1. **Шаг 1:**

|  |
| --- |
| Создайте интрефейс с именем Item, который обяжет все классы реализовывать следующий  функции.  • Константу с именем manufacturer, которой будет задаваться значение "OracleProduction"  • Метод setProductionNumber одним целочисленным параметром  • Метод setName с одним строковым параметром  • Метод getName, возвращающий строку  • Метод getManufactureDate, возвращающий объект Date  • Метод getSerialNumber, возвращающий целое число |

**Результат**: Item.java

|  |
| --- |
| import java.util.Date;  /\*\*   \* Шаг 1  Создайте интрефейс с именем Item, который обяжет все классы реализовывать следующий  функции.  • Константу с именем manufacturer, которой будет задаваться значение "OracleProduction"  • Метод setProductionNumber одним целочисленным параметром  • Метод setName с одним строковым параметром  • Метод getName, возвращающий строку  • Метод getManufactureDate, возвращающий объект Date  • Метод getSerialNumber, возвращающий целое число   \*   \* @author (Зыкин Герман Леонидович)   \*/  public interface Item {          String MANUFACTURER = "OracleProduction"; // Константа          void setProductionNumber(int productionNumber); // Метод для установки номера          void setName(String name); // Метод для установки названия          String getName(); // Метод для получения названия          Date getManufactureDate(); // Метод для получения даты производства          int getSerialNumber(); // Метод для получения серийного номера  } |

1. **Шаг 2:**

|  |
| --- |
| Все элементы имеют заранее заданый тип. В настоящий момент есть 4 типа. Создайте перечисление enum с именем ItemType, которое будет хранить следующие значения. |

**Результат**: ItemType.Java

|  |
| --- |
| /\*\*   \* Step 2  Все элементы имеют заранее заданый тип. В настоящий момент есть 4 типа. Создайте  перечисление enum с именем ItemType  \*  \*@author (Зыкин Герман Леонидович)   \*/  public enum ItemType {      Audio("AU"),      Visual("VI"),      AudioMobile("AM"),      VisualMobile("VM");        private final String code;        ItemType(String code) {          this.code = code;      }        public String getCode() {          return code;      }  } |

1. **Шаг 3 / Шаг 14/Шаг 17:**

|  |
| --- |
| **Шаг 3.** Создайте класс Product, который будет реализовывать интерфейс Item. В Product  будет реализован базовый функционал, которым обладают все элементы производственной  линии. Добавьте в Product следующие поля:  • int serialNumber  • String manufacturer  • Date manufacuredOn  • String name  Добавьте целочисленную переменную класса с именем currentProductionNumber. В ней будет  хранится следующее число, которое будет присваиваться полю serialNumber.  Реализуйте методы из интерфейса Item.  Добавьте конструктор, который будет принимать в качестве аргумента имя продукта и  присвойте это имя полю name. Вы также заполните значение серийного номера согласно  значению из перменной currentProductionNumber. Значение переменной  currentProductionNumber должно увеличиваться и быть готовым к следующему экземпляру.  Установите значение поле manufacturedOn текущими значениями времени и даты.  Добавьте метод toString, который будет возвращать следующее:    **Шаг 14:**  Добавьте функционал к своим классам, который позволит сортировать их экземпляры по имени с использованием метода Collections.sort.  **Шаг 17:**  Создайте статический метод с именем printType в классе Product, который будет проходить по  коллекции и вывдоить все объекты определенного типа.  Например, вывести только объекты класса AudioPlayer из коллекции.  В качестве дополнительного бонуса вы можете изменить этот метод, чтобы он принимал класс,  объекты которого нужно вывести в качестве параметрa. Таким образом мы сможем использовать  этот метод для классов, которые мы пока не создали.  Ограничьте коллекцию только объектами, которые являются экземплярами подклассов Product |

