|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа, обработки и интерпретации больших данных**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 4 |

**Название:** Внутренние классы, интерфейсы

**Дисциплина:** Языки программирования для работы с большими данными

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | Н.Б.Гендина | |
|  | (Группа) |  | (Подпись, дата) | | (И.О. Фамилия) |
|  |  |  |  | |  |
| Преподаватель |  |  |  | | П.В. Степанов |
|  |  |  | (Подпись, дата) | | (И.О. Фамилия) |

Москва, 2024

**Цель работы:** освоить базовые принципы работы с внутренними классами и интерфейсами на языке Java.

**Задание 1 вариант 4**

Создать класс Художественная Выставка с внутренним классом, с помощью объектов которого можно хранить информацию о картинах, авторах и времени проведения выставок.

|  |
| --- |
| import java.time.LocalDate;  import java.time.LocalTime;  public class Main {  public static void main(String[] args) {  Exhibition exhibition1 = new Exhibition("Современное искусство", "Эрарта");  Exhibition exhibition2 = new Exhibition("Классическое искусство", "Эрмитаж");  exhibition1.addStructure(  new String[]{"Черный квадрат", "Пейзаж", "Красный квадрат"},  new String[]{"Малевич", "Кандинский"},  LocalDate.parse("2024-03-03")  );  exhibition1.addStructure(  new String[]{"Черный квадрат", "Красный квадрат"},  new String[]{"Малевич"},  LocalDate.parse("2024-03-10")  );  exhibition1.addStructure(  new String[]{"Черный квадрат", "Пейзаж", "Подсолнухи"},  new String[]{"Малевич", "Кандинский", "Ван Гог"},  LocalDate.parse("2024-03-17")  );  exhibition2.addStructure(  new String[]{"Три грации", "Союз земли и воды"},  new String[]{"Рубенс"},  LocalDate.parse("2024-03-11")  );  exhibition2.addStructure(  new String[]{"Даная", "Три грации", "Возвращение блудного сына"},  new String[]{"Рубенс", "Рембрант",},  LocalDate.parse("2024-03-13")  );  System.out.println(exhibition1);  System.out.println(exhibition2);  System.out.println();  System.out.println(exhibition1.getInfoAboutAllDates());  System.out.println(exhibition2.getInfoAboutAllStructures());  System.out.println(exhibition1.getInfoAboutStructure(LocalDate.parse("2024-03-10")));  }  } |

|  |
| --- |
| import java.time.LocalDate;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.List;  public class Exhibition {  private final String name;  private final String place;  private List<Structure> structures;  public Exhibition(String name, String place) {  this.name = name;  this.place = place;  this.structures = new ArrayList<>();  }  public String toString() {  return String.format(  "Выставка %s проходит в %s, количество событий %d",  this.name, this.place, structures.size()  );  }  public String getInfoAboutAllStructures() {  StringBuilder result = new StringBuilder();  for (Structure structure: this.structures) {  result.append(structure).append("\n");  }  return result.toString();  }  public String getInfoAboutAllDates() {  StringBuilder result = new StringBuilder("Выставка проводится в даты: ");  for (Structure structure: this.structures) {  result.append(structure.date.toString()).append(", ");  }  return result.toString();  }  public String getInfoAboutStructure(LocalDate date) {  return Arrays.toString(this.structures.stream().filter((s) -> s.date.equals(date)).toArray());  }  public void addStructure(String[] paintings, String[] authors, LocalDate date) {  this.structures.add(new Structure(paintings, authors, date));  }  class Structure {  private final String[] paintings;  private final String[] authors;  private final LocalDate date;  Structure(String[] paintings, String[] authors, LocalDate date) {  this.paintings = paintings;  this.authors = authors;  this.date = date;  }  public String toString() {  return String.format(  "Выставка %s числа, представлены картины: %s от авторов: %s",  this.date.toString(), Arrays.toString(this.paintings), Arrays.toString(this.authors)  );  }  }  } |

**Задание 1 вариант 5**

Создать класс Календарь с внутренним классом, с помощью объектов которого можно хранить информацию о выходных и праздничных днях.

|  |
| --- |
| public class Main {  public static void main(String[] args) {  Calendar calendar = new Calendar("март", 2024);  calendar.addDayOffList("8 марта", 2, 8, true);  calendar.addDayOffList("суббота", 6, 3, false);  calendar.addDayOffList("суббота", 6, 10, false);  calendar.addDayOffList("воскресенье", 7, 4, false);  calendar.addDayOffList("воскресенье", 7, 11, false);  System.out.println(calendar);  for(Calendar.DayOff dayOff: calendar.getDayOffList()) {  System.out.println(dayOff);  }  }  } |

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  public class Calendar {  private final String month;  private final int year;  private List<DayOff> dayOffList;  public Calendar(String month, int year) {  this.month = month;  this.year = year;  this.dayOffList = new ArrayList<>();  }  public String toString() {  return String.format(  "Календарь за %s %d год, количество праздников и выходных: %d",  this.month, this.year, this.dayOffList.size()  );  }  public List<DayOff> getDayOffList() {  return this.dayOffList;  }  public void addDayOffList(String name, int dayOfWeekNumber, int day, boolean isHoliday) {  this.dayOffList.add(new DayOff(name, dayOfWeekNumber, day, isHoliday));  }  enum WeekDay {  MONDAY,  TUESDAY,  WEDNESDAY,  THURSDAY,  FRIDAY,  SATURDAY,  SUNDAY,  }  class DayOff {  private final String name;  private final int dayOfWeekNumber;  private final int day;  private final boolean isHoliday;  DayOff(String name, int dayOfWeekNumber, int day, boolean isHoliday) {  this.name = name;  this.dayOfWeekNumber = dayOfWeekNumber;  this.day = day;  this.isHoliday = isHoliday;  }  public String toString() {  if (this.isHoliday) {  return String.format(  "Праздник %s %d числа (%s)",  this.name, this.day, WeekDay.values()[this.dayOfWeekNumber - 1]  );  }  return String.format(  "Обычный выходной %d числа (%s)",  this.day, WeekDay.values()[this.dayOfWeekNumber - 1]  );  }  }  } |

