

Полиморфизм. Интерфейсы



ПРЕПОДАВАТЕЛЬ



Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

ПЛАН ЗАНЯТИЯ

1. Основной блок
2. Вопросы по основному блоку
3. Домашняя работа

1

ОСНОВНОЙ БЛОК

Введение

- Перечисляя то, что было
- Полиморфы атакуют!
- Лицом к лицу



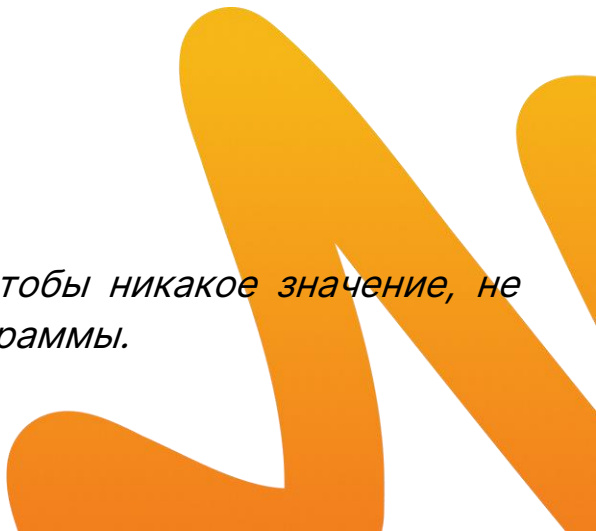
Проблема

В реальном мире существуют типы данных, которые отличаются от примитивных типов.

Например, существуют ограниченные наборы – такие, как

- 7 цветов радуги
- 7 нот в музыке
- 10 пальцев на руках
- 12 месяцев в году
- др.

Хотелось бы в программном коде иметь такие наборы. И чтобы никакое значение, не содержащееся в наборе, не могло помешать выполнению программы.



Перечисляя то, что было

Перечисление

Перечисление (enum) – это ссылочный тип данных (разновидность класса), который имеет ограниченный набор значений (ограниченный набор объектов), причём доступ к каждому значению можно получить через имя класса.

DayOfWeek.java

```
public enum DayOfWeek {  
    MONDAY,  
    TUESDAY,  
    WEDNESDAY,  
    THURSDAY,  
    FRIDAY,  
    SATURDAY,  
    SUNDAY  
}
```

Использование перечисления DayOfWeek в классе Main (файл Main.java)

```
public class Main {  
    public static void main(String[] args) {  
        DayOfWeek today = DayOfWeek.THURSDAY;  
        DayOfWeek tomorrow = DayOfWeek.FRIDAY;  
        DayOfWeek weekend = DayOfWeek.SATURDAY;  
    }  
}
```


Перечисляя то, что было

Структура проекта

Для создания перечисления нужно в структуре проекта щёлкнуть правой кнопкой мыши (ПКМ) и выбрать New -> Java class -> выбрать Enum и ввести имя перечисления.

Хорошей практикой структурирования проекта считается создание **пакетов (package)** – подпапок проекта. Это позволяет хранить связанные классы вместе и создаёт уникальные пути для одноимённых классов. Пакеты можно вкладывать друг в друга.

Чтобы создать пакет, щёлкните ПКМ на каталоге src или любом вложенном в него пакете -> New -> Package -> введите имя пакета.

Если класс (перечисление) лежит в пакете, то в месте его использования (клиентский код) нужно делать импорт класса с помощью оператора import в начале файла.

Перечисляя то, что было

Структура проекта

Если класс (перечисление) лежит в пакете, то в начале его файла указывается полное имя пакета после оператора *package*. А в месте его использования (клиентский код) нужно делать импорт класса с помощью оператора *import* в начале файла.

```
package person;  
  
public enum Gender {  
    MALE,  
    FEMALE,  
    OTHER,  
    UNKNOWN  
}
```

```
import person.Gender;  
  
public class Main {  
    public static void main(String[] args) {  
        Gender person1Gender = Gender.FEMALE;  
        Gender person2Gender = Gender.MALE;  
        Gender person3Gender = Gender.UNKNOWN;  
    }  
}
```

Перечисляя то, что было

Именованние перечислений

Перечисления (как и любые другие классы) именуются

- существительными в единственном числе, отражающими суть данных в перечислении
- в стиле *PascalCase*, т. е. с большой буквы и каждое новое слово «приклеивается» к предыдущему и начинается с большой буквы

Элементы перечисления именуются по правилу UPPER_SNAKE_CASE, т. е. все буквы заглавные, а слова разделяются знаком подчёркивания.

Пакеты именуются с маленькой буквы, разделение слов через точку

```
package person;  
  
public enum Gender {  
    MALE,  
    FEMALE,  
    OTHER,  
    UNKNOWN  
}
```

Перечисляя то, что было

Важные методы перечислений

name() – метод возвращает имя элемента перечисления;

ordinal() – метод возвращает порядковый номер элемента перечисления (начинается с 0);

toString() – превращает элемент перечисления в строку;

valueOf() – превращает строку, переданную в метод, в элемент перечисления. Если такого элемента нет, то бросает исключение;

values() – возвращает массив, в котором записаны все элементы перечисления.

Откуда взялись эти методы, если мы их не создаём?

Перечисляя то, что было

Важные методы перечислений

name() – метод возвращает имя элемента перечисления;

ordinal() – метод возвращает порядковый номер элемента перечисления (начинается с 0);

toString() – превращает элемент перечисления в строку;

valueOf() – превращает строку, переданную в метод, в элемент перечисления. Если такого элемента нет, то бросает исключение;

values() – возвращает массив, в котором записаны все элементы перечисления.

Откуда взялись эти методы, если мы их не создаём?

Каждое новое перечисление неявно наследуется от класса Enum, в котором эти методы определены.

Перечисляя то, что было

Задание



Напишите программу, которая по стилю музыки, указанному пользователем, выводит описание этого стиля.

- 1 В новом проекте создайте пакет `music`.
- 2 В пакете `music` создайте перечисление `Style`. В перечислении укажите столько названий стилей музыки, сколько сможете вспомнить.
- 3 В методе `main` реализуйте ввод пользователя.
- 4 С помощью `switch` получите описание стиля и выведите его в консоль.

Перечисляя то, что было

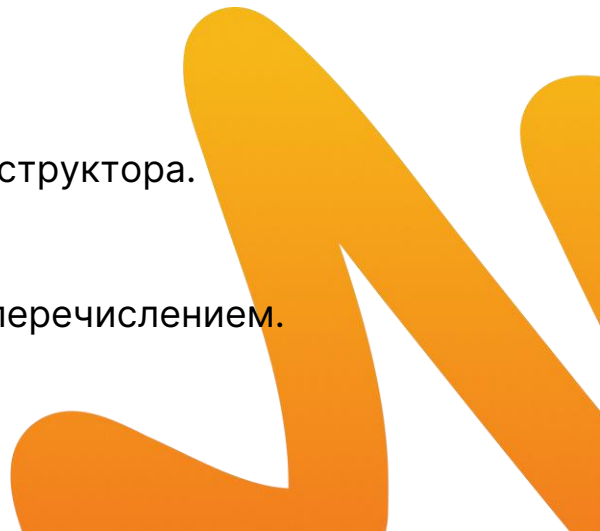
Перечисление – это класс

Перечисление является классом, в котором каждый элемент – это объект, хранящийся в статическом поле.

Мы можем создать поля для перечисления, чтобы хранить внутри элемента дополнительную информацию.

- Создать приватные поля с модификатором `final`.
- Создать конструктор с указанными полями.
- Создание элемента перечисления становится вызовом конструктора.
- Создать геттеры для новых полей.

Мы можем добавить и другие полезные методы для работы с перечислением.



Перечисляя то, что было

Задание

Допишите программу по выбору стиля музыки:

Перенесите информацию о стиле в поле `description`.

Добавьте поле `rusName`, в котором будет храниться название стиля на русском языке.

Создайте метод, который по названию стиля на русском языке будет возвращать элемент перечисления `Style`.

В методе `main` по названию стиля на русском языке, введённом пользователем, получите элемент перечисления.

Проблема

Представьте, что Вы работаете руководителем собственной IT-компании. У Вас в подчинении есть системные аналитики, которые формализуют требования к продукту, есть разработчики, которые на основании требований пишут код, и есть тестировщики, которые отлавливают баги. Чтобы сотрудники работали, Вам нужно каждое утро подходить к ним и вежливо напоминать, что пора заняться работой. В реальной жизни достаточно сказать «*Марш работать!*», но в программе у каждого класса нужно вызывать свой метод: у аналитика – `getTask()`, у разработчика `writeCode()`, у тестировщика – `findBug()`.

