ИНТЕРФЕЙСЫ

Назначение интерфейса — описание или спецификация функциональности,  
которую должен реализовывать каждый класс, его имплементирующий. Класс,  
реализующий интерфейс, предоставляет к использованию объявленный интерфейс в виде набора public-методов в полном объеме.  
Интерфейсы подобны абстрактным классам, хотя и не являются классами.  
В языке Java существует три вида интерфейсов: интерфейсы, определяющие  
функциональность для классов посредством описания методов, но не их реализации; функциональные интерфейсы, специфицирующие в одном абстрактном методе свое применение; интерфейсы, реализация которых автоматически придает классу определенные свойства. К последним относятся, например, интерфейсы Cloneable, Serializable, Externalizable, отвечающие за клонирование и сохранение объекта (сериализацию) в информационном потоке соответственно.

Общее определение интерфейса имеет вид:

[public] interface Name [extends NameOtherInterface,…, NameN] {  
// constants, methods

}

Все объявленные в интерфейсе абстрактные методы автоматически трактуются как public abstract, а все поля — как public static final, даже если они так  
не объявлены. Интерфейсы могут объявлять статические методы. В интерфейсах могут объявляться методы с реализацией с ключевым словом default. Эти  
методы могут быть public или private.

Класс может реализовывать любое число интерфейсов, указываемых через  
запятую после ключевого слова implements, дополняющего определение класса. После этого класс обязан реализовать все абстрактные методы, полученные  
им от интерфейсов, или объявить себя абстрактным классом.  
Если необходимо определить набор функциональности для какого-либо  
рода деятельности, например, для управления счетом в банке, то следует использовать интерфейс вида:

/\* # объявление интерфейса управления банковским счетом #AccountAction.java \*/package by.learn.advanced;public interface AccountAction{boolean openAccount();boolean closeAccount();void blocking();default void unBlocking() {}double depositInCash(int accountId, int amount);boolean withdraw(int accountId, int amount);boolean convert(double amount);boolean transfer(int accountId, double amount);}

В интерфейсе обозначены, но не реализованы, действия, которые может  
производить клиент со своим счетом. Реализация не представлена из-за возможности различного способа выполнения действия в конкретной ситуации.

А именно: счет может блокироваться автоматически, по требованию клиента или администратором банковской системы. В каждом из трех указанных случаев реализация метода **blocking()** будет уникальной и никакого базового решения предложить невозможно. С другой стороны, наличие в интерфейсе метода заставляет класс, его имплементирующий, предоставить реализацию методу. Программист получает повод задуматься о способе реализации функциональности, так как наличие метода в интерфейсе говорит о необходимости той или иной функциональности всем классам, реализующим данный интерфейс.

Метод **unblocking()** предоставляет дефолтную реализацию метода, которая может быть при необходимости переопределена в подклассе.  
Интерфейс следует проектировать ориентированным на выполнение близких по смыслу задач, например, отделить действия по созданию, закрытию  
и блокировке счета от действий по снятию и пополнению средств. Такое разделение разумно в ситуации, когда клиенту посредством Интернет-системы не предоставляется возможность открытия, закрытия и блокировки счета.

Тогда вместо одного общего интерфейса можно записать два специализированных: один для администратора, второй — для клиента.

/\* # 2 # общее управление банковским счетом # AccountBaseAction.java \*/package by.learn.advanced;public interface AccountBaseAction {boolean openAccount();boolean closeAccount();void blocking();default void unBlocking() {}}

/\* # 3 # операционное управление банковским счетом # AccountManager.java \*/package by.learn.advanced;public interface AccountManager {double depositInCash(int accountId, int amount);boolean withdraw(int accountId, int amount);boolean convert(double amount);boolean transfer(int accountId, double amount);}

Реализация интерфейса при реализации всех методов выглядит следующим образом:  
/\* # 4 # реализация общего управления банковским счетом# AccountBaseActionImpl.java \*/package by.learn.advanced.impl;import by.learn.advanced.AccountBaseAction;public class AccountBaseActionImpl implements AccountBaseAction {public boolean openAccount() {// more code}public boolean closeAccount() {// more code}public void blocking() {// more code}public void unBlocking() {// more code}}

Если по каким-либо причинам для данного класса не предусмотрена реализация метода или его реализация нежелательна, рекомендуется генерация непроверяемого исключения в теле метода, а именно:

public boolean blocking() {throw new UnsupportedOperationException();}Менее хорошим примером будет реализация в виде:public boolean blocking() {return false;}

так как пользователь метода может считать реализацию корректной.   
В интерфейсе не могут быть объявлены поля без инициализации, интерфейс в качестве полей содержит только *final static* константы.

