Лабораторная работа № 3: Интерфейсы и абстрактные классы

1. Создайте файл `AbstractExample.java`, демонстрирующий работу с абстрактными классами и их методами:

```
// абстрактный класс
abstract class Storage {
    // сохранить данные (абстрактный метод)
    public abstract void store(String data);
    // чтение данных (абстрактный метод)
    public abstract String read();
}
// класс, реализующий абстрактный
class CD extends Storage {
    private String data = "";
    public CD() {}
    @Override // реализация
    public void store(String data) {
        this.data += data;
        System.out.println("Data stored on CD");
    }
    @Override // реализация
    public String read() {
        return this.data;
    }
}
// класс, реализующий абстрактный
class SDCard extends Storage {
    private String data = "";
    public SDCard() {}
    @Override // реализация
    public void store(String data) {
        this.data += data;
        System.out.println("Data stored on SDCard");
    }
    @Override // реализация
    public String read() {
        return this.data;
    }
}

public class AbstractExample {
    public static void main(String[] args) {
        // создание объекта CD
        CD cd = new CD();
        cd.store("Some data to CD");
        System.out.println(cd.read());
        // создание объекта SDCard
        SDCard sd = new SDCard();
        sd.store("Some data to SDCard");
        System.out.println(sd.read());
        // попытка создать объект класса Storage
        // требует реализации абстрактных методов
```

```
        Storage storage = new Storage() {
            private String data = "";
            @Override
            public void store(String data) {
                this.data += data;
                System.out.println("Data stored on storage");
            }
            @Override
            public String read() {
                return this.data;
            }
        };
        storage.store("Some data to storage");
        System.out.println(storage.read());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Data stored on CD
Some data to CD
Data stored on SDCard
Some data to SDCard
Data stored on storage
Some data to storage
```

Абстрактные классы и методы

Абстрактные методы служат для объявления сигнатуры методов без его реализации (тела метода). Абстрактный метод объявляется с использованием ключевого слова `abstract`:

```
abstract void erase();
```

Если класс содержит хоть один абстрактный метод, то он считается абстрактным и должен быть объявлен с использованием ключевого слова `abstract`:

```
abstract class Storage { . . . }
```

Абстрактные классы в Java широко используются при наследовании для определения шаблонов реализации классов, унаследованных от абстрактного. В показанном примере объявлен класс `Storage` с абстрактными методами `store` и `read` для записи и чтения данных соответственно. От него унаследованы классы `CD` и `SDCard`, реализующие методы в соответствии со списком, описанным в классе `Storage`. Все абстрактные методы, указанные в абстрактном классе, обязательно должны быть реализованы в унаследованных классах. При попытке создать объект абстрактного класса, необходимо напрямую указывать реализацию его абстрактных в фигурных скобках после создания объекта, в ином случае возникнет ошибка компиляции, например:

```
Storage storage = new Storage() {
    private String data = "";
    @Override
```

```
    public void store(String data) {
        this.data += data;
        System.out.println("Data stored on storage");
    }
    @Override
    public String read() {
        return this.data;
    }
};
```

2. Создайте абстрактный класс `Gift`, объявив для него абстрактные методы:
   - `buy()` – без аргументов и возвращаемого значения, выводи на экран информацию, о том, что подарок куплен;
   - `give()` – принимает на вход имя получателя в виде строки, без возвращаемого значения.

Создайте следующие классы, унаследованные от `Gift`:
   - `Postcard` – класс реализует абстрактные методы класса `Gift`, дополнительно к которым имеет строковое поле персонального пожелания wish, метод `writeWish()`, принимающий на вход текст поздравления, метод `getWish()`, возвращающий значение пожеланий (`wish`). Реализованный метод give выводит на экран пожелание с подставленным в него именем получателя.
   - `Painting` – класс реализует абстрактные методы класса `Gift`, дополнительно к которым имеет строковые поля `title` и `author` и геттеры и сеттеры для данных полей.

Создайте запускаемый класс `GiftSharing`, в котором реализуйте проверку всех методов.