Вот бы нам такой способ, чтобы всем классам можно было говорить «Марш работать!» вне зависимости от того, какому классу мы это говорим.

И ещё раз

наследование

инкапсуляция

полиморфизм

Абстракция

~~несогласиями~~



Полиморфы атакуют!

Полиморфизм

Полиморфизм (много форм) – способность объекта «становиться» одним из своих предков в программе. Полиморфизм проявляется как:

1 Любой экземпляр класса-потомка помещается в переменную типа класса-предка или реализуемого интерфейса. (Но не наоборот!)

При этом потомок для программы становится экземпляром класса предка, т. е. скрывает поля и методы класса-потомка (проявление сокрытия).

2 Если у объекта класса-потомка, помещённого в переменную класса-предка, вызвать переопределённый метод, то будет выполнен метод класса-потомка (JVM определит его автоматически)



Полиморфы атакуют!

Где встречается полиморфизм

1. Когда используем универсальные методы, как `System.out.println()` или `String.valueOf()` мы можем передать в метод любой тип данных в качестве параметра.
Но как это возможно, если Java – строго типизированный язык программирования?
2. Когда мы переопределяем методы класса `Object`.
Предположим, мы сложили в массив `Object[]` объекты разных классов и в цикле решили вывести элементы массива.
Какой метод `toString()` будет вызван у элемента – тот, который определён в `Object` (`className + @ + hashCode`) или переопределённый метод конечного класса?



Полиморфы атакуют!

Пример полиморфизма



polymorphism.zip



Полиморфы атакуют!

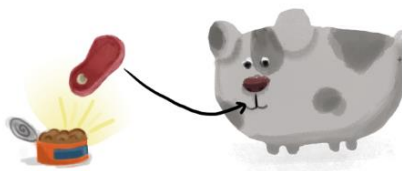
Пример полиморфизма

Incapsulation



every animal eats
and then poop

Polymorphism



each animal can eat
its own type of food

Inheritance



you can create new type of animal
changing or adding properties

Задание 1

- 1 Создайте абстрактный класс *Animal*, определите в нём абстрактный метод *makeSound()*.
- 2 Создайте подклассы *Animal*. Например, *Cat*, *Dog*, *Bird*. Переопределите (реализуйте) метод *makeSound()* в каждом классе так, чтобы метод возвращал звук, издаваемый этим животным.
- 3 Создайте массив объектов *Animal* и в цикле вызовите метод *makeSound()* для каждого объекта, используя полиморфизм.