Множественное наследование между интерфейсами допустимо. Классы,  
в свою очередь, интерфейсы только реализуют. Класс может наследовать один  
суперкласс и реализовывать произвольное число интерфейсов. В Java интерфейсы обеспечивают большую часть той функциональности, которая в C++ представляется с помощью механизма множественного наследования.  
При именовании интерфейса в конец его названия может добавляться able  
в случае, если в названии присутствует действие (глагол). Не рекомендуется  
в имени интерфейса использовать слово Interface или любое его сокращение.  
В конец имени класса, реализующего интерфейс, для указания на источник  
действий можно добавлять слово Impl. Реализации интерфейсов помещать  
следует в подпакет с именем impl. Интерфейсы можно применить, например, в задаче, где на поверхности линии соединяются в фигуры. Описание функционала методов по обработке информации о линиях и фигурах:

/\* # 5 # объявление интерфейсов # LineGroupAction.java \*/package by.learn.advanced;  
import by.learn.entity.AbstractShape;  
public interface LineGroupAction {  
double computePerimeter(AbstractShape shape);  
}  
/\* # 6 # наследование интерфейсов # ShapeAction.java \*/package by.learn.advanced;  
import by.learn.entity.AbstractShape;  
public interface ShapeAction extends LineGroupAction {  
double computeSquare(AbstractShape shape);  
}

Класс, который будет реализовывать интерфейс ShapeAction, должен будет определить все методы из цепочки наследования интерфейсов. В данном случае это методы computePerimeter() и computeSquare().

**/\* # 7 # абстрактная фигура # AbstractShape.java \*/**  
package by.learn.entity;  
public class AbstractShape {  
private long shapeId;  
public long getShapeId() {  
return shapeId;  
}  
public void setShapeId(long shapeId) {  
this.shapeId = shapeId;  
}  
**}**

Интерфейсы обычно объявляются как public, потому что описание функциональности, предоставляемое ими, может быть использовано в других пакетах проекта. Интерфейсы с областью видимости в рамках пакета, атрибут доступа по умолчанию, могут использоваться только в этом пакете и нигде более.

Реализация интерфейсов классом может иметь вид:  
[public] class NameClass implements Name1, Name2,…, NameN {/\* \*/}  
Здесь *Name1, Name2,…, NameN* — перечень используемых интерфейсов.  
Класс, который реализует интерфейс, должен предоставить полную реализацию всех абстрактных методов, объявленных в интерфейсе. Кроме того, данный класс может объявлять свои собственные методы.

\* # 8 # реализация интерфейса # RectangleAction.java \*/package by.learn.advanced.impl;  
import by.learn.entity.AbstractShape;  
import by.learn.entity.Rectangle;  
import by.learn.advanced.ShapeAction;  
public class RectangleAction implements ShapeAction {  
@Override  
public double computeSquare(AbstractShape shape) {  
double square;  
if (shape instanceof Rectangle rectangle) {  
square = rectangle.getHeight() \* rectangle.getWidth();  
} else {  
throw new IllegalArgumentException("Incompatible shape " + shape.getClass());  
}  
return square;  
}  
@Override  
public double computePerimeter(AbstractShape shape) {  
double perimeter;  
if (shape instanceof Rectangle rectangle) {  
perimeter = 2 \* (rectangle.getWidth() + rectangle.getHeight());  
} else {  
throw new IllegalArgumentException("Incompatible shape " + shape.getClass());  
}  
return perimeter;  
}  
}

**/\* # 9 # реализация интерфейса *# TriangleAction.java \*/***package by.learn.advanced.impl;  
import by.learn.entity.AbstractShape;  
import by.learn.entity.RightTriangle;  
import by.learn.advanced.ShapeAction;  
public class TriangleAction implements ShapeAction {

@Override  
public double computeSquare(AbstractShape shape) {  
double square;  
if (shape instanceof RightTriangle triangle) {  
square = 1./2 \* triangle.getSideA() \* triangle.getSideB();  
} else {  
throw new IllegalArgumentException("Incompatible shape " + shape.getClass());  
}  
return square;  
}  
@Override  
public double computePerimeter(AbstractShape shape) {  
double perimeter = 0;  
if (shape instanceof RightTriangle triangle) {  
double a = triangle.getSideA();  
double b = triangle.getSideB();  
perimeter = a + b + Math.hypot(a, b);  
} else {  
throw new IllegalArgumentException("Incompatible shape " + shape.getClass());  
}  
return perimeter;  
}  
}

При неполной реализации интерфейса класс должен быть объявлен как абстрактный, что говорит о необходимости реализации абстрактных методов уже в его подклассе.

/\* # 10 # неполная реализация интерфейса # PentagonAction.java \*/  
package by.learn.advanced.impl;  
import by.learn.entity.AbstractShape;  
import by.learn.advanced.ShapeAction;  
public abstract class PentagonAction implements ShapeAction {  
@Override  
public double computePerimeter(AbstractShape shape) {  
// code  
}  
}  
Классы для определения фигур, используемые в интерфейсах:  
/\* # 11 # прямоугольник, треугольник # Rectangle.java # RightTriangle.java \*/  
package by.learn.entity;  
public class Rectangle extends AbstractShape {  
private double height;  
private double width;  
public Rectangle(double height, double width) {  
this.height = height;

this.width = width;  
}  
public double getHeight() {  
return height;  
}  
public double getWidth() {  
return width;  
}  
}  
package by.epam.learn.entity;  
import java.util.StringJoiner;  
public class RightTriangle extends AbstractShape {  
private double sideA;  
private double sideB;  
public RightTriangle(double sideA, double sideB) {  
this.sideA = sideA;  
this.sideB = sideB;  
}  
public double getSideA() {  
return sideA;  
}  
public void setSideA(double sideA) {  
this.sideA = sideA;  
}  
public double getSideB() {  
return sideB;  
}  
public void setSideB(double sideB) {  
this.sideB = sideB;  
}  
@Override  
public String toString() {  
return new StringJoiner(", ", RightTriangle.class.getSimpleName() + "[", "]")  
.add("sideA=" + sideA).add("sideB=" + sideB).toString();  
}  
}  
/\* # 12 # свойства ссылок на интерфейс # ActionMain.java \*/  
package by.learn.advanced;  
import by.learn.advanced.impl.RectangleAction;  
import by.learn.advanced.impl.TriangleAction;  
import by.learn.entity.Rectangle;  
import by.learn.entity.RightTriangle;  
public class ActionMain {  
public static void main(String[] args) {  
ShapeAction action;  
try {  
Rectangle rectShape = new Rectangle(2, 5);  
action = new RectangleAction();

System.out.println("Square rectangle: "+ action.computeSquare(rectShape));  
System.out.println("Perimeter rectangle: "  
+ action.computePerimeter(rectShape));  
RightTriangle trShape = new RightTriangle(3, 4);  
action = new TriangleAction();  
System.out.println("Square triangle: " + action.computeSquare(trShape));  
System.out.println("Perimeter triangle: "  
+ action.computePerimeter(trShape));  
action.computePerimeter(rectShape); // runtime exception  
} catch (IllegalArgumentException e) {  
System.err.println(e.getMessage());  
}  
}  
}  
В результате будет выведено:  
Square rectangle: 10.0  
Perimeter rectangle: 14.0  
Square triangle: 6.0  
Perimeter triangle: 12.0  
Incompatible shape class by.learn.entity.Rectangle

Допустимо объявление ссылки на интерфейсный тип или использование ее в качестве параметра метода. Такая ссылка может указывать на экземпляр любого класса, который реализует объявленный интерфейс. При вызове метода через такую ссылку будет вызываться его реализованная версия, основанная на текущем экземпляре класса. Выполняемый метод разыскивается динамически во время выполнения, что позволяет создавать классы позже кода, который вызывает их методы.

**Вопросы:**

1. Что такое интерфейс? Как определить и реализовать интерфейс в java-программе?  
2. Можно ли описывать в интерфейсе конструкторы и создавать объекты?

3. Можно ли создавать интерфейсные ссылки и если да, то на какие объекты  
они могут ссылаться?

4. Какие идентификаторы по умолчанию имеют поля интрефейса?

5. Какие идентификаторы по умолчанию имеют методы интрефейса?

6. Чем отличается абстрактный класс от интерфейса?

7. Когда применять интерфейс логичнее, а когда абстрактный класс?

8. Бывают ли интерфейсы без методов? Для чего?

9. Могут ли классы внутри классов реализовывать интерфейсы?

10. Возможно ли анонимный класс создать на основе реализации интерфейса?

11. Привести два способа объявления статического метода в интерфейсе?

**Задания:**

Разработать проект управления процессами на основе создания и реализации интерфейсов для следующих предметных областей.

(Реализовать отдельный поверочный класс с методом main(), который реализует поставленную задачу. Весь процесс должен детально выводиться пользователю на консоль. НЕ реализовывать консольный пользовательский интерфейс.)

1. **Полиграфические издания**. Возможности: оформить договор; открыть\  
редактировать\верстать издание; отправить на печать; отказаться от издания; оплатить издание; возобновить\закрыть издание. Добавить специализированные методы для Книги, Журнала, Учебного пособия.

2. **Абонент мобильного оператора**. Возможности: оформить договор; открыть счет и номер; редактировать учетную запись абонента; проверить состояние баланса и остаток трафика; сменить тарифный план; пополнить счет; закрыть счет и номер. Добавить специализированные методы для Учетной записи интернет, реального и корпоративного абонента.

3. **Конфеты**. Возможности: выпустить партию конфет; получить информацию по продукции определенного производителя; добавить и редактировать информацию о продукции; добавить новые ингредиенты; создать новый тип конфет; снять с производства определенный тип конфет. Добавить  
специализированные методы для Конфеты, Шоколад, Леденец, Ирис.

4. **Растение.** Возможности: задать и редактировать информацию о растении; получить информацию об имеющихся растениях в оранжерее/растениях определенного вида/месте происхождения; закупить новый вид растений; произвести полив; задать температуру; установить освещение; выкопать определенный вид растений (убрать из оранжереи). Добавить специализированные методы для Кустовое растение, Цветковое растение, Комнатное растение.

5. **Авиарейс.** Возможности: получить или изменить различную информацию  
о рейсе: номер рейса, пункт назначения, марка судна, статус судна (загрузка,  
разгрузка, заправка, в пути, в ремонте, готов к вылету);  
узнать среднее время рейса; узнать снаряженную массу; узнать количество  
топлива; отправить в пункт назначения; отремонтировать; заправить; загрузить; разгрузить; узнать хрупкость/ценность груза. Добавить дополнительные возможности для грузового и пассажирского рейса.

6**. Вагон.** Возможности: получить или изменить различную информацию  
о вагоне: регистрационный номер вагона, пункт назначения, владелец, статус (загрузка, разгрузка, в пути, в ремонте, готов к отправке, требуется ремонт); узнать снаряженную массу; отправить в пункт назначения; обслужить; отремонтировать; загрузить; разгрузить; узнать хрупкость/ценность  
груза. Добавить дополнительные возможности для грузового и пассажирского вагонов.

7. **Средство передвижения**. Возможности: получить или изменить различную  
информацию о средстве передвижения: регистрационный номер, марка, модель, VIN-номер, владелец, тип движущей силы; заправить\отремонтировать\обслужить; пройти техосмотр. Добавить дополнительные возможности для автомобиля, велосипеда, самоката, мотоцикла, квадроцикла.

8. **Лекарственный препарат**. Возможности: добавить действующее вещество; рассчитать дозировку; провести исследование вещества; изменить статус вещества (запрещенное, по рецепту, разрешенное); получить и изменить информации о действующем веществе. Добавить дополнительные  
возможности для Лекарства, Мази, Таблетки и Раствора.

9**. Самолеты**. Возможности: пройти техосмотр; отремонтировать; осуществить рейс; заправить; получить\изменить информацию о транспортном средстве. Добавить дополнительные возможности для самолета (военного и гражданского самолета).

10. **Дом.** Возможности: построить дом; рассчитать цену за квадратный метр;  
узнать сколько комнат; увеличить площадь; сдавать дом в аренду; сделать  
ремонт (в какой-либо комнате). Добавить специализированные методы для  
Дома, Офисного здания, Торгового центра.

11. **Банковский вклад**. Возможности: изменить продолжительность вклада  
(бессрочный/долгосрочный/краткосрочный); узнать, какой вклад (отзывной  
или безотзывной); закрыть один вклад и открыть новый с такими же условиями, но в другой валюте; рассчитать начисление процентов по вкладу.

12. **Компьютер**. Возможности: создать новую модель компьютера; установить  
цену; добавить объем оперативной памяти; изменить разрешение экрана;  
изменить процессор компьютера; добавить новые комплектующие в базовую комплектацию. Добавить специализированные методы для Computer,  
SmartPhone, Pad.

