**Отличия класса Abstract от Interface**

Абстрактные классы и интерфейсы в Java

*В каких случаях стоит использовать абстрактный класс, а в каких — интерфейс? Давайте разбираться, в чем между ними разница.*

Абстрактные классы и интерфейсы встречаются повсюду как в Java-приложениях, так и в самом [Java Development Kit](https://www.infoworld.com/article/3296360/what-is-the-jdk-introduction-to-the-java-development-kit.html) (JDK). Каждый из них служит своей цели:

* Интерфейс — это контракт, который должен быть реализован конкретным классом.
* Абстрактный класс похож на обычный, но отличается тем, что может содержать абстрактные методы — методы без реализации, и нельзя создать экземпляр абстрактного класса.

Многие разработчики не видят разницы между интерфейсами и абстрактными классами, но на самом деле между ними есть весьма существенное различие.

Интерфейсы

Интерфейс — это контракт, который реализуется в некотором классе. У интерфейса не может быть состояния, поэтому в нем нельзя использовать изменяемые поля экземпляра. В интерфейсе могут быть только неизменяемые final-поля.

Когда использовать интерфейсы

Интерфейсы очень полезны для [уменьшения связанности (coupling) кода и реализации полиморфизма](https://www.infoworld.com/article/3290403/java-challengers-3-polymorphism-and-inheritance.html). Для примера давайте взглянем на интерфейс List из JDK:

**public** **interface** **List**<**E**> **extends** **Collection**<**E**> {

**int** **size**();

**boolean** **isEmpty**();

**boolean** **add**(E e);

E **remove**(**int** index);

**void** **clear**();

}

Как вы, вероятно, заметили, код весьма краток и лаконичен. Здесь мы видим сигнатуры методов, которые будут реализованы в конкретном классе, реализующем этот интерфейс.

Контракт интерфейса List реализуется классами ArrayList, Vector, LinkedList и другими.

При использовании полиморфизма тип переменной объявляем как List, и присваиваем ей любую из доступных реализаций. Например:

List list = **new** ArrayList();

System.out.println(list.getClass());

List list = **new** LinkedList();

System.out.println(list.getClass());

Результат:

**class** **java**.**util**.**ArrayList**

**class** **java**.**util**.**LinkedList**

В этом случае в каждом классе присутствует своя реализация методов. И это отличный пример использования интерфейсов. Если вы заметили, что ряд ваших классов содержит одинаковые методы, но с разными реализациями, то стоит использовать интерфейс.

Переопределение метода интерфейса

Помните, что интерфейс — это контракт, который должен быть реализован конкретным классом. Методы интерфейса неявно абстрактны и обязаны быть реализованы в классе, реализующем этот интерфейс.

Рассмотрим следующий пример:

**public** **class** **OverridingDemo** {

**public** **static** **void** **main**(String[] args) {

Challenger challenger = **new** JavaChallenger();

challenger.doChallenge();

}

}

**interface** **Challenger** {

**void** **doChallenge**();

}

**class** **JavaChallenger** **implements** **Challenger** {

@Override

**public** **void** **doChallenge**() {

System.out.println("Challenge done!");

}

}

Результат будет следующий:

Challenge done!

Обратите внимание еще раз, что методы интерфейса неявно абстрактны и их не нужно явно объявлять как abstract.

Неизменяемые переменные

Еще одно правило, которое следует помнить, заключается в том, что интерфейс может содержать только неизменяемые переменные. Следующий код вполне рабочий:

**public** **interface** **Challenger** {

**int** number = 7;

String name = "Java Challenger";

}

Обратите внимание, что обе переменные неявно final и static. Это означает, что они являются константами, не зависят от экземпляра и не могут быть изменены.

При попытке изменить поля в интерфейсе Challenger, например, следующим образом:

Challenger.number = 8;

Challenger.name = "Another Challenger";

будет ошибка компиляции:

Cannot assign a value to final variable 'number'

Cannot assign a value to final variable 'name'

Default-методы

После появления в Java 8 методов по умолчанию, некоторые разработчики решили, что интерфейсы стали абстрактными классами. Однако это не так, поскольку у интерфейсов не может быть состояния.

У методов по умолчанию может быть реализация, а у абстрактных методов — нет. Методы по умолчанию — результат появления лямбда-выражений и Stream API, но использовать их нужно с осторожностью.