3. Создайте файл `StructureUnit.java`, описывающий интерфейс классов, описывающих структурное подразделение университета:

```
public interface StructureUnit {
    String university = "ITMO University";
    // нанять сотрудника
    void hireEmployee(String name);
    // уволить сотрудника
    void fireEmployee(String name);
    String getInfo();
}
```

4. Создайте файл `Department.java`, реализующий интерфейс `StructureUnit`:

```
import java.util.ArrayList;
import java.util.Arrays;
public class Department implements StructureUnit {
    private String title;
    private    ArrayList<String>    employees;    //    множество
сотрудников
    public Department() { this.employees = new ArrayList<>(); }
    public void setTitle(String title) { this.title = title; }
    // нанять сотрудника
    public         void         hireEmployee(String         name){
this.employees.add(name); }
```

```
    // уволить сотрудника
    public void fireEmployee(String name){
this.employees.remove(name); }
    // получение информации о структурной единице
    public String getInfo(){
        int num = this.employees.size();
        return "Title: " + this.title + " | Number of employees:
"  +  num  +  "  |  Employees:  "  +
Arrays.toString(this.employees.toArray());
    }
}
```

5. Создайте файл `Faculty.java`, реализующий интерфейс `StructureUnit`:

```
import java.util.ArrayList;
import java.util.Arrays;
public class Faculty implements StructureUnit {
    String title;
    private ArrayList<Department> departments;
    private  ArrayList<String>  employees;  //  множество
сотрудников
    public Faculty() {
        this.employees = new ArrayList<>();
        this.departments = new ArrayList<>();
    }
    public void setTitle(String title) { this.title = title; }
    // нанять сотрудника
    public void hireEmployee(String name){
        this.employees.add(name);
    }
    // уволить сотрудника
    public void fireEmployee(String name){
        this.employees.remove(name);
    }
    // добавить вложенную структ. единицу
    public void addDepartment(Department department) {
        this.departments.add(department);
    }
    // получение информации о структурной единице
    public String getInfo() {
        int num = this.employees.size();
        String info =  "Title: " + this.title + " | Number of
employees:  "  +  num  +  "  |  Employees:  "  +
Arrays.toString(this.employees.toArray());
        info += "\n Info from departments: \n";
        for (Department d : departments) info += d.getInfo() +
"\n";
        return info;
    }
}
```

6. Создайте файл `InterfaceExample.java`:

```
public class InterfaceExample {
    public static void main(String[] args) {
        Department department = new Department();
        department.setTitle("Department of CET");
        department.hireEmployee("John");
        System.out.println(department.getInfo());
        Faculty faculty = new Faculty();
        faculty.setTitle("Faculty of SECS");
        faculty.addDepartment(department);
        faculty.hireEmployee("Jack");
        System.out.println(faculty.getInfo());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Title: Department of CET | Number of employees: 1 | Employees:
[John]
Title: Faculty of SECS | Number of employees: 1 | Employees:
[Jack]
 Info from departments:
Title: Department of CET | Number of employees: 1 | Employees:
[John]
```

Ключевое слово `final`

При использовании ключевого слова `final` с полями класса и переменными:

- инициализация значения переменной происходит один раз;
- ссылка на переменную, объявленную с помощью `final`, не может быть изменена;
- при использовании совместно с модификатором `static` позволяет определить константу для класса, например: `static final double pi = 3.14`.

Использование ключевого слова `final` с методами класса запрещает наследование этого метода любым классом, унаследованным от текущего. Объявление метода с модификатором `final`:

```
public final void f() { . . .}
```

Создание класса как `final` позволяет запретить наследование любых методов данного класса. Объявление класса с модификатором `final`:

```
public final class Finalist { . . .}
```

Интерфейсы

Интерфейсы в Java представляют собой полностью абстрактные классы. Все методы интерфейсов по умолчанию являются абстрактными методами и не требуют наличия модификатора `abstract` при их объявлении. Объявление интерфейса происходит при помощи ключевого слова `interface` следующим образом:

```
interface Food {
```

```
    Boolean eatable = true; // static, final
    void eat();
}
```

В приведенном примере создается интерфейс `Food` с (абстрактным) методом `eat()` и полем `eatable`. Все поля интерфейса по умолчанию являются `static` и `final`. Использование модификатора `public` перед ключевым словом `interface` позволяет сделать интерфейс общедоступным, иначе, по умолчанию, он будет виден исключительно внутри текущего пакета.

Реализация методов, указанных в интерфейсе, осуществляется в классах, реализующих данный интерфейс, объявленных с использованием ключевого слова `implements`. Методы интерфейса, реализованные в классе в обязательном порядке должны быть объявлены с модификатором `public`.

```
public class Tomato implements Food {
    public void eat(){
        System.out.println("Tomato eaten");
    }
}
```

Таким образом, интерфейс определяет форму тех классов, которые будут реализовывать этот интерфейс. Основными отличиями интерфейсов от абстрактных классов являются:

- отсутствие неабстрактных методов;
- все поля интерфейса являются `static final`;
- все методы интерфейса по умолчанию `public` и должны оставаться `public` при реализации;
- интерфейс может расширять другой интерфейс (в том числе базовые интерфейсы Java), но не может быть унаследованным от класса;
- для интерфейса не могут быть созданы никакие объекты данного типа, объекты могут быть созданы исключительно для классов, реализующих интерфейс.