**Результат:** Product.java

|  |
| --- |
| import java.util.Collection;  import java.util.Date;  /\*\*   \* Шаг 3 - Создайте класс Product, который будет реализовывать интерфейс Item. В Product  будет реализован базовый функционал, которым обладают все элементы производственной  линии. Добавьте в Product следующие поля:  • int serialNumber  • String manufacturer  • Date manufacuredOn  • String name  Добавьте целочисленную переменную класса с именем currentProductionNumber. В ней будет  хранится следующее число, которое будет присваиваться полю serialNumber.  Реализуйте методы из интерфейса Item.  Добавьте конструктор, который будет принимать в качестве аргумента имя продукта и  присвойте это имя полю name. Вы также заполните значение серийного номера согласно  значению из перменной currentProductionNumber. Значение переменной  currentProductionNumber должно увеличиваться и быть готовым к следующему экземпляру.  Установите значение поле manufacturedOn текущими значениями времени и даты.  Добавьте метод toString,   \*/  public class Product implements Item,Comparable<Product> {      private static int currentProductionNumber = 0; // Хранит текущее значение номера производства      private int serialNumber; //серийник      private Date manufacturedOn; //дата производства      private String name; //нпименование      public Product(String name) {          this.name = name;          this.serialNumber = currentProductionNumber++; // Присваиваем новый серийный номер          this.manufacturedOn = new Date(); // Устанавливаем текущую дату и время      }      @Override //сеттер номера      public void setProductionNumber(int productionNumber) {          this.serialNumber = productionNumber;      }      @Override      public void setName(String name) { //сеттер имени          this.name = name;      }      @Override //геттер имени      public String getName() {          return this.name;      }      @Override //геттер даты производства      public Date getManufactureDate() {          return this.manufacturedOn;      }      @Override //геттер серийника      public int getSerialNumber() {          return this.serialNumber;      }        @Override //вывод      public String toString() {          return "Manufacturer : " + MANUFACTURER + "\n" +          "Serial Number : " + serialNumber + "\n" +          "Date : " + manufacturedOn.toString() + "\n" +          "Name : " + name;      }        /\*\*       \* Шаг 14       \* Добавьте функционал к своим классам, который позволит сортировать их экземпляры по имени с  использованием метода Collections.sort.       \*/        @Override      public int compareTo(Product other) {          return this.name.compareTo(other.name); // Сравнение по имени      }      /\*\*       \* Шаг 17:  Создайте статический метод с именем printType в классе Product, который будет проходить по  коллекции и вывдоить все объекты определенного типа.  Например, вывести только объекты класса AudioPlayer из коллекции.  В качестве дополнительного бонуса вы можете изменить этот метод, чтобы он принимал класс,  объекты которого нужно вывести в качестве параметрa. Таким образом мы сможем использовать  этот метод для классов, которые мы пока не создали.  Ограничьте коллекцию только объектами, которые являются экземплярами подклассов Product       \*/      public static <T extends Product> void printType(Collection<T> products, Class<T> type) {          for (T product : products) {              if (type.isInstance(product)) {                  System.out.println(product);              }          }      }  } |

1. **Шаг 4:**

|  |
| --- |
| Все элементы этой производственной линии будут иметь базовые элементы управления  медиа. Созaдайте интрефейс MultimediaControl, в котором объявлены следующие методы.  • public void play();  • public void stop();  • public void previous();  • public void next(); |

**Результат:** MultimediaControl.java

|  |
| --- |
| /\*\*   \* Шаг 4  Все элементы этой производственной линии будут иметь базовые элементы управления  медиа. Созaдайте интрефейс MultimediaControl, в котором объявлены следующие методы.  • public void play();  • public void stop();  • public void previous();  • public void next();   \*   @author (Зыкин Герман Леонидович)   \*/  public interface MultimediaControl {      void play();      void stop();      void previous();      void next();  } |

1. **Шаг 5:**

|  |
| --- |
| Нам необходим конкретный класс, который позволит получить данные об аудиоплеере.  Создайте класс AudioPlayer, который является подклассом Product и реализует интерфейс  MultimediaControl.  В классе будут 2 поля  • String audioSpecification  • ItemType mediaType  Создайте конструктор с двумя параметрами – name и audioSpecification.  Конструктор также должен вызывать конструктор родителя и задавать media type  Реализуйте методы из интерфейса MultimediaControl, выводя в консоль сообщение о  соответствующем действии. Например, для play выведите System.out.println("Playing"); Обычно  нам приходится программировать инструкции для медиаплеера, но в этом задании просто будем  выводить сообщение.  Создайте метод toString, который будет вызывать метод toString родительского класса и добавит  вывод полей Audio Spec и Type. |

