

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ

ОТЧЁТ
по лабораторной работе №5

Выполнил:

студент группы ПО-8
Вейгандт И.О.

Проверил:
Крощенко А.А.

Брест 2024

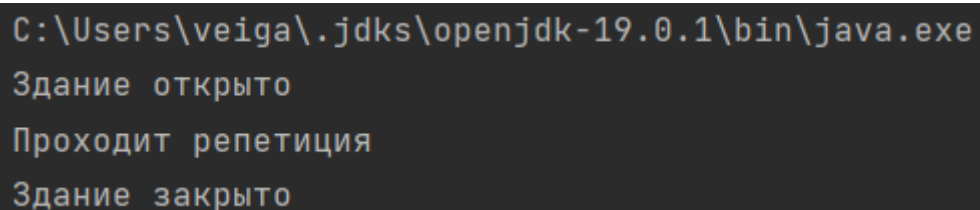
Вариант 5

Цель работы: приобрести практические навыки в области объектно-ориентированного проектирования.

Задание 1:

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов: interface Здание ← abstract class
Общественное Здание ← class Театр.

```
interface Building{
    void Open();
    void Close();
}
abstract class PublicBuilding implements Building{
    @Override
    public void Open() {
        System.out.println("Здание открыто");
    }
    @Override
    public void Close() {
        System.out.println("Здание закрыто");
    }
}
class Theater extends PublicBuilding{
    public void doEvent(){
        System.out.println("Проходит репетиция");
    }
}
public class Main {
    public static void main(String[] args) {
        Theater theater = new Theater();
        theater.Open();
        theater.doEvent();
        theater.Close();
    }
}
```



```
C:\Users\veiga\.jdk\openjdk-19.0.1\bin\java.exe
Здание открыто
Проходит репетиция
Здание закрыто
```

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.

```
import java.util.ArrayList;
import java.util.List;

class Employee{
    private String name;
    private int id;
    private double salary;
    private double finalSalary;
    private double bonus = 0;

    public Employee(String name, int id, double salary) {
        this.name = name;
        this.id = id;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

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

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double getBaseSalary() {
```

```

        return salary;
    }

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

    public void setBonus(double bonus) {
        this.bonus = bonus;
    }

    public double getBonus() {
        return bonus;
    }

    public double getSalary() {
        return salary;
    }

    public double getFinalSalary() {
        return finalSalary;
    }

    public void setFinalSalary(double finalSalary) {
        this.finalSalary = finalSalary;
    }

    public void doWork(){}
}
class Manager extends Employee {

    public Manager(String name, int employeeId, double
baseSalary) {
        super(name, employeeId, baseSalary);
    }

    @Override
    public void doWork() {
        System.out.println("Я " + this.getName() + "
менеджер, все под моим контролем");
    }

    public void awarding(Employee employee, double bonus) {
        employee.setBonus(employee.getBonus() + bonus);
        System.out.println("Менеджер " + this.getName() + "
награждает сотрудника " + employee.getName());
    }
}

```

```

}
class Booker extends Employee {

    public Booker(String name, int employeeId, double
baseSalary) {
        super(name, employeeId, baseSalary);
    }

    @Override
    public void doWork() {
        System.out.println("Я " + this.getName() + "
бухгалтер, у миня многа диняк");
    }

    public void calculateSalary(Employee employee) {
        employee.setFinalSalary(employee.getBonus() +
employee.getBaseSalary());
        employee.setBonus(0.0);
        System.out.println("Бухгалтер " + this.getName() +
" расчитал зп сотрудника " + employee.getName());
        System.out.println("Его зарплата = " +
employee.getFinalSalary());
    }
}
class Programmer extends Employee {

    public Programmer(String name, int employeeId, double
baseSalary) {
        super(name, employeeId, baseSalary);
    }
    @Override
    public void doWork() {
        System.out.println("Я " + this.getName() + "
программист, ща выключу интернет и никаких тиктоков");
    }

}
class Tester extends Employee {

    public Tester(String name, int employeeId, double
baseSalary) {
        super(name, employeeId, baseSalary);
    }
    @Override
    public void doWork() {
        System.out.println("Я " + this.getName() + "

```

```

тестировщик, я тестировщик....");
    }
}
class Designer extends Employee {

    public Designer(String name, int employeeId, double
baseSalary) {
        super(name, employeeId, baseSalary);
    }
    @Override
    public void doWork() {
        System.out.println("Я " + this.getName() + "
дизайнер, ща нарисую красоту");
    }

}

class Analyst extends Employee {

    public Analyst(String name, int employeeId, double
baseSalary) {
        super(name, employeeId, baseSalary);
    }
    @Override
    public void doWork() {
        System.out.println("Я " + this.getName() + "
аналитик, и что я тут делаю");
    }

}

public class Main {
    public static void main(String[] args) {

        Manager manager = new Manager("Илья", 1, 1000);
        Booker booker = new Booker("Егор", 2, 700);
        Programmer programmer = new Programmer("Семён", 3,
900);
        Tester tester = new Tester("Никита", 4, 700);
        Analyst analyst = new Analyst("Максим", 5, 800);
        Designer designer = new Designer("Даниил", 6, 800);

        List<Employee> employees = new ArrayList<>();

        employees.add(manager);
        employees.add(booker);