7. Логически разделите устройства из следующего списка: центральный процессор, основная память, шина, клавиатура, мышь, принтер, жесткий диск, монитор, на смысловые категории, для каждой из которых создайте свой интерфейс с основными методами, необходимыми для реализации. В каждом из реализующих классов, созданных в соответствии с указанным перечнем устройств компьютера, создайте по одному дополнительному методу, специфичному для конкретного устройства. В отдельно созданном запускаемом классе осуществите проверку всех классов и их методов.

8. Создайте файл `DocumentFactory.java`, описывающий интерфейс фабрики документов:

```
public interface DocumentFactory {
    Document getDocument();
}
```

9. Создайте файл `Document.java`, описывающий интерфейс отдельного документа:

```
public interface Document {
```

```
    void sign(String name);
    String getSignatures();
    void printInfo();
}
```

10. Создайте файл `Regulation.java`, описывающий класс документа-положения:

```
public class Regulation implements Document{
    private String signatures = "";
    @Override
    public void sign(String name) {
        if (this.signatures.length()>0)
            this.signatures += ", ";
        this.signatures += name;
    }
    @Override
    public String getSignatures() { return this.signatures; }
    @Override
    public void printInfo() {
        System.out.println("Regulation info: \n" +
                           "Title: Regulation \n" +
                           "Signed by: " + this.signatures);
    }
}
```

11. Создайте файл `RegulationFactory.java`, описывающий класс фабрики документов-положений:

```
public class RegulationFactory implements DocumentFactory {
    @Override
    public Document getDocument() {
        return new Regulation();
    }
}
```

12. Создайте файл `Contract.java`, описывающий класс документа-договора:

```
public class Contract implements Document{
    private String signatures = "";
    @Override
    public void sign(String name) {
        if (this.signatures.length()>0)
            this.signatures += ", ";
        this.signatures += name;
    }
    @Override
    public String getSignatures() { return this.signatures; }
    @Override
    public void printInfo() {
        System.out.println("Contract info: \n" +
                "  Title: Contract \n" +
                "  Signed by: " + this.signatures);
    }
```

```
}
```

13. Создайте файл `ContractFactory.java`, описывающий класс фабрики документов-договоров:

```
public class ContractFactory implements DocumentFactory {
    @Override
    public Document getDocument() {
        return new Contract();
    }
}
```

14. Создайте файл `FactoriesExample.java`:

```
public class FactoriesExample {
    public static void workWithDocument(DocumentFactory dF){
        Document doc = dF.getDocument();
        doc.sign("Paul");
        doc.sign("Jack");
        System.out.println(doc.getSignatures());
        doc.printInfo();
    }
    public static void main(String[] args){
        workWithDocument(new RegulationFactory());
        System.out.println();
        workWithDocument(new ContractFactory());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Paul, Jack
Regulation info:
 Title: Regulation
 Signed by: Paul, Jack

Paul, Jack
Contract info:
 Title: Contract
 Signed by: Paul, Jack
```

Множественная реализация интерфейсов

Разработка с использованием фабрик (или множественных реализаций интерфейсов) позволяет программисту не работать напрямую с определенным типом объекта, а осуществлять создание объекта с использованием нужной ему фабрики, определяющей тип автоматически.
Фабрики позволяют:

- разработать более защищенное от случайных ошибок приложение;
- разработать фреймворки, для которых характерно наличие схожих методов (в

частности это может быть использовано при программировании игр: например, шахматы и шашки, для которых игра ведется похожим образом на одном и том же поле), или же разработке систем анализа различных документов и т.д.

15. Создайте пакет `factory`, содержащий классы Java-приложения, построенного с применением множественной реализации интерфейсов. Разработайте систему в соответствии с вашим вариантом задания. Подберите не менее трех различных примеров типов системы (например, для системы «Зарядное устройство» – проводное и беспроводное), отличающихся между собой по каким-либо признакам. Для общего интерфейса системы создайте методы, указанные в вашем варианте, реализуя их в классах определенных на этапе разработки типов системы. Каждый из методов должен выводить в терминал отличительную информацию для конкретного рассматриваемого типа системы. По окончании работы создайте запускаемый класс для проверки Java-приложения, используйте задание нескольких базовых значений характеристик системы для реализации всех необходимых функций.