**Результат**: AudioPlayer.java

|  |
| --- |
| /\*\*   \* Шаг - 5  Нам необходим конкретный класс, который позволит получить данные об аудиоплеере.  Создайте класс AudioPlayer, который является подклассом Product и реализует интерфейс  MultimediaControl.  В классе будут 2 поля  • String audioSpecification  • ItemType mediaType  Создайте конструктор с двумя параметрами – name и audioSpecification.  Конструктор также должен вызывать конструктор родителя и задавать media type  Реализуйте методы из интерфейса MultimediaControl, выводя в консоль сообщение о  соответствующем действии. Например, для play выведите System.out.println("Playing"); Обычно  нам приходится программировать инструкции для медиаплеера, но в этом задании просто будем  выводить сообщение.  Создайте метод toString, который будет вызывать метод toString родительского класса и добавит  вывод полей Audio Spec и Type.   \*   @author (Зыкин Герман Леонидович)   \*/  public class AudioPlayer extends Product implements MultimediaControl  {      private String audioSpecification; /      private ItemType mediaType; //тип из ItemType      public AudioPlayer(String name, String audioSpecification) {          super(name); // Вызов конструктора родителя          this.audioSpecification = audioSpecification;          this.mediaType = ItemType.Audio; // Предположим, что тип медиа – AUDIO по умолчанию      }      @Override      public void play() {          System.out.println("Играет");      }      @Override      public void stop() {          System.out.println("Остановлено");      }      @Override      public void previous() {          System.out.println("Предыдущий трек");      }      @Override      public void next() {          System.out.println("Следующий трек");      }      @Override      public String toString() {          return super.toString() + "\nСпецификация : " + audioSpecification + "\nТип : " + mediaType;      }  } |

1. **Шаг 6:**

|  |
| --- |
| Создайте управляющий класс для AudioPlayer, который будет проверять, можем ли мы создавать  экземпляры AudioPlayer, использовать элементы управления и выводить информацию в консоль |

**Результат**: Test\_AudioPlayer.java

|  |
| --- |
| /\*\*   \* Шаг 6   \* Создайте управляющий класс для AudioPlayer, который будет проверять, можем ли мы создавать  экземпляры AudioPlayer, использовать элементы управления и выводить информацию в консоль  @author (Зыкин Герман Леонидович)   \*/  public class Test\_AudioPlayer {          public void test() {          // Создание экземпляра AudioPlayer         AudioPlayer player = new AudioPlayer("iPod", "wav");          // Использование методов управления медиа         player.play();         player.next();         player.previous();         player.stop();          // Вывод информации о плеере         System.out.println(player.toString());          }  } |

1. **Шаг 7:**

|  |
| --- |
| На производстве также будут создаваться портативные видеоплееры. Основным отличием их от  аудиоплеер является наличие экранов.  Создайте перечисление с именем MonitorType, в котором будут храниться значения |

**Результат:** MonitorType.java

|  |
| --- |
| /\*\*   \* На производстве также будут создаваться портативные видеоплееры. Основным отличием их от  аудиоплеер является наличие экранов.  Создайте перечисление с именем MonitorType, в котором будут храниться значения  @author (Зыкин Герман Леонидович)   \*/  public enum MonitorType {      LCD,      LED;  } |

1. **Шаг 8:**

|  |
| --- |
| Создайте интерфейс с именем ScreenSpec. В нем будут объявлены 3 метода:  • public String getResolution();  • public int getRefreshRate();  • public int getResponseTime(); |

**Результат:** ScreenSpec.java

|  |
| --- |
| /\*\*   \* Создайте интерфейс с именем ScreenSpec. В нем будут объявлены 3 метода:  • public String getResolution();  • public int getRefreshRate();  • public int getResponseTime();  @author (Зыкин Герман Леонидович)   \*/  public interface ScreenSpec {      String getResolution();      int getRefreshRate();      int getResponseTime();  } |

1. **Шаг 9:**

|  |
| --- |
| Создайте класс с именем Screen, который реализует интрефейс ScreenSpec. Добавьте 3 поля:  String resolution  int refreshrate  int responsetime  Закончите методы из интерфейса ScreenSpec.  Добавьте метод toString, который будет выводить данные трех полей в том же формате, как и в классе Product. |

**Результат:** Screen.Java

|  |
| --- |
| /\*\*   \* Шаг 9  Создайте класс с именем Screen, который реализует интрефейс ScreenSpec. Добавьте 3 поля:  String resolution  int refreshrate  int responsetime  Закончите методы из интерфейса ScreenSpec.  Добавьте метод toString, который будет выводить данные трех полей в том же формате, как и в  классе Product.  @author (Зыкин Герман Леонидович)   \*/  public class Screen implements ScreenSpec {      private String resolution;      private int refreshRate;      private int responseTime;      public Screen(String resolution, int refreshRate, int responseTime) {          this.resolution = resolution;          this.refreshRate = refreshRate;          this.responseTime = responseTime;      }      @Override      public String getResolution() {          return resolution;      }      @Override      public int getRefreshRate() {          return refreshRate;      }      @Override      public int getResponseTime() {          return responseTime;      }      @Override      public String toString() {          return "Разрешение: " + resolution + "\nЧастота обновления: " + refreshRate + " Гц.\nВремя отклика: " + responseTime + " мсек";      }  } |

