**Множественное наследование интерфейсов**

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

[Уровень 3](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=3), Лекция 5

— Привет, Амиго! Наконец-то мы добрались до очень интересной темы. Сегодня я расскажу тебе про **множественное наследование**. На самом деле множественное наследование очень интересный и мощный инструмент. И если бы не некоторые проблемы, то в Java было бы множественное наследование классов. Но т.к. его нет, придется довольствоваться **множественным наследованием интерфейсов**. Что тоже не мало.

[](https://cdn.javarush.ru/images/article/db1a2ff7-692c-45b5-8588-0bc2d60f20ea/original.jpeg)

Представь, что ты пишешь компьютерную игру. И ее герои – твои объекты – должны демонстрировать очень сложное поведение: ходить по карте, собирать предметы, выполнять квесты, общаться с другими героями, кого-то убивать, кого-то спасать. Допустим, ты смог разделить все объекты на 20 категорий. Это значит, что если тебе повезет, ты можешь обойтись всего 20-ю классами, для их описания. А теперь вопрос на засыпку: сколько всего уникальных видов взаимодействия у этих объектов. Объект каждого типа может иметь уникальные взаимодействия с 20-ю видами других объектов (себе подобных тоже считаем). Т.е. всего нужно запрограммировать 20 на 20 – 400 взаимодействий! А если уникальных видов объектов будет не 20, а 100, количество взаимодействий может достигнуть 10,000!

— Ничего себе! Теперь понимаю, почему программирование такая непростая работа.

— Она простая. Благодаря многим абстракциям. И в не последнюю очередь – множественному наследованию интерфейсов.

Очень часто можно упростить взаимодействие объектов, если взаимодействовать будут не объекты, а их роли и/или способности. А способности, как мы уже знаем, легко добавляются в класс, когда он реализует некоторый интерфейс.

Когда пишется большая программа, обычно с этого сразу и начинают:

**1)** Определяют все существующие способности/роли.

**2)** Затем описывают взаимодействие между этими ролями.

**3)** А потом просто наделяют все классы их ролями.

— А можно пример?

— Конечно. Давай рассмотрим роли, на основе героев мультика «Том и Джерри».

|  |  |
| --- | --- |
| **Код на Java** | **Описание** |
| interface Moveable  {} | — роль/способность передвигаться. |
| interface Eatable  {} | — роль/способность быть съеденным. |
| interface Eat  {} | — роль/способность съесть кого-нибудь. |
| class Tom extends Cat implements Moveable, Eatable, Eat  {} | Tom – это кот, у которого есть три роли: 1) может передвигаться 2) может кого-то съесть 3) может быть съеденным кем-то (собакой) |
| class Jerry extends Mouse implements Moveable, Eatable  {} | Jerry – это мышь, у которого есть две роли: 1) может передвигаться 2) может быть съеденным кем-то |
| class Killer extends Dog implements Moveable, Eat  {} | Killer – это собака, у которого есть две роли: 1) может передвигаться 2) может кого-то съесть |

Зная всего эти три роли (интерфейса) можно написать программу и описать корректное взаимодействие этих ролей. Например, объект будет гнаться (посредством интерфейса Moveable) за тем, «кого ты можешь съесть» и убегать от того, «кто может съесть тебя». И все это без знаний о конкретных объектах. Если в программу добавить еще объектов (классов), но оставить эти роли, она будет прекрасно работать – управлять поведением своих объектов.