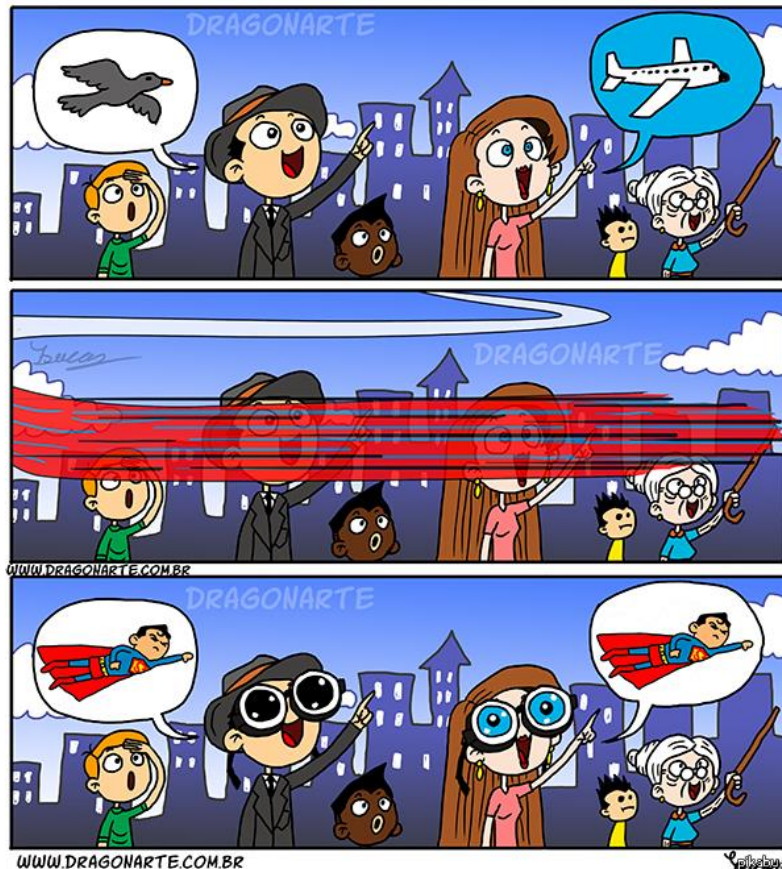
Задание 2

- 1 Создайте абстрактный класс *Employee*, определите в нём абстрактный метод *calculateSalary()*.
- 2 Создайте подклассы *Employee*. Например, *Manager* и *Engineer*, которые переопределяют метод *calculateSalary()*. Переопределите (реализуйте) метод *calculateSalary()* в каждом классе так, чтобы метод возвращал размер заработной платы сотрудника.
- 3 Создайте класс *Company*, который хранит массив/список *Employee*. Создайте метод расчёта количества денег, которое потребуется для зарплаты в текущий месяц.

Проблема

Очень часто мы объединяем разнородные объекты реального мира в группы по определённому признаку. Например, птицы летают. В программе разные виды птиц (орёл, соловей, снегирь и др.) будут наследниками одного и того же класса *Птица*. Но ведь летать может также самолёт и бумажный воздушный змей, и даже Супермен. Они явно не будут являться наследниками класса *Птица* (нет клюва и перьев, не чирикают). Самолёт будет наследником класса *Транспорт*. А Супермен – класса *Герой*.

Можем ли мы работать с такими объектами в единой коллекции, чтобы поднять их в воздух?



Лицом к лицу

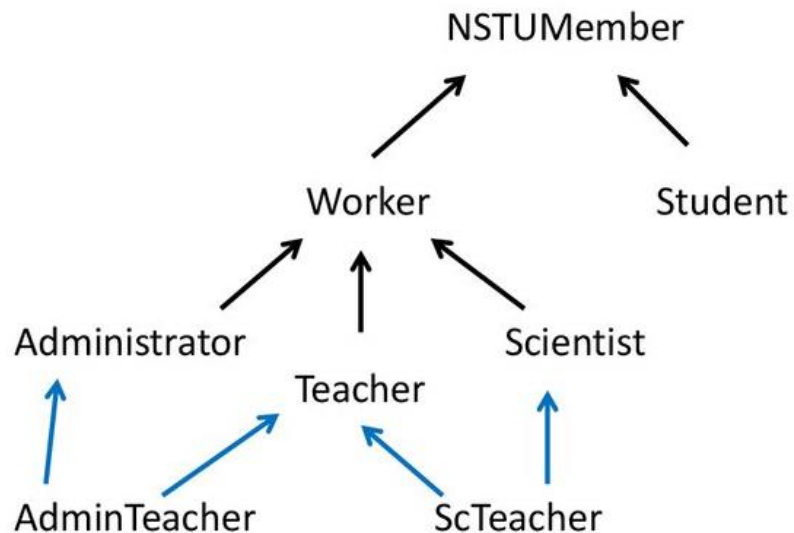
Как было раньше

В **C++** было доступно множественное наследование, т. е. один потомок расширял одновременно несколько классов, но это часто приводило к проблемам при написании конструкторов и последующем использовании объектов класса-наследника.