```

```

        employees.add(programmer);
        employees.add(tester);
        employees.add(analyst);
        employees.add(designer);

        for (Employee employee:employees){
            employee.doWork();
            booker.calculateSalary(employee);
        }
        manager.awarding(booker, 100);
        manager.awarding(designer, 200);

        booker.calculateSalary(booker);
        booker.calculateSalary(designer);

    }
}

```

```

C:\Users\veiga\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent
Я Илья менеджер, все под моим контролем
Бухгалтер Егор рассчитал зп сотрудника Илья
Его зарплата = 1000.0
Я Егор бухгалтер, у меня много диняк
Бухгалтер Егор рассчитал зп сотрудника Егор
Его зарплата = 700.0
Я Семён программист, ща выключу интернет и никаких тиктоков
Бухгалтер Егор рассчитал зп сотрудника Семён
Его зарплата = 900.0
Я Никита тестировщик, я тестировщик....
Бухгалтер Егор рассчитал зп сотрудника Никита
Его зарплата = 700.0
Я Максим аналитик, и что я тут делаю
Бухгалтер Егор рассчитал зп сотрудника Максим
Его зарплата = 800.0
Я Даниил дизайнер, ща нарисую красоту
Бухгалтер Егор рассчитал зп сотрудника Даниил
Его зарплата = 800.0
Менеджер Илья награждает сотрудника Егор
Менеджер Илья награждает сотрудника Даниил
Бухгалтер Егор рассчитал зп сотрудника Егор
Его зарплата = 800.0
Бухгалтер Егор рассчитал зп сотрудника Даниил
Его зарплата = 1000.0

```

Задание 3:

В задании 3 ЛР No4, где возможно, заменить объявления суперклассов объявлениями абстрактных классов или интерфейсов.

```
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class Main {

    public static void main(String[] args) {
        Library library = new Library();

        Book book1 = new Book( "Toyota. Путь к
совершенству", "Цунёси Нодзи");
        Book book2 = new Book("Цунёси Нодзи", "Дэвид
Килпатрик");
        Book book3 = new Book("Чистый код. Создание, анализ
и рефакторинг", "Роберт Мартин");
        Book book4 = new Book("Удивительный мир звука",
"Игорь Клюкин");

        Catalog catalog = new Catalog();

        catalog.addBook(book1);
        catalog.addBook(book2);
        catalog.addBook(book3);
        catalog.addBook(book4);

        Reader reader1 = new Reader("Илья");
        Reader reader2 = new Reader("Семён");
        Reader reader3 = new Reader("Егор");

        Order order1 =
reader1.placeOrderForReadingRoom(book1);
        Order order2 =
reader2.placeOrderForHome(book2, LocalDate.of(2024, 3, 1));
        Order order3 = reader3.placeOrderForHome(book3,
LocalDate.of(2024, 5, 5));
        Order order4 =
reader1.placeOrderForReadingRoom(book3);

        LibraryWorker libraryWorker = new
```



```

LibraryWorker(library);

    libraryWorker.addOrder(order1);
    libraryWorker.addOrder(order2);
    libraryWorker.addOrder(order3);
    libraryWorker.addOrder(order4);

    libraryWorker.issueBookToReader(order1);
    libraryWorker.issueBookToReader(order2);
    libraryWorker.issueBookToReader(order3);
    libraryWorker.issueBookToReader(order4);

    reader1.returnBook(book1,libraryWorker);
    reader2.returnBook(book2,libraryWorker);
    reader3.returnBook(book3,libraryWorker);

    Order order5 =
reader2.placeOrderForReadingRoom(book3);
    libraryWorker.issueBookToReader(order5);
    libraryWorker.issueBookToReader(order4);
}
}
class Book {
    private String title;
    private String author;
    private boolean available =true;
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public boolean isAvailable() {
        return available;
    }

    public void setAvailable(boolean available) {
        this.available = available;
    }
}

```

```

}
class Catalog {
    ArrayList<Book> books;

    public Catalog() {
        this.books = new ArrayList<>();
    }
    public void addBook(Book book){
        books.add(book);
    }
    public Book searchBook(String title){
        for (Book book: books){
            if(book.getTitle().equals(title)){
                return book;
            }
        }
        return null;
    }
}
class Order{
    private Reader reader;
    private Book book;
    boolean isForReadingRoom;

    private LocalDate returnDate = null;
    public Order(Reader reader, Book book, boolean
isForReadingRoom) {
        this.reader = reader;
        this.book = book;
        this.isForReadingRoom = isForReadingRoom;
    }
    public Order(Reader reader, Book book, boolean
isForReadingRoom, LocalDate date) {
        this.reader = reader;
        this.book = book;
        this.isForReadingRoom = isForReadingRoom;
        this.returnDate = date;
    }

    public Reader getReader() {
        return reader;
    }

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