В качестве примера default-метода из JDK можно привести метод forEach() из интерфейса Iterable. Вместо копирования кода этого метода во все реализации Iterable, мы можем переиспользовать метод forEach:

**default** **void** **forEach**(Consumer<? **super** T> action) {

// Code implementation here…

Любая реализация Iterable может использовать метод forEach() без необходимости реализации этого нового метода.

Давайте рассмотрим пример с методом по умолчанию:

**public** **class** **DefaultMethodExample** {

**public** **static** **void** **main**(String[] args) {

Challenger challenger = **new** JavaChallenger();

challenger.doChallenge();

}

}

**class** **JavaChallenger** **implements** **Challenger** { }

**interface** **Challenger** {

**default** **void** **doChallenge**() {

System.out.println("Challenger doing a challenge!");

}

}

Результат:

Challenger doing a challenge!

Важно отметить, что у default-метода должна быть реализация и default-метод не может быть статическим.

Абстрактные классы

У абстрактных классов может быть состояние в виде изменяемых полей экземпляра. Например:

**public** **abstract** **class** **AbstractClassMutation** {

**private** String name = "challenger";

**public** **static** **void** **main**(String[] args) {

AbstractClassMutation abstractClassMutation = **new** AbstractClassImpl();

abstractClassMutation.name = "mutated challenger";

System.out.println(abstractClassMutation.name);

}

}

**class** **AbstractClassImpl** **extends** **AbstractClassMutation** { }

Результат:

mutated challenger

Абстрактные методы в абстрактных классах

Аналогично интерфейсам в абстрактных классах могут быть абстрактные методы. Абстрактный метод — это метод без тела (без реализации). Но в отличие от интерфейсов, абстрактные методы в абстрактных классах должны быть явно объявлены как абстрактные.

public abstract class AbstractMethods {

abstract void doSomething();

}

Попытка объявить метод без реализации и без ключевого слова abstract, например, следующим образом:

public abstract class AbstractMethods {

void doSomethingElse();

}

приведет к ошибке компиляции:

Missing method body, or declare abstract

Когда использовать абстрактные классы

Рекомендуется использовать абстрактный класс, когда вам нужно изменяемое состояние. В качестве примера можно привести класс [AbstractList](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractList.html) из Java Collections Framework, который использует состояние.

Если хранить состояние класса не нужно, обычно лучше использовать интерфейс.

Хороший пример использования абстрактных классов — паттерн ["шаблонный метод" (template method)](https://javachallengers.com/design-patterns-saga-8-real-project-situations-with-template-method/). Шаблонный метод манипулирует переменными экземпляра (полями) внутри конкретных методов.

Различия между абстрактными классами и интерфейсами

С точки зрения объектно-ориентированного программирования основное различие между интерфейсом и абстрактным классом заключается в том, что интерфейс не может иметь состояния, тогда как абстрактный класс может (в виде полей экземпляра).

Другое ключевое различие заключается в том, что классы могут реализовывать более одного интерфейса, но расширять только один абстрактный класс. Множественное наследование может привести к тупиковым ситуациям в коде, поэтому авторы Java решили этого избежать, отказавшись от него.

Еще одно различие состоит в том, что интерфейс может быть реализован классом или расширен другим интерфейсом, а класс может быть только расширен.

Также важно отметить, что лямбда-выражения могут использоваться только с функциональными интерфейсами (интерфейс только с одним методом), но не с абстрактными классами с одним абстрактным методом.

В таблице 1 обобщены различия между абстрактными классами и интерфейсами.

**Таблица 1. Сравнение интерфейсов и абстрактных классов**

|  |  |
| --- | --- |
| **Интерфейсы** | **Абстрактные классы** |
| Могут содержать только final static поля. Интерфейс никогда не может изменять свое состояние. | Могут быть любые поля, в том числе статические, изменяемые и неизменяемые. |
| Класс может реализовывать несколько интерфейсов. | Класс может расширять только один абстрактный класс. |
| Может быть реализован с помощью ключевого слова implements.  Может расширять другой интерфейс с помощью extends. | Может быть только расширен с помощью extends. |
| Можно использовать только static final поля. Параметры и локальные переменные в методах. | Могут быть изменяемые поля экземпляра. Параметры и локальные переменные в методах. |
| В лямбда-выражениях могут использоваться только функциональные интерфейсы. | Абстрактные классы с одним абстрактным методом не могут использоваться в лямбда-выражениях. |
| Не может быть конструктора. | Может содержать конструктор. |
| Могут быть абстрактные методы.  Могут быть default и static методы (c Java 8).  Могут быть private методы с реализацией (с Java 9). | Могут быть любые методы. |