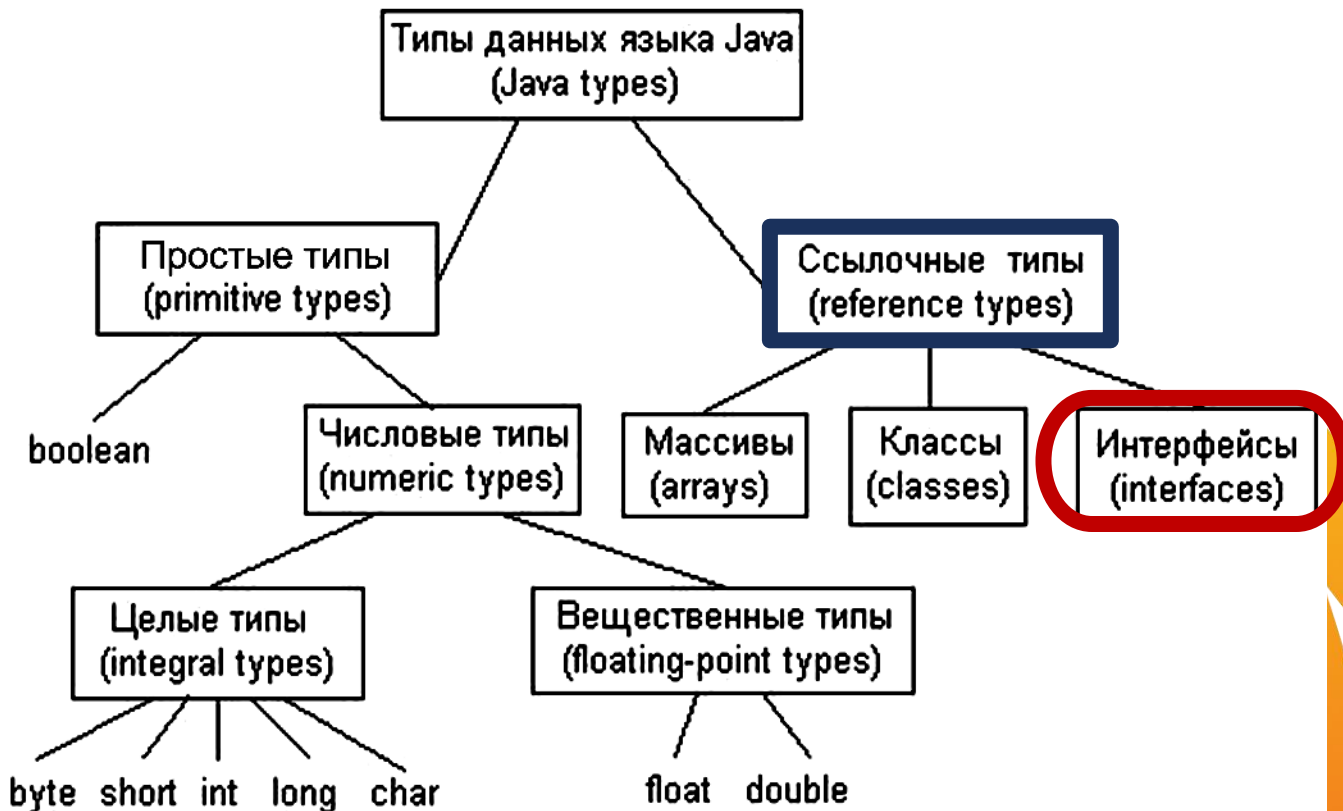
Все классы **Java** могут расширять только **один** класс-предок.



Пример множественного наследования
C++



Лицом к лицу Интерфейсы



Лицом к лицу Интерфейсы

«Интерфейс» – это перегруженный термин, потому что так называют

- публичную часть объекта (публичные поля, методы и бросаемые исключения)
- конструкцию *interface* в Java.

В данной презентации речь идёт о механизме интерфейсов, которые объявляют через ключевое слово *interface*. Такие конструкции являются «контрактами» взаимодействия объектов.

По своей сути ***interface*** – это набор публичных методов, которые должны реализовать классы, которые эти интерфейсы реализуют (impleментируют).

Реализация – это такой тип взаимоотношения классов, когда интерфейс или абстрактный класс не содержит тела метода. Метод должен быть реализован (написано тело) в подклассах.

```
package flytransport;

import java.util.List;

public interface FlyingTransport {
    void fly(String origin, String destination, List<String> passengers);
}
```

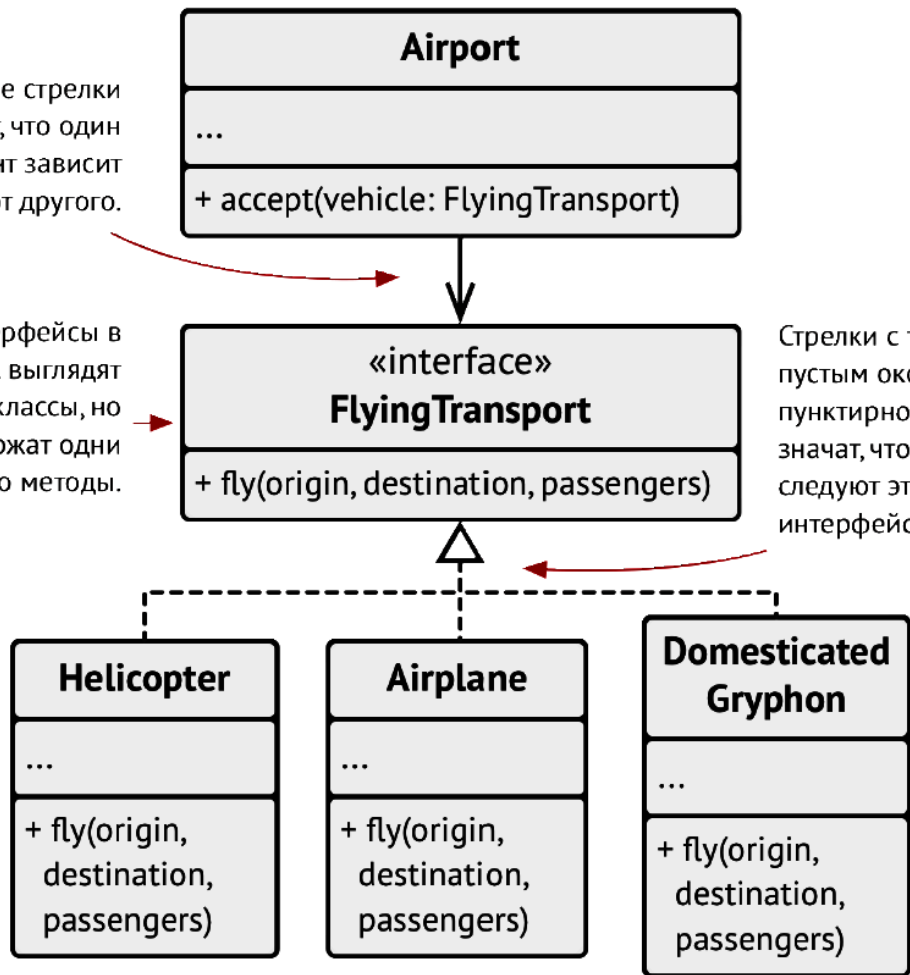
Лицом к лицу Интерфейсы

Вы создали интерфейс *ЛетающийТранспорт* с методом *лететь* (откуда, куда, пассажиры), а затем описали методы класса *Аэропорт* так, чтобы они принимали любые объекты с этим интерфейсом. Теперь вы можете быть уверены, что любой объект, реализующий интерфейс – будь то *Самолёт*, *Вертолёт* или *ДрессированныйГрифон* – сможет работать с *Аэропортом*.

Простые стрелки значат, что один элемент зависит от другого.

Интерфейсы в UML выглядят как классы, но содержат одни только методы.

Стрелки с треугольным пустым окончанием, но пунктирной линией значат, что классы следуют этому интерфейсу.



UML-диаграмма реализации и использования интерфейса.

Лицом к лицу Интерфейсы



interfaceExample.zip



Лицом к лицу

Свойства интерфейсов

1 Все методы интерфейса являются публичными.

2 Интерфейсы могут наследовать друг друга с помощью ключевого слова *extends*. Это помогает избежать «жирных» интерфейсов. Т.к. все методы интерфейса должны быть реализованы, то даже ненужные в классе методы приходится имплементировать, либо помечать класс как *abstract*.

3 Класс может расширять только одного предка, но реализовывать множество интерфейсов (*class MyClass extends ParentClass implements Interface1, Interface2, Serializable*).

Лицом к лицу

Свойства интерфейсов

4 В JDK 8 были добавлены

- методы по умолчанию. Содержат в «шапке» модификатор *default*. Интерфейсы кроме определения методов могут иметь их реализацию по умолчанию, которая используется, если класс, реализующий данный интерфейс, не реализует метод;
- статические методы (аналогичны статическим методам классов). Статические методы в интерфейсах не наследуются.

5 Начиная с Java 9 в интерфейсе можно определить статические и нестатические методы с модификатором *private*. Private-методы не могут иметь реализации по умолчанию и используются только внутри самого интерфейса.

Лицом к лицу

Свойства интерфейсов



6 В интерфейсах могут быть определены статические константы. Константы не имеют модификаторов, но по умолчанию они имеют модификатор доступа *public static final*, и поэтому их значение доступно из любого места программы.

7 Как и классы, интерфейсы могут быть вложенными, то есть могут быть определены в классах или других интерфейсах.

