

Java Professional

lecture #2. Inheritance. Abstract classes. Polymorphism.

lecture #2. Inheritance. Abstract classes. Polymorphism.

- Inheritance
 - Важная терминология
 - Использование наследования
 - Типы наследования
 - Факты и только факты
- Abstract classes
 - Understanding. Syntax
 - Important Conclusions
 - Abstract class VS Interface
- Polymorphism
 - overloading
 - overriding

Наследование в Java (Inheritance)

- Это аспект в java, с помощью которого одному классу разрешено наследовать (перенять) свойства (поля и методы) другого класса.
- С помощью наследования можно расширить функционал уже имеющихся классов

Ключевое слово, используемое для наследования, — **extends**

Терминология и свойства

- Super Class: класс, функции которого наследуются, называется **суперклассом**.
- Sub Class: класс, который наследует другой класс, называется **подкласс**.
- Reusability: Наследование поддерживает концепцию «**повторного использования**»

Свойства:

1. Наследуем только один класс. Один класс - один родитель.
2. Наследуется все кроме private переменных и private методов.
3. Переопределить @Override метод класса-родителя.
4. Вызываем методы родителя через ключевое слово super.
5. Запрещаем наследование – **final**.



Как использовать

Ключевое слово, используемое для наследования, — **extends**.

```
class Ananas extends Fruit
```

Класс Fruit является базовым классом, класс Ananas является производным классом, который расширяет класс Fruit.

При создании объекта класса Ananas для копия всех полей суперкласса выделяется память в этом объекте.


??? вопрос

Сколько объектов создается?



Типы наследования

- 1. Одиночное наследование.
- 1. Многоуровневое наследование.
- 1. Иерархическое наследование.
- 1. Множественное наследование



Факты! Коллеги, только факты о наследовании!


!Суперкласс по умолчанию: кроме класса Object, у которого нет суперкласса, каждый класс имеет один и только один прямой суперкласс (одиночное наследование).

!Суперкласс может быть только один: Суперкласс может иметь любое количество подклассов. Но подкласс может иметь только один суперкласс.

!Наследование конструкторов: подкласс наследует все члены (поля, методы и вложенные классы) от своего суперкласса.

Конструкторы не являются членами, поэтому они **НЕ** наследуются подклассами.

!Наследование private: подкласс не наследует закрытые члены своего родительского класса. (доступ может быть только через геттеры и сеттеры)



Abstract Classes - understanding

abstract — это ключевое слово в java, применимо к классам, методам, но **НЕ** к переменным.

Используется для достижения абстракции, которая является одним из аспектов объектно-ориентированного программирования (ООП).

Абстрактные классы - имеющий частичную реализацию (т. е. не все методы, присутствующие в классе, имеют определение метода).

Чтобы объявить абстрактный класс, используйте этот синтаксис:

```
public abstract class class-name {  
    // body of the class  
}
```


Abstract Classes - Important Conclusions

1. Экземпляр абстрактного класса не может быть создан.
2. Конструкторы в абстрактном классе разрешены.
3. У нас может быть абстрактный класс без какого-либо абстрактного метода.
4. В абстрактном классе может быть final метод , но final метод не может быть абстрактным сам по себе (final abstract -> wrong combination)
5. Мы можем определить статические методы в абстрактном классе.
6. Мы можем использовать ключевое слово abstract для объявления классов верхнего уровня (внешний класс), а также внутренних классов как абстрактных.
7. Если класс содержит хотя бы один абстрактный метод, то обязательно следует объявить класс абстрактным.
8. Если класс наследник, не может обеспечить реализацию всех абстрактных методов родительского класса , мы должны объявить этот класс абстрактным , чтобы класс следующего уровня обеспечивал реализацию оставшегося абстрактного метода.

Abstract Classes - Abstract class VS Interface

1. Интерфейс описывает только поведение. У него нет состояния. А у абстрактного класса состояние есть: он описывает и то, и другое.
1. Интерфейс может иметь только абстрактные методы. Абстрактный класс может иметь абстрактные и неабстрактные методы.
1. