager
public class Manager extends BaseEmployee {
    private double baseSalary;
    private int numberOfProjects;

    public Manager(String name, double baseSalary, int numberOfProjects, in
        super(name, hireYear);
        this.baseSalary = baseSalary;
        this.numberOfProjects = numberOfProjects;
    }

    @Override
    public double calculateSalary() {
        return baseSalary + (numberOfProjects * 1000);
    }


    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Manager{");
        sb.append("baseSalary=").append(baseSalary);
        sb.append(", numberOfProjects=").append(numberOfProjects);
```

```
            sb.append('}');
            return super.toString() + sb;
        }
    }
```

../lesson_30/code/Classwork_30/src/entity/Salesperson.java

```
package entity;

// Класс Salesperson
public class Salesperson extends BaseEmployee {
    private double baseSalary;
    private int numberOfDeals;

    public Salesperson(String name, double baseSalary, int numberOfDeals, i
        super(name, hireYear);
        this.baseSalary = baseSalary;
        this.numberOfDeals = numberOfDeals;
    }

    @Override
    public double calculateSalary() {
        return baseSalary + (numberOfDeals * 200);
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Salesperson{");
        sb.append("baseSalary=").append(baseSalary);
        sb.append(", numberOfDeals=").append(numberOfDeals);
        sb.append('}');
        return super.toString() + sb;
    }
}
```

../lesson_30/code/Classwork_30/src/repo/EmployeeRepository.java

```
package repo;

import entity.BaseEmployee;

public class EmployeeRepository implements EmployeeRepositoryInterface {
```

```
    private BaseEmployee[] employees = new BaseEmployee[10]; // Массив для 
    private int size = 0; // количесвто работников
    private static int counterId = 46985;

    // employees {1, 2 ,3, null, null, ...  } // size = 3
    // employees {1, 2 ,3 }

    public boolean addEmployee(BaseEmployee employee) {
        if (size < employees.length) {
            employees[size] = employee; // employees[3] = employees {1, 2 ,
            // size = 2
            // {1, 2, null, 4, null, ...  }
            // employees[2] = 5
            // {1, 2, 5, 4, null, ...  }

            // size = 3
            // {1, 2, null, 4, null, ...  }
            // employees[3] = 6
            // {1, 2, 5, 6, null, ...  }
            size++;
            employee.setId(++counterId);
            return true;
        } else {
            System.out.println("Репозиторий работников заполнен.");
            return false;
        }
    }

    public boolean removeEmployee(int id) { // employees[3] = employees {1,
        for (int i = 0; i < size; i++) {
            if (employees[i].getId() == id) {
//                // Если найден работник с заданным ID, удаляем его и сдви
                employees[i] = null;
                for (int j = i; j < size; j++) {
                    employees[j] = employees[j + 1];
                }
                // {1, 2 ,3, 4, null, ...  }
                // {1, 2 , null, 4, null, ...  }
                // {1, 2 , 4, null, ...    }
                // employees[2] = employees {1, 2 ,null, 4, null, ...  }
                size--;             // size = 3, -> size 2
                return true;
            }
```

```
            }
            System.out.println("Работник с ID " + id + " не найден.");
            return false;
        }

        public boolean removeEmployee2(int id) {
            for (int i = 0; i < size; i++) {
                if (employees[i] != null && employees[i].getId() == id) {
                    employees[i] = null;
                    System.out.println("Работник с ID " + id + " уволен");
                    return true;
                }
            }
            System.out.println("Работник с ID " + id + " не найден.");
            return false;
        }

        public BaseEmployee findEmployeeById(int id) {
            for (int i = 0; i < size; i++) {
                if (employees[i].getId() == id) {
                    return employees[i];
                }
            }
            return null;
        }

        public BaseEmployee[] getAllEmployees() {
            BaseEmployee[] result = new BaseEmployee[size];
            for (int i = 0; i < size; i++) {
                result[i] = employees[i];
            }
            return result;
        }

        public int countEmployees() {
            return size;
        }

        public BaseEmployee[] getAll() {
            return employees;
        }
    }
```

../lesson_30/code/Classwork_30/src/repo/EmployeeRepositoryInterface.java

```java
package repo;

import entity.BaseEmployee;
import entity.Employee;

/**
 * Интерфейс для репозитория работников.
 */
public interface EmployeeRepositoryInterface {

    /**
     * Добавляет работника в репозиторий.
     *
     * @param employee Добавляемый работник.
     */
    boolean addEmployee(BaseEmployee employee);

    /**
     * Удаляет работника из репозитория по его ID.
     *
     * @param id ID работника, которого необходимо удалить.
     */
    boolean removeEmployee(int id);

    /**
     * Ищет работника в репозитории по его ID.
     *
     * @param id ID работника, которого необходимо найти.
     * @return Найденный работник или null, если работник не найден.
     */
    BaseEmployee findEmployeeById(int id);

    /**
     * Получает массив всех работников в репозитории.
     *
     * @return Массив всех работников в репозитории.
     */
    BaseEmployee[] getAllEmployees();

    /**
     * Возвращает количество работников в репозитории.
     *
     * @return Количество работников в репозитории.
```

```
     */
    int countEmployees();
}
```

../lesson_30/code/Classwork_30/src/repo/EmployeeRepositoryTest.java

```
package repo;

import entity.BaseEmployee;
import entity.Developer;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

/**
 * @author Andrej Reutow
 * created on 18.10.2023
 */
class EmployeeRepositoryTest {

    private EmployeeRepository repository;

    @BeforeAll
    public static void init() {
        System.out.println("@BeforeAll");
    }

    @BeforeEach
    public void setUp() {
        System.out.println("@BeforeEach");
        repository = new EmployeeRepository();
    }

    @Test
    void test_countEmployees() {

        int result = repository.countEmployees();

        Assertions.assertEquals(0, result);
    }

    @Test
    void test_removeEmployee_() {
```

```
        //Дано
        BaseEmployee developer1 = new Developer("dev1", 100, 180, 2023);
        BaseEmployee developer2 = new Developer("dev2", 200, 150, 2023);
        BaseEmployee developer3 = new Developer("dev3", 300, 200, 2023);

        repository.addEmployee(developer1);
        repository.addEmployee(developer2);
        repository.addEmployee(developer3);

        BaseEmployee[] employees = repository.getAll();
        Assertions.assertEquals(developer2.getName(), employees[1].getName(

        // Когда
        boolean isRemoved = repository.removeEmployee(developer2.getId());

        // Тогда
        Assertions.assertTrue(isRemoved);
        Assertions.assertEquals(2, repository.countEmployees());

        Assertions.assertEquals(developer1.getName(), employees[0].getName(
        Assertions.assertEquals(developer3.getName(), employees[1].getName(
        for (int i = 2; i < employees.length; i++) {
            Assertions.assertNull(employees[i]);
        }
    }
}
```

../lesson_30/code/Classwork_30/src/Main.java

```
import entity.BaseEmployee;
import entity.Developer;
import entity.Manager;
import entity.Salesperson;
import repo.EmployeeRepository;

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

        BaseEmployee developer1 = new Developer("John", 25.0, 160, 2022);
        BaseEmployee developer2 = new Developer("Alice", 30.0, 150, 2020);
        BaseEmployee manager1 = new Manager("Bob", 3000.0, 5, 2019);
        BaseEmployee salesperson1 = new Salesperson("Eve", 2000.0, 10, 2021
```

```
        repository.addEmployee(developer1);
        repository.addEmployee(developer2);
        repository.removeEmployee(46987);
        repository.addEmployee(manager1);
        repository.addEmployee(salesperson1);


//        // Повысить зарплату для работников с опытом от 2 до 5 лет на 10%
//        BaseEmployee[] allEmployees = repository.getAllEmployees();
//        for (int i = 0; i < allEmployees.length; i++) {
//            Employee employee = allEmployees[i];
//            if (employee instanceof BaseEmployee) {
//                BaseEmployee baseEmployee = (BaseEmployee) employee;
//                baseEmployee.adjustSalaryByExperience(2, 5, 10);
//            }
//        }

        // Вывести информацию о работниках
        BaseEmployee[] allEmployees = repository.getAllEmployees();
        for (BaseEmployee employee : allEmployees) {
            System.out.println(employee);
        }
    }
}
```