**Причины существования интерфейсов — поддержка заявленного поведения**

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

[Уровень 3](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=3), Лекция 1

— Привет, Амиго! Хочу сегодня тебе рассказать о причинах существования интерфейсов. Тебе очень часто придется слышать, что такой-то класс, объект или сущность поддерживает определенный интерфейс. Что же это значит – **поддерживать интерфейс**?

[](https://cdn.javarush.ru/images/article/6c1c13b8-2a0d-457d-be06-09dbc5e92910/original.jpeg)

В более широком смысле интерфейс какой-нибудь вещи – это механизм взаимодействия этой вещи с другими предметами. Например, пульт от телевизора – это дистанционный интерфейс. Собака понимает и исполняет команды — это значит, что собака поддерживает голосовой интерфейс (управления). Если все это подытожить, то можно сказать, что интерфейс – это стандартизированный способ взаимодействия двух вещей, и этот стандарт известен двум сторонам. Когда человек говорит собаке «сидеть», он отдает команду в соответствии с «голосовым интерфейсом управления собакой», и если собака выполняет эту команду, то мы говорим, что собака поддерживает этот интерфейс.

Так же и в программировании. Методы – это действия над объектом, над его данными. И если класс реализует определенные методы, то он «поддерживает исполнение» определенных команд. Какие же преимущества дает объединение методов в интерфейс?

**1)** Каждый **interface**, как и **class**, имеет уникальное имя. Обе стороны могут быть на 100% уверены, что вторая сторона поддерживает именно нужный (известный им) интерфейс, а не похожий.

**2)** Каждый интерфейс налагает определенные ограничения на тот класс, который собирается поддерживать его. Класс сам решает (его разработчик), что он будет делать в случае вызова его методов, которые он унаследовал от интерфейса, но результат должен находиться в пределах ожиданий. Если мы скомандовали собаке «сидеть», и она покрутилась 5 минут на месте и села, то это – поддержка интерфейса. А если она вместо этого вцепилась вам в ногу, то ни о какой поддержке тут не может быть и речи. Выполнение команды не привело к ожидаемым результатам.

Допустим, ты с друзьями участвуешь в написании компьютерной игры. И тебе досталась работа запрограммировать поведение одного персонажа. Один ваш коллега уже написал код по отображению всех персонажей на экран. Второй, отвечающий за сохранение игры на диск, написал код по сохранению всех объектов игры в файл. Каждый из них написал много кода и сделал интерфейс для взаимодействия с ним. Например, это может выглядеть так:

|  |  |
| --- | --- |
| **Код на Java** | **Описание** |
| interface Saveable  {  void saveToMap(Map<String, Object> map);  void loadFromMap(Map<String, Object> map);  } | — интерфейс по сохранению/загрузке объекта из map’а. |
| interface Drawable  {  void draw(Screen screen);  } | — интерфейс по отрисовке объекта внутри переданного объекта screen. |
| class PacMan implements Saveable, Drawable  {  …  } | — твой класс, реализующий поддержку двух интерфейсов. |

Другими словами, чтобы поддержать реализацию какого-то интерфейса (группы интерфейсов) в своем классе нужно:

**1)** Унаследоваться от них

**2)** Реализовать объявленные в них методы

**3)** Методы должны делать то, для чего они предназначены.

Тогда остальной код программы, который ничего не знает о твоем классе и его объектах, сможет успешно работать с ним.

— А почему код может ничего не знать о моем классе?

— Допустим, ты взял код программы, который кто-то написал год назад. Или твои друзья купили/лицензировали движок игры у кого-то еще. Есть рабочий код игры. Тысячи объектов, которые взаимодействуют друг с другом. И они могут с легкостью **правильно взаимодействовать с твоими объектами**, если взаимодействие организовано через интерфейсы, и ты правильно реализовал эти интерфейсы в своих классах.

— Круто! Не знал что так можно.

— На этом принципе основаны все большие проекты. Уже давно никто ничего не пишет с нуля.

Люди тоже не изобретают математику и алфавит каждый раз заново, а изучают все то, что было придумано до них.