Java

Абстрактные классы и интерфейсы

- Абстрактный класс — это максимально абстрактная, очень приблизительная «заготовка» для группы будущих классов. Эту заготовку нельзя использовать в готовом виде — слишком «сырая». Но она описывает некое общее состояние и поведение, которым будут обладать будущие классы — наследники абстрактного класса.

Зачем нужны абстрактные классы?

- Допустим, мы делаем программу для обсулживания банковских операций и определяем в ней три класса: Person, который описывает человека, Employee, который описывает банковского служащего, и класс Client, который представляет клиента банка. Очевидно, что классы Employee и Client будут производными от класса Person, так как оба класса имеют некоторые общие поля и методы. И так как все объекты будут представлять либо сотрудника, либо клиента банка, то напрямую мы от класса Person создавать объекты не будем. Поэтому имеет смысл сделать его абстрактным.

Вопросы

1. Могут ли в языке Java у абстрактного класса быть конструкторы?

Да, в абстрактном классе в Java можно объявить и определить конструкторы. Поскольку создавать экземпляры абстрактных классов нельзя, вызвать такой конструктор можно только при формировании цепочки конструкторов, то есть при создании экземпляра конкретного класса-реализации.

2. Могут ли абстрактные классы в языке Java реализовывать интерфейсы? Должны ли они реализовывать все методы?

Да, абстрактные классы могут реализовывать интерфейсы с помощью ключевого слова implements. Поскольку они абстрактные, то не обязаны реализовывать все методы. Наличие абстрактного базового класса и интерфейса для объявления типа является рекомендуемой практикой. Пример — интерфейс java.util.List и соответствующий абстрактный класс java.util.AbstractList. Поскольку AbstractList реализует все общие методы, то конкретные реализации (например, LinkedList и ArrayList) не должны реализовать все методы, как в случае, если бы они реализовали интерфейс List напрямую.

Почему в Java нет множественного наследования классов?

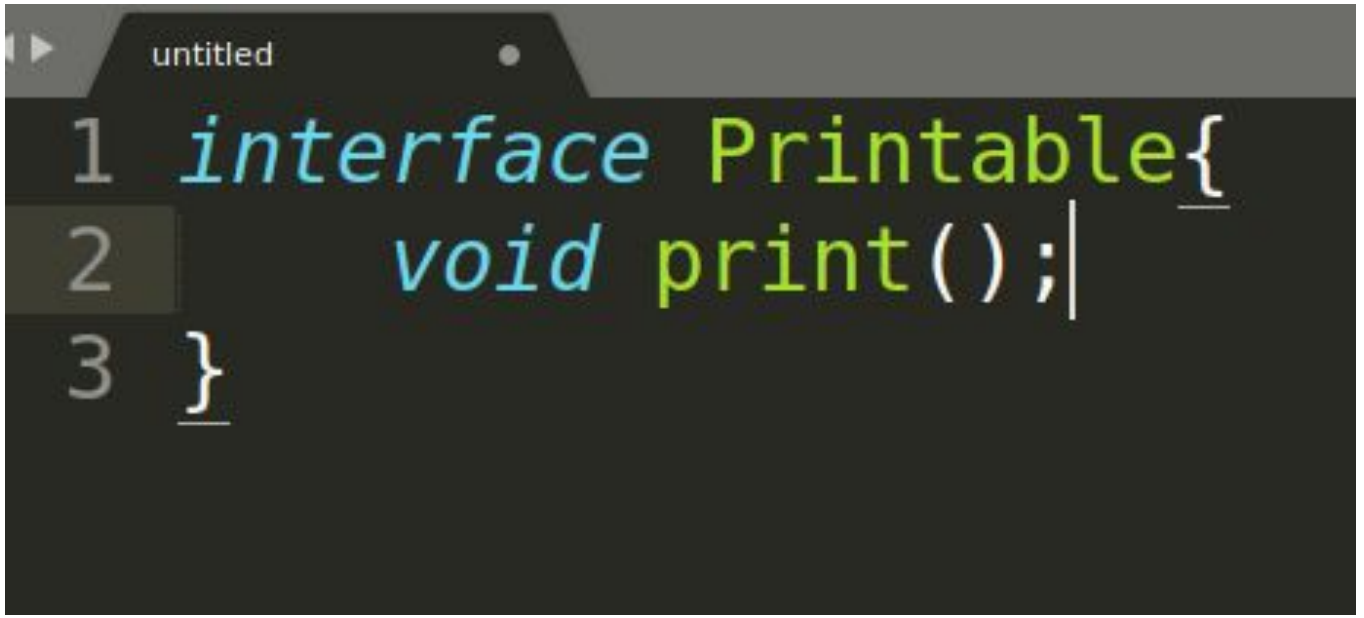
Мы уже говорили, что в Java нет множественного наследования, но так толком и не разобрались почему. Давай попробуем сделать это сейчас. Дело в том, что если бы в Java было множественное наследование, дочерние классы не могли бы определиться, какое именно поведение им выбрать.

Практика

1. Создать абстрактный класс "Машина"
2. Создать абстрактный класс "Фигура"

Интерфейсы

- Механизм наследования очень удобен, но он имеет свои ограничения. В частности мы можем наследовать только от одного класса, в отличие, например, от языка C++, где имеется множественное наследование.
- В языке Java подобную проблему частично позволяют решить интерфейсы. Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов.


untitled
1 interface Printable{
2 void print();
3 }

Разница между интерфейсом и абстрактным классом?

- Абстрактный(Abstract) класс - класс, который имеет хотя б 1 абстрактный (не определенный) метод; обозначается как abstract. Интерфейс - такой же абстрактный класс,только в нем не может быть свойств и не определены тела у методов.

  Так же стоит заметить, что абстрактный класс наследуется (extends), а интерфейс реализуется (implements). Вот и возникает разница между ними, что наследовать мы можем только 1 класс, а реализовать сколько угодно.
  ВАЖНО! При реализации интерфейса, необходимо реализовать все его методы, иначе будет Fatal error, так же это можно избежать, присвоив слово abstract.

Методы по умолчанию

- Ранее до JDK 8 при реализации интерфейса мы должны были обязательно реализовать все его методы в классе. А сам интерфейс мог содерать только определения методов без конкретной реализации. В JDK 8 была добавлена такая функциональность как **методы по умолчанию**. И теперь интерфейсы кроме определения методов могут иметь их реализацию по умолчанию, которая используется, если класс, реализующий данный интерфейс, не реализует метод.

```java
interface Printable {
    default void print(){
        System.out.println("Undefined printable");
    }
}
```