1. **Шаг 10:**

|  |
| --- |
| Создайте управляющий класс для Screen, который будет проверять функционал класса Screen. |

**Результат:** Test\_Screen.java

|  |
| --- |
| /\*\*   \* Шаг 10   \* Создайте управляющий класс для Screen, который будет проверять функционал класса Screen.   \*   \* @author (Зыкин Герман Леонидович)   \*/  public class Test\_Screen {      public void test() {            Screen testScreen = new Screen("1920x1080", 144, 5);          System.out.println(testScreen.toString());      }  } |

1. **Шаг 11:**

|  |
| --- |
| Создайте класс с именем MoviePlayer который наследует класс Product и реализует  MultimediaControl.  Добавьте в этот класс два поля с именами screen и monitor type и присвойте им соответствующие  значения.  Реализуйте методы из интерфейса MultimediaControl по аналогии с аудиоплеером.  Создайте метод toString, который вызывает метод toString класса product и вывдоит детали о мониторе и экране. |

**Результат**: MoviePlayer.java

|  |
| --- |
| import java.util.Date;  /\*\*   \* Создайте класс с именем MoviePlayer который наследует класс Product и реализует  MultimediaControl.  Добавьте в этот класс два поля с именами screen и monitor type и присвойте им соответствующие  значения.  Реализуйте методы из интерфейса MultimediaControl по аналогии с аудиоплеером.  Создайте метод toString, который вызывает метод toString класса product и вывдоит детали о мониторе и экране.  @author (Зыкин Герман Леонидович)   \*/  public class MoviePlayer extends Product implements MultimediaControl {      private Screen screen;      private MonitorType monitorType;        public MoviePlayer(String name, Screen screen, MonitorType monitorType) {          super(name); // Вызов конструктора родителя          this.screen = screen;          this.monitorType = monitorType;        }      @Override      public void play() {          System.out.println("Playing");      }      @Override      public void stop() {          System.out.println("Stopped");      }      @Override      public void previous() {          System.out.println("Playing previous track");      }      @Override      public void next() {          System.out.println("Playing next track");      }      @Override      public String toString() {          return super.toString() +                  "\nScreen Details: " + screen.toString() +                  "\nMonitor Type: " + monitorType;      }  } |

1. **Шаг 12:**

|  |
| --- |
| Создайте управляющий класс, чтобы проверить функционал видеоплеера |

**Результат:** Test\_MoviePlayer.java

|  |
| --- |
| /\*\*   \* Шаг 12  Создайте управляющий класс, чтобы проверить функционал видеоплеера  @author (Зыкин Герман Леонидович)   \*/  public class Test\_MoviePlayer {      public void test() {          Screen testScreen = new Screen("1920x1080", 144, 5);          MoviePlayer moviePlayer = new MoviePlayer("TestMoviePlayer", testScreen, MonitorType.LED);           // Использование методов управления медиа          moviePlayer.play();          moviePlayer.next();          moviePlayer.previous();          moviePlayer.stop();          // Вывод информации о плеере          System.out.println(moviePlayer.toString());      }  } |

1. **Шаг 13:**

|  |
| --- |
| Аудиоплеер и видеоплеер имеют один и тот же управляющий интерфейс для физических  устройств. Управляющему интерфейсу неважен тип устройства, видео или аудио. Создайте  управляющий класс, который продемонстрирует, что для любого класса, реализующего интрефейс  MultimediaControl можно создать экземпляр и использовать его методы, независимо от того  является этот экземпляр аудио или видеоплеером. |

