**Интерфейсы**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 2](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=2), Лекция 7

— Привет, Амиго! Сегодня у тебя день открытий. Новая и интересная тема – это интерфейсы.

— Ага. День настолько чудесный, что я приду домой и приму ванну полную воды.

— **Интерфейс** – это дитя Абстракции и Полиморфизма. Интерфейс очень напоминает абстрактный класс, у которого все методы абстрактные. Он объявляется так же, как и класс, только используется ключевое слово **interface**. Примеры:

|  |  |
| --- | --- |
| **Код** | **Описание и Факты** |
| interface Drawable  {  void draw();  }  interface HasValue  {  int getValue();  } | 1) Вместо слова class пишем interface.  2) Содержит только абстрактные методы (слово abstract писать не нужно).  3) На самом деле у интерфейсов все методы — public. |
| interface Element extends Drawable, HasValue  {  int getX();  int getY();  } | Интерфейс может наследоваться только от интерфейсов.  Интерфейсов-родителей может быть много. |
| abstract class ChessItem implements Drawable, HasValue  {  private int x, y, value;  public int getValue()  {  return value;  }  public int getX()  {  return x;  }  public int getY()  {  return y;  }  } | Класс может наследоваться от нескольких интерфейсов (и только от одного класса). При этом используется ключевое слово implements.  Класс ChessItem объявлен абстрактным: он реализовал все унаследованные методы, кроме draw.  Т.е. класс ChessItem содержит один абстрактный метод: draw(). |

— Интересно. А зачем нужны интерфейсы? Когда их используют?

— У интерфейсов есть два сильных преимущества по сравнению с классами:

**1) Отделение «описания методов» от их реализации.**

Раньше я тебе рассказывал, что если ты хочешь разрешить вызывать методы своего класса из других классов, то их нужно пометить ключевым словом public. Если же хочешь, чтобы какие-то методы можно было вызывать только из твоего же класса, их нужно помечать ключевым словом private. Другими словами мы делим методы класса на две категории: «для всех» и «только для своих».

С помощью интерфейсов, это деление можно усилить еще больше. Мы сделаем специальный «класс для всех», и второй «класс для своих», который унаследуем от первого. Вот как это примерно будет:

Было

class Student

{

private String name;

public Student(String name)

{

this.name = name;

}

public String getName()

{

return this.name;

}

private void setName(String name)

{

this.name = name;

}

Стало

interface Student

{

public String getName();

}

class StudentImpl implements Student

{

private String name;

public StudentImpl(String name)

{

this.name = name;

}

public String getName()

{

return this.name;

}

private void setName(String name)

{

this.name = name;

}

}

Было

public static void main(String[] args)

{

Student student =

new Student("Alibaba");

System.out.println(student.getName());

}

Стало

public static void main(String[] args)

{

Student student =

new StudentImpl("Ali");

System.out.println(student.getName());

}

Мы разбили наш класс на два: интерфейс и класс, унаследованный от интерфейса.

— И в чем тут преимущество?

— Один и тот же интерфейс могут реализовывать (наследовать) различные классы. И у каждого может быть свое собственное поведение. Так же как **ArrayList** и **LinkedList** – это две различные реализации интерфейса List.

Таким образом, мы скрываем не только различные реализации, но и даже сам класс, который ее содержит (везде в коде может фигурировать только интерфейс). Это позволяет очень гибко, прямо в процессе исполнения программы, подменять одни объекты на другие, меняя поведение объекта скрытно от всех классов, которые его используют.

Это очень мощная технология в сочетании с полиморфизмом. Сейчас далеко не очевидно, зачем так нужно делать. Ты должен сначала столкнуться с программами, состоящими из десятков или сотен классов, чтобы понять, что интерфейсы способны существенно упростить тебе жизнь.

**2) Множественное наследование.**

В Java все классы могут иметь только одного класса-родителя. В других языках программирования, классы часто могут иметь несколько классов-родителей. Это очень удобно, но приносит так же много проблем.

В Java пришли к компромиссу – запретили множественное наследование классов, но разрешили множественное наследование интерфейсов. Интерфейс может иметь несколько интерфейсов-родителей. Класс может иметь несколько интерфейсов-родителей и только один класс-родитель.