13. **Город.** Возможности: вывести основную информацию о городе (страна,  
год основания, популяция, площадь, расположение, бюджет, действующий мэр, язык общения); выбрать нового мэра города; увеличить/уменьшить популяцию населения; редактировать бюджет; изменить статус города (закрытие на карантин при определенном проценте зараженных);  
ввести\изменить закон. Добавить специализированные методы для Города, Деревни, Хутора.

14. **Товар** (Product). Возможности: добавить товар; получить данные о товаре:  
id, наименование, производитель, цена, срок хранения, количество;  
изменить данные о товаре; переместить товар на склад, переместить товар  
в торговый зал; оплатить товар; списать товар. Добавить специализированные методы для масла, молока, творога.

15. **Пациент** (Patient). Возможности: регистрировать пациента; оформить договор обслуживания; получить персональные данные: id, фамилия, имя,  
адрес, телефон, номер медицинской карты; изменить персональные данные; записаться на прием к врачу; записать диагноз; сдать лабораторные исследования; получить назначение лечения; отказаться от обслуживания; возобновить обслуживание. Добавить специализированные  
методы для Пациента инфекционного отделения, Пациента стационара,  
Пациента с COVID-19.

16. **Покупатель** (Customer). Возможности: оформить договор; получить\изменить персональные данные; добавить покупки; получить оплату; добавить скидки; расторгнуть договор; возобновить договор. Добавить специализированные методы для Оптового покупателя, VIP-покупателя.

17. **Сувениры.** Возможности: просмотреть сувенир поставщика; создать макет; редактировать макет; оформить/расторгнуть договор с поставщиком; оплатить сувенир; отказаться от сувенира; произвести сувенир; убрать сувенир. Добавить специализированные методы для Блокнота, Майки, Магнита.

18. **Столовые приборы**. Возможности: просмотреть приборы определенного  
бренда; изменить материал изготовления приборов; произвести партию  
приборов; остановить производство приборов; оформить договор на производство приборов; оценить стоимость приборов; реставрировать приборы. Добавить специализированные методы для Вилка, Нож, Палочка для еды.

19. **Планета.** Возможности: добавить планету; сообщить об открытии; добавить/изменить характеристики; рассчитать период вращения; заказать дополнительные исследования; купить индивидуальное название; добавить снимки; напечатать отчет о планете; выбрать планеты по параметру. Добавить специализированные методы для Газовых гигантов, Ледяных гигантов, Карликовых планет.

20. **Погода.** Возможности: ввести/получить данные мониторинга метеорологических условий; запросить данные; анализировать карты погоды; анализировать климатические данные; рассчитать прогнозные показатели; уточнить прогнозные показатели; ввести фактические показатели; вывести прогноз погоды; оценить прогноз. Добавить специализированные методы для Срединный регион, Нижний регион, Верхний регион.

21. **Электроинструменты.** Возможности: создать модель инструмента; ввести характеристики инструмента; добавить/изменить комплектующие; рассчитать стоимость производства; добавить чертежи; отправить на доработку; отказать в производстве; запустить в производство; отправить на тестирование; получить результаты тестов. Добавить специализированные методы для Пиление, Резка, Обработка дерева, Сверление отверстий.

22. **Видеотека.** Возможности: создать/удалить видеотеку; создать/изменить/  
удалить актера/режиссера/фильм; отобразить информацию о самых новых/  
популярных фильмах; отобразить список фильмов; открыть фильм; улучшить/ухудшить качество видео; включить/выключить звук; закрыть фильм. Добавить специализированные методы для Библиотека, Сервер.

23. **Расписание занятий.** Возможности: создать/удалить расписание; создать/  
изменить/удалить предмет/преподавателя; отобразить расписание для конкретного преподавателя/конкретной группы; отобразить список всех предметов/преподавателей; отменить занятие; добавить дополнительное занятие; посчитать количество занятий у преподавателя/группы. Добавить специализированные методы для Расписания занятий в школе, Расписания занятий в автошколе.

24. **Письма.** Возможности: создать/удалить письмо, добавить/удалить отправителя; добавить текст письма; добавить/удалить приложения к письму; поставить дату отправки; отправить/принять письмо; проверить адресат на существование. Добавить специализированные методы для Заказного письма, Письма с доставкой, Обычного письма.

25. **Литературное произведение**. Возможности: написать произведение; отредактировать произведение; опубликовать произведение; посвятить произведение кому-либо/чему-либо; перевести на другой язык; продать права на использование произведения; получить критику на произведение. Добавить  
дополнительные возможности для Стихотворение, Белый стих.