**Результат**: Test\_AnyMultimedia.java

|  |
| --- |
| import java.util.ArrayList;  /\*\*   \* Шаг 13   \* Аудиоплеер и видеоплеер имеют один и тот же управляющий интерфейс для физических  устройств. Управляющему интерфейсу неважен тип устройства, видео или аудио. Создайте  управляющий класс, который продемонстрирует, что для любого класса, реализующего интрефейс  MultimediaControl можно создать экземпляр и использовать его методы, независимо от того  является этот экземпляр аудио или видеоплеером.  @author (Зыкин Герман Леонидович)   \*/  public class Test\_AnyMultimedia {      public void test() {            ArrayList<MultimediaControl> devices = new ArrayList<MultimediaControl>();            AudioPlayer player = new AudioPlayer("iPod", "wav");            Screen testScreen = new Screen("1920x1080", 144, 5);          MoviePlayer moviePlayer = new MoviePlayer("TestMoviePlayer", testScreen, MonitorType.LED);              devices.add(player);          devices.add(moviePlayer);          for(MultimediaControl device : devices) {              controlDevice(device);          }      }      public void controlDevice(MultimediaControl device) {          device.play();          device.next();          device.previous();          device.stop();      }  } |

1. **Шаг 15 / Шаг 16:**

|  |
| --- |
| **Шаг 15:** Вы будете хранить коллекцию устройств, добавляя их по мере их производства. Выберите  подходящую коллекцию Java и создайте управляющий класс, который демонстрирует ее  использование. Вы также должны продемонстрировать работу метода Collections.sort.  **Шаг 16:**  Создайте метод с именем print, который будет печатать ее содержимое. Он должен работать для всех ваших классов |

**Результат:** Test\_Sort.java

|  |
| --- |
| /\*\*   \* Шаг 15   \* Вы будете хранить коллекцию устройств, добавляя их по мере их производства. Выберите  подходящую коллекцию Java и создайте управляющий класс, который демонстрирует ее  использование. Вы также должны продемонстрировать работу метода Collections.sort.  @author (Зыкин Герман Леонидович)   \*/  import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public class Test\_Sort {      public void test() {          ArrayList<Product> products = new ArrayList<Product>();          products = testCollection();          Collections.sort(products);          print(products);      }      public static ArrayList<Product> testCollection() {          ArrayList<Product> products = new ArrayList<Product>();          AudioPlayer ap1 = new AudioPlayer("iPod", "WAV");          AudioPlayer ap2 = new AudioPlayer("ChinaNoname player", "Mp3 ");          MoviePlayer mp1 = new MoviePlayer("Sony",new Screen("640x480", 40, 22), MonitorType.LCD);          MoviePlayer mp2 = new MoviePlayer("Panasonic",new Screen("1920x1080", 40, 22), MonitorType.LED);          products.add(ap1);          products.add(ap2);          products.add(mp1);          products.add(mp2);          return products;      }        /\*\*       \* Шаг 16       \* Создайте метод с именем print, который будет печатать ее содержимое. Он должен работать для          всех ваших классов       \*/      public static void print(ArrayList<Product> products) {          for (Product p : products) {              System.out.println(p);          }      }  } |

1. **Шаг 18 / Шаг 19 / Шаг 20:**

|  |
| --- |
| **Шаг 18:**  Для тестрирования качества производства необходима программа испытаний, которая должна  хранить данные о сотруднике, который проводил испытание. Для реализации этого вам  потребуется создать класс с именем EmployeeInfo, который позволит пользователю ввести имя, а затем создать id для пользователя, состоящий из первой буквы имени и фамилии.  В этом классе будет 2 поля  • StringBuilder name;  • String code;  В классе будут определены следующие методы:  • public StringBuilder getName()  • public String getCode()  • private void setName()  • private void createEmployeeCode(StringBuilder name)  • private String inputName()  • private boolean checkName(StringBuilder name)  Метод setName() будет вызываться из конструктора, который будет использовать inputName() для  получения имени (имени и фамилией) в виде единого ввода пользователя, перед тем как  вызывается метод checkName() для проверки наличия пробела во введенном имени.  Если введенное имя корректно, то вызывается метод createEmployeeCode(), который берет первую  букву имени и добавляет к фамилии для создания кода (id). Если пробела нет, то в качестве кода  будет использоваться значение guest.  Создайте в классе TestProductionLine объект employee, используя класс EmployeeInfo. Используйте  метод getCode() чтобы вывести код сотрудника в конце вывода данных о продукте.  **Шаг 19:**  Для провдения испытаний требуется дополнительная информация о подразделении сотрудника.  Код подразделения состоит из четырех букв и двух цифр. Формат кода подразделения: первая  буква заглавная, три последующих - строчные, и среди них не должно быть пробелов.  В класс EmployeeInfo нужно добавить три поля:  • String deptId;  • Pattern p;  • Scanner in;  Необходимо определить следующие методы:  • public String getDeptId()  • private void setDeptId()  • private String getId()  • private boolean validId(String id)  **Шаг 20:**  Чтобы предотвратить утечку информации, необходимо шифровать данные, сохраняемые в файл.  Чтобы выполнить это требование добавьте новый метод в класс EmployeeInfo, который будет  "переворачивать" текст, хранящийся в department id. Это нужно сделать рекурсивно с  использованием метода reverseString().  Нужно объявить следующий метод:  • public String reverseString(String id)  Если предоставлен корректный код подразделения (department id), то, перед тем как полю  deptId присваивается значение, вызывается reverseString(). |