Вариант	**Описание систем**
1	Система: «Словарь» Методы: поиск слова в словаре, поиск синонима слова, получение информации о словаре
2	Система: «Выставка» Методы: просмотреть информацию о выставке, купить билет, посетить
3	Система: «Музей» Методы: просмотреть информацию о музее, посетить, приобрести разрешение на фотосъемку
4	Система: «Компьютер» Методы: установить операционную систему, изменить настройки системы, подключить клавиатуру
5	Система: «Морское судно» Методы: отправиться в путь, пришвартоваться, получить информацию о судне
6	Система: «Накопитель данных» Методы: сохранить данные, удалить данные, узнать состояние накопителя
7	Система: «Растение» Методы: посадить растение, полить растение, получить информацию о растении
8	Система: «Университет» Методы: учредить, определить структуру, нанять сотрудников
9	Система: «Отпуск» Методы: запланировать отпуск, установить тип отпуска, подписать заявление об отпуске
10	Система: «Аккумулятор» Методы: подключить, проверить статус заряда, зарядить
11	Система: «Автомобиль» Методы: получить информацию об автомобиле, установить дополнительные модули, заполнить бак/зарядить
12	Система: «Выигрыш» Методы: забрать выигрыш, получить информацию о выигрыше, выполнить действие (потратить/начать копить)
13	Система: «Регулятор температуры» Методы: увеличить температуру, уменьшить температуру, изменить режим работы

14	Система: «Принтер» Методы: напечатать документ, проверить очередь печати, проверить картридж
15	Система: «Лента новостей» Методы: просмотреть новости, добавить новость в избранное, удалить новость из избранного
16	Система: «Сувенир» Методы: подарить, упаковать, поместить на полку
17	Система: «Часы» Методы: установить время, узнать время, надеть часы
18	Система: «Монитор» Методы: настроить разрешение, настроить яркость, настроить контрастность
19	Система: «Клавиатура» Методы: подключить клавиатуру, изменить раскладку, включить дополнительный блок кнопок (напр. NumLock)
20	Система: «Расписание» Методы: создать запись расписания, применить шаблон, получить информацию о расписании на заданные даты
21	Система: «Телевизор» Методы: получить информацию о телевизоре, настроить, переключить канал
22	Система: «Сигнализация» Методы: подключить, получить информацию об устройстве, просмотреть статистику
23	Система: «Сооружение» Методы: построить, спроектировать, разрушить
24	Система: «Поезд» Методы: назначить маршрут, изменить характеристики, отправить в путь
25	Система: «Тренажер» Методы: установить нагрузку, просмотреть статистику упражнений, подключить
26	Система: «Магазин» Методы: нанять сотрудников, приобрести товар, получить информацию о магазине
27	Система: «Склад» Методы: выдать продукцию, получить продукцию, получить информацию о состоянии склада
28	Система: «Стадион» Методы: назначить мероприятие, узнать расписание, настроить освещение
29	Система: «Парк» Методы: посадить растение, установить фонтан, поставить беседку
30	Система: «Таможня» Методы: проверить груз, уплатить пошлину, арестовать груз
31	Система: «Фабрика» Методы: отгрузить продукцию, получить сырье, нанять сотрудников
32	Система: «Поисковик» Методы: выполнить запрос, сформировать отчет, разместить карточку ресурса
33	Система: «Офис продаж» Методы: провести анализ рынка, сформировать стратегию, сформировать отчет
34	Система: «Компания» Методы: зарегистрировать компанию, нанять сотрудников, арендовать помещение

35	Система: «Почтовая служба» Методы: принять отправление, доставить отправление, выдать отправление
36	Система: «Чемпионат» Методы: принять регламент, сформировать призовой фонд, определить победителей
37	Система: «Группа» Методы: сформировать группу, определить структуру, добавить членов группы
38	Система: «Медицинское учреждение» Методы: закупить лекарства, сформировать расписание приемных часов, сформировать историю болезни
39	Система: «Школа» Методы: назначить директора, установить расписание, приобрести оборудование
40	Система: «Космическое тело» Методы: исследовать, сформировать траекторию, изменить траекторию
41	Система: «Сеть» Методы: добавить узел, установить связь, проложить маршрут
42	Система: «Мебель» Методы: собрать, установить, переместить
43	Система: «Онлайн-магазин» Методы: сделать заказ, узнать статус заказа, получить информацию о магазине
44	Система: «Граница» Методы: установить границу, переместить границу, удалить границу
45	Система: «Резервуар» Методы: наполнить, опустошить, нагреть
46	Система: «Культурно-массовое мероприятие» Методы: определить программу, продать билеты, посетить
47	Система: «Аттестующее мероприятие» Методы: установить шкалу, сформировать задание, сформировать отчет
48	Система: «Издательство» Методы: принять материалы для публикации, сформировать план публикаций, издать публикацию
49	Система: «Университет» Методы: учредить, определить структуру, нанять сотрудников
50	Система: «Отношение» Методы: установить отношение, изменить отношение, удалить отношение