Задачка

Давайте изучим основные различия между интерфейсами и абстрактными классами с помощью небольшой задачки. Вы также можете посмотреть данный материал [в формате видео (англ.)](https://youtu.be/yJ_lgsrTw0U).

В приведенном ниже коде объявлены интерфейс, абстрактный класс и используются лямбда-выражения.

**public** **class** **AbstractResidentEvilInterfaceChallenge** {

**static** **int** nemesisRaids = 0;

**public** **static** **void** **main**(String[] args) {

Zombie zombie = () -> System.out.println("Graw!!! " + nemesisRaids++);

System.out.println("Nemesis raids: " + nemesisRaids);

Nemesis nemesis = **new** Nemesis() { **public** **void** **shoot**() { shoots = 23; }};

Zombie.zombie.shoot();

zombie.shoot();

nemesis.shoot();

System.out.println("Nemesis shoots: " + nemesis.shoots +

" and raids: " + nemesisRaids);

}

}

**interface** **Zombie** {

Zombie zombie = () -> System.out.println("Stars!!!");

**void** **shoot**();

}

**abstract** **class** **Nemesis** **implements** **Zombie** {

**public** **int** shoots = 5;

}

Как вы думаете, какой будет вывод, когда мы запустим этот код? Выберите один из следующих вариантов:

Вариант 1

Compilation error at line 4

Вариант 2

Graw!!! 0

Nemesis raids: 23

Stars!!!

Nemesis shoots: 23 and raids:1

Вариант 3

Nemesis raids: 0

Stars!!!

Graw!!! 0

Nemesis shoots: 23 and raids: 1

Вариант 4

Nemesis raids: 0

Stars!!!

Graw!!! 1

Nemesis shoots: 23 and raids:1

Вариант 5

Compilation error at line 6

Разбор задачи

Эта задачка демонстрирует понятия об интерфейсах, абстрактных методах и о некоторых других вещах. Давайте разберем код строка за строкой.

В первой строке main() присутствует лямбда-выражение для интерфейса Zombie. Обратите внимание, что в этой лямбде мы инкрементируем статическое поле. Здесь также можно было использовать поле экземпляра, но не локальную переменную, объявленную вне лямбда-выражения. То есть код компилируется без ошибок. Также обратите внимание, что это лямбда-выражение еще не выполняется, оно только объявлено, и поле nemesisRaids не будет увеличено.

Далее мы выводим значение поля nemesisRaids, которое еще не увеличено. Следовательно, вывод будет:

Nemesis raids: 0

Еще один интересный момент заключается в том, что мы используем анонимный внутренний класс. Мы создаем не экземпляр абстрактного класса Nemesis, но экземпляр анонимного класса, расширяющего Nemesis. Также обратите внимание, что первый конкретный класс в иерархии наследования всегда будет обязан реализовать абстрактные методы.

В интерфейсе Zombie есть поле с типом интерфейса Zombie, объявленное с помощью лямбда-выражения. Поэтому, когда мы вызываем метод Zombie.zombie.shoot(), получим следующий вывод:

Stars!!!

В следующей строке вызывается лямбда-выражение, которое мы создали в начале. Следовательно, переменная nemesisRaids будет увеличена. Однако, поскольку мы используем оператор постинкремента, она будет увеличена только после этого выражения. Следующий вывод будет:

Graw!!! 0

Далее вызовем метод shoot для nemesis, который изменяет поле экземпляра shoots на 23. Обратите внимание, что как раз здесь мы видим основную разницу между интерфейсом и абстрактным классом.

Наконец, мы выводим значение nemesis.shoots и nemesisRaids.

Nemesis shoots: 23 and raids: 1

Правильный ответ — вариант 3:

Nemesis raids: 0

Stars!!!

Graw!!! 0

Nemesis shoots: 23 and raids: 1