**Результат:** EmployeeInfo.java

|  |
| --- |
| import java.util.Scanner;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*   \* Шаг 18:  Для тестрирования качества производства необходима программа испытаний, которая должна  хранить данные о сотруднике, который проводил испытание. Для реализации этого вам  потребуется создать класс с именем EmployeeInfo, который позволит пользователю ввести имя, а  затем создать id для пользователя, состоящий из первой буквы имени и фамилии.  В этом классе будет 2 поля  • StringBuilder name;  • String code;  В классе будут определены следующие методы:  • public StringBuilder getName()  • public String getCode()  • private void setName()  • private void createEmployeeCode(StringBuilder name)  • private String inputName()  • private boolean checkName(StringBuilder name)  Метод setName() будет вызываться из конструктора, который будет использовать inputName() для  получения имени (имени и фамилией) в виде единого ввода пользователя, перед тем как  вызывается метод checkName() для проверки наличия пробела во введенном имени.  Если введенное имя корректно, то вызывается метод createEmployeeCode(), который берет первую  букву имени и добавляет к фамилии для создания кода (id). Если пробела нет, то в качестве кода  будет использоваться значение guest.  Создайте в классе TestProductionLine объект employee, используя класс EmployeeInfo. Используйте  метод getCode() чтобы вывести код сотрудника в конце вывода данных о продукте.  Шаг 19:  Для провдения испытаний требуется дополнительная информация о подразделении сотрудника.  Код подразделения состоит из четырех букв и двух цифр. Формат кода подразделения: первая  буква заглавная, три последующих - строчные, и среди них не должно быть пробелов.  В класс EmployeeInfo нужно добавить три поля:  • String deptId;  • Pattern p;  • Scanner in;  Необходимо определить следующие методы:  • public String getDeptId()  • private void setDeptId()  • private String getId()  • private boolean validId(String id)  @author (Зыкин Герман Леонидович)   \*/  public class EmployeeInfo {    StringBuilder name = new StringBuilder();    String code;    String deptId;    Pattern p;    Scanner in;    /\*\*     \* Шаг 20     \* Чтобы предотвратить утечку информации, необходимо шифровать данные, сохраняемые в файл.  Чтобы выполнить это требование добавьте новый метод в класс EmployeeInfo, который будет  "переворачивать" текст, хранящийся в department id. Это нужно сделать рекурсивно с  использованием метода reverseString().  Нужно объявить следующий метод:  • public String reverseString(String id)  Если предоставлен корректный код подразделения (department id), то, перед тем как полю  deptId присваивается значение, вызывается reverseString().     \*/    public String reverseString(String id) {      char letter = id.charAt(id.length() - 1);      if (id.length() == 1) {        return Character.toString(letter);      } else {        return letter + reverseString(id.substring(0, id.length() - 1));      }    }    public String getDeptId() {      System.out.println("Введите номер отдела:");      String input2 = in.nextLine();      return input2;    }    private void setDeptId() {      String inputId = getDeptId();      if (validId(inputId)) {        deptId = reverseString(inputId);        deptId = inputId;      } else {        deptId = "none";      }    }    private String getId() {      return deptId;    }    private boolean validId(String id) {      Boolean matches = false;      Matcher matcher = p.matcher(id);      matches = matcher.matches();      return matches;    }    public StringBuilder getName() {      return name;    }    public String getCode() {      return code;    }    private void setName() {      name.append(inputName());    }    private void createEmployeeCode(StringBuilder name) {      Boolean containsSpace = name.toString().contains(" ");      if (containsSpace) {        code = name.toString().substring(0, 1);        int spot = name.toString().indexOf(" ");        int spot1 = spot;        code += name.toString().substring(++spot, name.length());      } else {        code = "гость";      }    }    private String inputName() {      System.out.print("Введите ваше ФИО: ");      String input1 = in.nextLine();      return input1;    }    private boolean checkName(StringBuilder name) {      Boolean validInput = false;      Boolean containsSpace = name.toString().contains(" ");      if (containsSpace) {        validInput = true;      }      return validInput;    }    public EmployeeInfo() {      in = new Scanner(System.in);      setName();      checkName(name);      createEmployeeCode(name);      String patternString = "([A-Z])[a-z][a-z][a-z]\\d\\d";      p = Pattern.compile(patternString);      setDeptId();      in.close();    }    @Override    public String toString() {      return "Номер сотрудника : " + code + "\n" +          "\nНомер отдела : " + deptId;    }  } |

1. **Шаг 21:**

|  |
| --- |
| OraclProduction Ltd требует сохранять результаты испытаний и сотрудника, который их провел,  в файл для последующего использования. Для этого создайте новый класс с именем ProcessFiles,  который будет создавать директорию и набор файлов, объединяя путь и имя файла. В нем будут  хранится данные о продукте и сотруднике, поэтому создайте два метода, которые будут  перегружать друг друга, которые могут принимать как объект employee, так и коллекцию  products.  Директория и файл должны создаваться при создании объекта.  В классе будет три поля:  • private Path p;  • private Path p2;  • private Path p3;  Должны быть объявлены следующие методы:  • private void CreateDirectory()  • public void WriteFile(EmployeeInfo emp) throws IOException  • public void WriteFile(ArrayList<Product> products) throws IOException  Если файл или директория не существуют, их необходимо создать. Директория должна быть  создана в корне диска C и иметь имя LineTests(p). Файл должен называться TestResults.txt(p2) и  создаваться в директории LineTests. Поле p3 нужно использовать для хранения разрешенного  (resolved) пути, объединяющего пути p и p2.  Обновите класс TestProductionLine и используйте метод WriteFile(), чтобы сохранить в файл  информацию о продукте и сотруднике. Информация должна дописываться в файл, чтобы  избежать потери истории.  Протестируйте вывод в файл, чтобы убедиться, что формат вывода такой же, как и в консоли. |

**Результат:** ProcessFiles.java

|  |
| --- |
| import java.io.FileWriter;  import java.io.IOException;  import java.io.PrintWriter;  import java.nio.file.Files;  import java.nio.file.Path;  import java.nio.file.Paths;  import java.util.ArrayList;  import java.io.FileWriter;  /\*\*   \* OraclProduction Ltd требует сохранять результаты испытаний и сотрудника, который их провел,  в файл для последующего использования. Для этого создайте новый класс с именем ProcessFiles,  который будет создавать директорию и набор файлов, объединяя путь и имя файла. В нем будут  хранится данные о продукте и сотруднике, поэтому создайте два метода, которые будут  перегружать друг друга, которые могут принимать как объект employee, так и коллекцию  products.  Директория и файл должны создаваться при создании объекта.  В классе будет три поля:  • private Path p;  • private Path p2;  • private Path p3;  Должны быть объявлены следующие методы:  • private void CreateDirectory()  • public void WriteFile(EmployeeInfo emp) throws IOException  • public void WriteFile(ArrayList<Product> products) throws IOException  Если файл или директория не существуют, их необходимо создать. Директория должна быть  создана в корне диска C и иметь имя LineTests(p). Файл должен называться TestResults.txt(p2) и  создаваться в директории LineTests. Поле p3 нужно использовать для хранения разрешенного  (resolved) пути, объединяющего пути p и p2.  Обновите класс TestProductionLine и используйте метод WriteFile(), чтобы сохранить в файл  информацию о продукте и сотруднике. Информация должна дописываться в файл, чтобы  избежать потери истории.  Протестируйте вывод в файл, чтобы убедиться, что формат вывода такой же, как и в консоли.  @author (Зыкин Герман Леонидович)  \*/  public class ProcessFiles {    private Path p;    private Path p2;    private Path p3;    public ProcessFiles() {      p = Paths.get("D:\\LineTests");      p2 = Paths.get("TestResults.txt");      p3 = p.resolve(p2);      CreateDirectory();    }    private void CreateDirectory() {      try {        Files.createDirectories(p);      } catch (Exception ex) {        System.out.println("Невозможно создать директорию");      }    }    public void WriteFile(EmployeeInfo emp) throws IOException {      FileWriter writer = new FileWriter(p3.toString(), true);      PrintWriter printWriter = new PrintWriter(writer);      printWriter.print(emp.toString());      printWriter.close();    }    public void WriteFile(ArrayList<Product> products) throws IOException {    }  } |