```

```

    public Book getBook() {
        return book;
    }

    public void setBook(Book book) {
        this.book = book;
    }

    public boolean isForReadingRoom() {
        return isForReadingRoom;
    }

    public void setForReadingRoom(boolean forReadingRoom) {
        isForReadingRoom = forReadingRoom;
    }

    public LocalDate getReturnDate() {
        return returnDate;
    }

    public void setReturnDate(LocalDate returnDate) {
        this.returnDate = returnDate;
    }
}

class Library{
    Catalog catalog;
    protected List<Reader> blackList;
    public List<Order> orders;

    public Library(Catalog catalog) {
        this.catalog = catalog;
    }

    public Library() {
        this.blackList = new ArrayList<>();
        this.orders = new ArrayList<>();
    }

}

class Reader extends Library{
    List<Book> books;
    String name;
    boolean inBlackList = false;

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

```

```

        this.books = new ArrayList<>();
    }

    Order placeOrderForReadingRoom(Book book){
        if(!inBlackList && book.isAvailable()) {
            Order newOrder = new Order(this, book, true);
            return newOrder;
        }else if(inBlackList) {
            System.out.println("Читатель " + name + " в
черном списке и не может взять книгу " + book.getTitle());
            return null;
        }else {
            System.out.println("Книга " + book.getTitle() +
" находится в прокате " + name+ "не может ее взять");
            return null;
        }
    }

    Order placeOrderForHome(Book book, LocalDate
returnDate){
        if(!inBlackList && book.isAvailable()) {
            Order newOrder = new Order(this, book, false,
returnDate);
            return newOrder;
        }else if(inBlackList) {
            System.out.println("Читатель " + name + " в
черном списке и не может взять книгу " + book.getTitle());
            return null;
        }else {
            System.out.println("Книга " + book.getTitle() +
" находится в прокате " + name+ "не может ее взять");
            return null;
        }
    }

    public void returnBook(Book book,LibraryWorker
libraryWorker) {
        if (books.contains(book)) {
            books.remove(book);
            System.out.println("Книга " + book.getTitle() +
" возвращена читателем " + name);
            libraryWorker.bookReception(book);
        }
    }

    public List<Book> getBooks() {
        return books;
    }

```

```

    }

    public void setBooks(List<Book> books) {
        this.books = books;
    }

    public String getName() {
        return name;
    }

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

    public boolean isInBlackList() {
        return inBlackList;
    }

    public void setInBlackList(boolean inBlackList) {
        this.inBlackList = inBlackList;
    }
}

class LibraryWorker extends Library{
    Library library;

    public LibraryWorker(Library library) {
        this.library = library;
    }

    public void issueBookToReader(Order order) {
        if(order != null){
            Reader reader = order.getReader();
            Book book = order.getBook();
            if(!reader.inBlackList && book.isAvailable()) {
                reader.books.add(book);
                book.setAvailable(false);
                System.out.println("Книга " +
book.getTitle()+ " выдана читателю " + reader.name);
            }else if(reader.inBlackList) {
                System.out.println("Читатель " +
reader.name + " находится в черном списке и не может взять
книгу" + book.getTitle());
            }else {
                System.out.println("Книга " +
book.getTitle() + " находится в прокате " + reader.name + "
не может взять эту книгу");
            }
        }
    }
}

```

```

        }
    }
}

public void bookReception(Book book) {
    for(Order order : library.orders){
        if(order.getBook().equals(book)){
            book.setAvailable(true);
            if(!order.isForReadingRoom() &&
!returnedOnTime(order.getReturnDate())){
                Administrator administrator = new
Administrator();

administrator.addToBlackList(order.getReader());
            }
        }
    }

}

}

boolean returnedOnTime(LocalDate returnDate){
    LocalDate currentDate = LocalDate.now();
    if(currentDate.isBefore(returnDate) ||
currentDate.equals(returnDate)){
        return true;
    }
    return false;
}

void addOrder(Order order){
    library.orders.add(order);
}

class Administrator {
    public void addToBlackList(Reader reader) {
        reader.setInBlackList(true);
        blackList.add(reader);
        System.out.println(reader.name + " добавлен в
черный список");
    }
}

}

```

```
C:\Users\veiga\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Книга Toyota. Путь к совершенству выдана читателю Илья
Книга Цунёси Нодзи выдана читателю Семён
Книга Чистый код. Создание, анализ и рефакторинг выдана читателю Егор
Книга Чистый код. Создание, анализ и рефакторинг находится в прокате Илья не может взять эту книгу
Книга Toyota. Путь к совершенству возвращена читателем Илья
Книга Цунёси Нодзи возвращена читателем Семён
Семён добавлен в черный список
Книга Чистый код. Создание, анализ и рефакторинг возвращена читателем Егор
Егор добавлен в черный список
Читатель Семён в черном списке и не может взять книгу Чистый код. Создание, анализ и рефакторинг
Книга Чистый код. Создание, анализ и рефакторинг выдана читателю Илья
```

Вывод: приобрёл практические в области объектно-ориентированного проектирования.