**Задание 2 вариант 4**

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

|  |
| --- |
| import java.time.LocalTime;  public class Main {  public static void main(String[] args) {  Theater theater1 = new Theater(  "Большой театр",  LocalTime.parse("09:00"),  LocalTime.parse("21:00"),  1000  );  Theater theater2 = new Theater(  "Малый театр",  LocalTime.parse("08:00"),  LocalTime.parse("20:00"),  200  );  System.out.println(theater1);  System.out.println(theater2);  theater1.displayOperatingTime();  theater2.displayOperatingTime();  theater1.openAllDoors();  theater2.openAllDoors();  theater1.buyTicket();  theater1.buyTicket();  theater1.buyTicket();  theater2.buyTicket();  theater2.buyTicket();  theater2.buyTicket();  theater2.buyTicket();  System.out.println("В " + theater1 + " осталось билетов: " + theater1.getNumberOfTickets());  System.out.println("В " + theater2 + " осталось билетов: " + theater2.getNumberOfTickets());  theater1.closeAllDoors();  theater2.closeAllDoors();  }  } |

|  |
| --- |
| import java.time.LocalTime;  interface IBuilding {  public abstract void closeAllDoors();  public abstract void openAllDoors();  }  abstract class PublicBuilding implements IBuilding {  protected final String name;  protected final LocalTime startTime;  protected final LocalTime endTime;  protected final int maximumCapacity;  protected boolean allDoorsOpened;  PublicBuilding(String name, LocalTime startTime, LocalTime endTime, int maximumCapacity) {  this.name = name;  this.startTime = startTime;  this.endTime = endTime;  this.maximumCapacity = maximumCapacity;  }  public abstract void displayOperatingTime();  public abstract String toString();  public int getMaximumCapacity() {  return this.maximumCapacity;  }  public LocalTime getStartTime() {  return this.startTime;  }  public LocalTime getEndTime() {  return this.endTime;  }  public boolean getAllDoorsOpened() {  return this.allDoorsOpened;  }  public String getName() {  return this.name;  }  public void closeAllDoors() {  this.allDoorsOpened = false;  System.out.println("Все двери закрыты");  }  public void openAllDoors() {  this.allDoorsOpened = true;  System.out.println("Все двери открыты");  }  }  public class Theater extends PublicBuilding {  private int numberOfTickets;  Theater(String name, LocalTime startTime, LocalTime endTime, int maximumCapacity) {  super(name, startTime, endTime, maximumCapacity);  numberOfTickets = 100;  }  public int getNumberOfTickets() {  return this.numberOfTickets;  }  public void buyTicket() {  numberOfTickets -= 1;  System.out.printf("Билет куплен, осталось %d\n", this.numberOfTickets);  }  @Override  public void displayOperatingTime() {  System.out.printf("Время работы театра: c %s по %s\n", this.startTime.toString(), this.endTime.toString());  }  @Override  public String toString() {  return String.format("Театр %s", this.name);  }  } |

**Задание 2 вариант 5**

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов: interface Mobile <- abstract class Siemens Mobile <- class Model.

|  |
| --- |
| public class Main {  public static void main(String[] args) {  Model model1 = new Model("Ультра", 2000, 8);  Model model2 = new Model("Супер", 2003, 12);  Model model3 = new Model("Про", 2004, 16);  System.out.println(model1);  System.out.println(model2);  System.out.println(model3);  model1.call("79999999999");  model2.call("78888888888");  model3.call("72222222222");  model1.sendMessage();  model2.sendMessage();  model3.sendMessage();  model1.takePhoto();  model2.takePhoto();  model3.takePhoto();  }} |
| interface IMobile {  public abstract void call(String number);  public abstract void takePhoto();  public abstract void sendMessage();  }  abstract class SiemensMobile implements IMobile {  protected final String name;  protected final int year;  protected final int ram;  SiemensMobile(String name, int year, int ram) {  this.name = name;  this.year = year;  this.ram = ram;  }  public String toString() {  return String.format(  "Мобильный телефон Siemens, модель %s, год выпуска %d, (RAM %d)",  this.name, this.year, this.ram  );  }  }  public class Model extends SiemensMobile {  Model(String name, int year, int ram) {  super(name, year, ram);  }  @Override  public void call(String number) {  System.out.printf("Bell.. Bell.. Звоню на номер %s с телефона %s\n", number, this.name);  }  @Override  public void takePhoto() {  System.out.printf("Делаю красивое фото с телефона %s\n", this.name);  }  @Override  public void sendMessage() {  System.out.printf("Отправляю сообщение с телефона %s\n",this.name);  }  } |

**Вывод:** в ходе лабораторной работы были освоены базовые принципы работы с внутренними классами и интерфейсами на языке Java.