1. **Шаг 23:**

|  |
| --- |
| Создайте программу, которая позволит пользователю добавить новый продукт и указать  количество единиц этого продукта, которые должны быть созданы. Затем программа должна  создать эти объекты и добавить их в коллекцию.  Коллекцию можно вывести в любой момент.  Также можно вывести статистику производства – общее количество произведнной продукции,  количество единиц каждого вида, количество уникальных продуктов и т.д.  Оператор будет сам решать, когда завершать работу приложения. В программе должно быть  меню для управления из консоли или через графический интерфейс. |

**Результат:** OrclProductionline.java

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  import java.util.Scanner;  import java.util.function.Predicate;  /\*\*   \* Шаг 23 Создайте программу, которая позволит пользователю добавить новый продукт и указать  количество единиц этого продукта, которые должны быть созданы. Затем программа должна  создать эти объекты и добавить их в коллекцию.  Коллекцию можно вывести в любой момент.  Также можно вывести статистику производства – общее количество произведнной продукции,  количество единиц каждого вида, количество уникальных продуктов и т.д.  Оператор будет сам решать, когда завершать работу приложения. В программе должно быть  меню для управления из консоли или через графический интерфейс.  @author (Зыкин Герман Леонидович)   \*/  public class OrclProductionline {          private static List<Product> list = new ArrayList<>();      private static Scanner scanner = new Scanner(System.in);        public static void main(String[] args) {          while (true)          {              System.out.println("------------------------------------------------------");              System.out.println("Выберите действие: \n 1 - Создать продукт \n 2 - Вывести продукты \n 3 - Вывести статистику \n 4 - Выйти из программы");              String s = scanner.nextLine();              int opt;              try              {                  opt = Integer.parseInt(s);              }              catch (NumberFormatException e)              {                  System.out.println("неподдерживаемая операция!");                  continue;              }              switch (opt)              {                  case 1: runCreationScript(); break;                  case 2: displayProducts(); break;                  case 3: displayStats(); break;                  case 4: System.exit(0); break;                  default: System.out.println("что??");              }              System.out.println('\n');          }      }      private static void displayStats()      {          System.out.println(                  "Всего продуктов          : " + list.size() + '\n' +                  "Кол-во Аудиоплееров        : " + list.stream().filter(product -> product instanceof AudioPlayer).count() + '\n' +                  "Кол-во Видеоплееров        : " + list.stream().filter(product -> product instanceof MoviePlayer).count() + '\n');      }      private static void displayProducts()      {          System.out.println(list);      }      private static void runCreationScript()      {          System.out.println("Выберите тип создаваемого продукта: \n 1 - Аудио плеер \n 2 - Видеоплеер");          String s = scanner.nextLine();          int opt;          try          {              opt = Integer.parseInt(s);              if (opt != 1 && opt != 2)              {                  System.out.println("Некорректное значение ввода!");                  return;              }          }          catch (NumberFormatException e)          {              System.out.println("Некорректное значение ввода!");              return;          }          if (opt == 1)          {              System.out.println("Введите наименование аудиоплеера:");              String name = scanner.nextLine();              System.out.println("Введите спецификацию аудиоплеера:");              list.add(new AudioPlayer(name, scanner.nextLine()));              return;          }          System.out.println("Введите наименование видеоплеера:");          String name = scanner.nextLine();          System.out.println("Введите разрешение экрана:");          String res = scanner.nextLine();          System.out.println("Введите частоту экрана целым числом:");          s = scanner.nextLine();          int rr;          try          {              rr = Integer.parseInt(s);              if (rr < 1)              {                  System.out.println("Некорректное значение ввода!");                  return;              }          }          catch (NumberFormatException e)          {              System.out.println("Некорректное значение ввода!");              return;          }          System.out.println("Введите отклик экрана целым числом больше 0:");          s = scanner.nextLine();          int rt;          try          {              rt = Integer.parseInt(s);              if (rt < 1)              {                  System.out.println("Некорректное значение ввода!");                  return;              }          }          catch (NumberFormatException e)          {              System.out.println("Некорректное значение ввода!");              return;          }          System.out.println("Введите тип монитора (LED / LCD):");          s = scanner.nextLine();          if (!s.equals("LED") && !s.equals("LCD"))          {              System.out.println("Некорректное значение ввода!");              return;          }          MonitorType monitorType;          if (s.equals("LCD"))              monitorType = MonitorType.LCD;          else              monitorType = MonitorType.LED;          Screen screen = new Screen(res, rr, rt);          list.add(new MoviePlayer(name, screen, monitorType));      }  } |