8 Как и у абстрактных классов, невозможно напрямую создать экземпляр интерфейса, а только поместить одну из имплементаций в переменную типа интерфейса.

9 Интерфейсы без методов используют как маркеры, чтобы пометить класс для JVM для какой-либо цели. Например, интерфейсом *Serializable* помечают классы, объекты которых нам нужно сериализовать.

Лицом к лицу

Свойства интерфейсов



10 Внутри интерфейсов можно описать перечисления и аннотации.

11 Если у двух интерфейсов одинаковые методы с одинаковым возвращаемым типом и сигнатурой, то такой метод достаточно реализовать в классе один раз. Если отличаются только возвращаемые типы, то реализовать получится только один из методов.



Задание

Определить абстрактный класс *Publication* с полем *name* и *text*. Создать два подкласса – *Book* (добавить поле *ISBN*) и *Magazine* (добавить поле *Дата выхода*). В классе определить метод *getInformation()*, возвращающий строку-карточку издания (все характеристики, кроме текста).

Отдельно от иерархии создайте класс *Paper*, который хранит текст.

Определить интерфейс *Printable*, содержащий метод *void print()*. Классы *Book*, *Magazine* и *Paper* должны реализовать интерфейс *Printable*.

Создать массив типа *Printable*, который будет содержать разные бумаги, книги и журналы.

В цикле пройти по массиву и вызвать метод *print()* для каждого объекта.

Создать статический метод *printBooks(Printable[] printable)* в классе *Book*, который выводит на консоль названия только книг. Используем оператор *instanceof* для проверки типа.

Домашнее задание

№1

Создайте программу, использующую перечисление с полями. По введённому пользователем названию месяца (на английском языке) программа выводит среднюю температуру месяца и к какому сезону года месяц относится (зима, весна, лето, осень).



Домашнее задание

№2. Необходимо написать консольное приложение с объектно-ориентированной архитектурой, выводящее характеристики заданной пользователем геометрической фигуры. Пользователь вводит имя фигуры и её параметры одной строкой. Результатом работы приложения являются вычисленные характеристики. Парсинг введённой пользователем строки выполните в отдельном утилитарном классе.

Фигура	Входные параметры	Вычисляемые параметры
Круг	Радиус	Название, Площадь, Периметр, Радиус, Диаметр
Прямоугольник	Длины сторон (два значения)	Название, Площадь, Периметр, Длина диагонали, Длина (размер длинной стороны), Ширина (размер короткой стороны)
Треугольник	Длины сторон (три значения)	Название, Площадь, Периметр, Длина стороны и противолежащий угол для каждой из трех сторон
Своя фигура (придумайте свой вариант)	Параметры Вашей геометрической фигуры	Название, Площадь, Периметр, и другие параметры Вашей геометрической фигуры

Домашнее задание

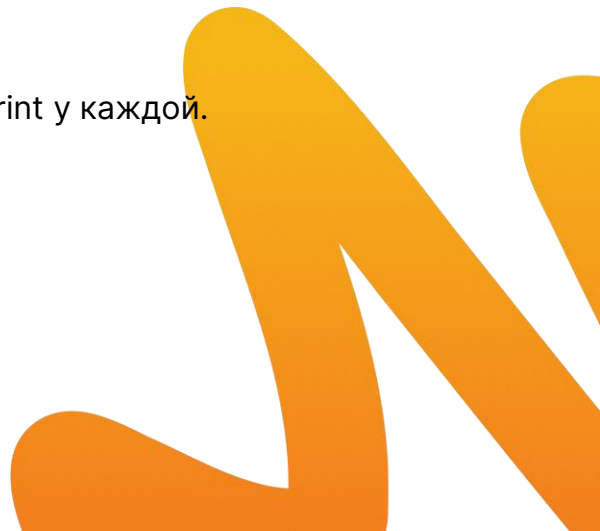
№3

Доделайте задание №2. Создайте интерфейс `IPrintable`, который будет содержать метод `print()`.

Каждая фигура должна наследовать этот интерфейс. Реализацию метода можно сделать в виде вывода фигуры в консоль символами `□`, `⊙`, `△`, либо текстом или графически (нарисовать символами).

При выводе данных о фигуре также вызовите метод `print`.

Создайте массив `IPrintable` из нескольких фигур и в цикле вызовите метод `print` у каждой.



ЗАКЛЮЧЕНИЕ

