МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
Кафедра ИИТ

ОТЧЁТ

по лабораторной работе №5
за 1 семестр 3 курса

Выполнил:
студент группы ПО-9(1)
3 курса
Зейденс Никита
Вячеславович

Проверил:
Крощенко
А. А.

Брест, 2024

Цель: Приобрести практические навыки в области объектно-ориентированного программирования.

Вариант 5

Задание 1: Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

interface Здание ← abstract class Общественное Здание ← class Театр.

Код программы:

```
// Интерфейс Здание
interface Building {
    void open();
    void close();
}
// Абстрактный класс Общественное Здание, реализующий интерфейс Здание
abstract class PublicBuilding implements Building {
    protected boolean opened;
    @Override
    public void open() {
        opened = true;
        System.out.println("Здание открыто");
    }
    @Override
    public void close() {
        opened = false;
        System.out.println("Здание закрыто");
    }
}

// Класс Театр, наследующийся от Общественного Здания
class Theatre extends PublicBuilding {
    private String name;
    public Theatre(String name) {
        this.name = name;
    }
    public void performance() {
        if (opened) {
            System.out.println("Идет представление в театре " + name);
        } else {
            System.out.println("Театр закрыт, представление не может быть
показано");
        }
    }
}
// Пример использования
public class Main {
    public static void main(String[] args) {
        Theatre theatre = new Theatre("Большой");
        theatre.open();
        theatre.performance();
        theatre.close();
        theatre.performance();
    }
}
```

Результат работы:

```
Здание открыто
Идет представление в театре Большой
Здание закрыто
Театр закрыт, представление не может быть показано
```

Задание 2: В следующих заданиях требуется создать суперкласс (абстрактный класс, интерфейс) и определить общие методы для данного класса. Создать подклассы, в которых добавить специфические свойства и методы. Часть методов переопределить. Создать массив объектов суперкласса и заполнить объектами подклассов. Объекты подклассов идентифицировать конструктором по имени или идентификационному номеру. Использовать объекты подклассов для моделирования реальных ситуаций и объектов.

Создать абстрактный класс Работник фирмы и подклассы Менеджер, Аналитик, Программист, Тестировщик, Дизайнер, Бухгалтер. Реализовать логику начисления зарплаты.

Код программы:

```
// Абстрактный класс Работник фирмы
abstract class Employee {
    private String name;
    private int employeeId;
    public Employee(String name, int employeeId) {
        this.name = name;
        this.employeeId = employeeId;
    }
    public String getName() {
        return name;
    }
    public int getEmployeeId() {
        return employeeId;
    }
    public abstract double calculateSalary();
}
// Подкласс Менеджер
class Manager extends Employee {
    private double baseSalary;
    private double bonus;
    public Manager(String name, int employeeId, double baseSalary, double
bonus) {
        super(name, employeeId);
        this.baseSalary = baseSalary;
        this.bonus = bonus;
    }
    @Override
    public double calculateSalary() {
        return baseSalary + bonus;
    }
}
// Подкласс Аналитик
class Analyst extends Employee {
    private double baseSalary;
    public Analyst(String name, int employeeId, double baseSalary) {
        super(name, employeeId);
        this.baseSalary = baseSalary;
    }
    @Override
    public double calculateSalary() {
        return baseSalary;
    }
}
// Подкласс Программист
class Programmer extends Employee {
```

```
    private double baseSalary;
    private int hoursWorked;
    private double hourlyRate;
    public Programmer(String name, int employeeId, double baseSalary, int
hoursWorked, double hourlyRate) {
        super(name, employeeId);
        this.baseSalary = baseSalary;
        this.hoursWorked = hoursWorked;
        this.hourlyRate = hourlyRate;
    }
    @Override
    public double calculateSalary() {
        return baseSalary + (hoursWorked * hourlyRate);
    }
}
// Подкласс Тестировщик
class Tester extends Employee {
    private double baseSalary;
    private int bugsFound;
    private double bugRate;
    public Tester(String name, int employeeId, double baseSalary, int
bugsFound, double bugRate) {
        super(name, employeeId);
        this.baseSalary = baseSalary;
        this.bugsFound = bugsFound;
        this.bugRate = bugRate;
    }
    @Override
    public double calculateSalary() {
        return baseSalary + (bugsFound * bugRate);
    }
}
// Подкласс Дизайнер
class Designer extends Employee {
    private double baseSalary;
    private int projectsCompleted;
    private double projectBonus;
    public Designer(String name, int employeeId, double baseSalary, int
projectsCompleted, double projectBonus) {
        super(name, employeeId);
        this.baseSalary = baseSalary;
        this.projectsCompleted = projectsCompleted;
        this.projectBonus = projectBonus;
    }
    @Override
    public double calculateSalary() {
        return baseSalary + (projectsCompleted * projectBonus);
    }
}
// Подкласс Бухгалтер
class Accountant extends Employee {
    private double baseSalary;
    private double taxRate;
    public Accountant(String name, int employeeId, double baseSalary, double
taxRate) {
        super(name, employeeId);
        this.baseSalary = baseSalary;
        this.taxRate = taxRate;
    }
    @Override
    public double calculateSalary() {
        return baseSalary - (baseSalary * taxRate);
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Employee[] employees = new Employee[6];
        employees[0] = new Manager("John", 1, 5000, 1000);
        employees[1] = new Analyst("Alice", 2, 4000);
        employees[2] = new Programmer("Bob", 3, 3000, 160, 20);
        employees[3] = new Tester("Eve", 4, 2500, 50, 10);
        employees[4] = new Designer("Mia", 5, 3500, 5, 100);
        employees[5] = new Accountant("Oliver", 6, 4500, 0.1);

        for (Employee employee : employees) {
            System.out.println("Employee: " + employee.getName());
            System.out.println("Employee ID: " + employee.getEmployeeId());
            System.out.println("Salary: $" + employee.calculateSalary());
            System.out.println("-----------------------------------");
        }
    }
}
```

Результат работы:

```
Employee: John
Employee ID: 1
Salary: $6000.0
-----------------------------------
Employee: Alice
Employee ID: 2
Salary: $4000.0
-----------------------------------
Employee: Bob
Employee ID: 3
Salary: $6200.0
-----------------------------------
Employee: Eve
Employee ID: 4
Salary: $3000.0
-----------------------------------
Employee: Mia
Employee ID: 5
Salary: $4000.0
-----------------------------------
Employee: Oliver
Employee ID: 6
Salary: $4050.0
-----------------------------------
```

Задание 3: В задании 3 ЛР №4, где возможно, заменить объявления суперклассов объявлениями абстрактных классов или интерфейсов.

Изменения:

Book

```
abstract class Book {
    private String name;
    private FullName author;
    private int year;

    Book() {
        this.name = "";
        this.author = new FullName();
        this.year = -1;
    }

    Book(String name, FullName author, int year) {
        this.name = name;
        this.author = author;
        this.year = year;
    }

    Book(Book book) {
        this.name = book.name;
        this.author = book.author;
        this.year = book.year;
    }

    public String getName() {
        return this.name;
    }

    public FullName getAuthor() {
        return this.author;
    }

    public int getYear() {
        return this.year;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAuthor(FullName author) {
        this.author = author;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public boolean equals(Book book) {
        boolean name = this.name.equals(book.name);
        boolean author = this.author.equals(book.author);
        boolean year = this.year == book.year;
        return name && author && year;
    }

    public abstract void print();
}
```

BorrowedBook

```
import java.util.function.Consumer;

class BorrowedBook extends Book {
    private int returnDay;
    private FullName reader;

    public BorrowedBook(Book book, int returnDay, FullName reader) {
        super(book);
        this.returnDay = returnDay;
        this.reader = reader;
    }

    public int getReturnDay() {
        return this.returnDay;
    }

    public FullName getReader() {
        return this.reader;
    }

    public void setReturnDay(int returnDay) {
        this.returnDay = returnDay;
    }

    public void setReader(FullName reader) {
        this.reader = reader;
    }

    @Override
    public void print() {
        System.out.print("Читатель: ");
        this.reader.print();
        System.out.print("Должен вернуть книгу в день: " + (this.returnDay +
1) + "\n");
        System.out.printf(getName() + ". " + getYear() + ". ");
        getAuthor().print();
    }

    public void printBook() {
        System.out.printf(getName() + ". " + getYear() + ". ");
        getAuthor().print();
    }
}
```

Main

124-128

```
new ConcreteBook(
        borrowedBooks.get(i).getName(),
        borrowedBooks.get(i).getAuthor(),
        borrowedBooks.get(i).getYear()
)
```

81-85

```
new ConcreteBook(
        BOOK_NAMES[random.nextInt(BOOK_NAMES.length)],
        AUTHOR_FULL_NAMES[random.nextInt(AUTHOR_FULL_NAMES.length)],
        BOOK_YEARS[random.nextInt(BOOK_YEARS.length)]
)
```

Добавлен:

ConcreteBook

```
public class ConcreteBook extends Book {
    public ConcreteBook() {
        super();
    }
    public ConcreteBook(String name, FullName author, int year) {
        super(name, author, year);
    }

    @Override
    public void print() {
        System.out.printf(getName() + ". " + getYear() + ". ");
        getAuthor().print();
    }
}
```

Результат работы прежний

Вывод: Практические навыки в области ООП были приобретены.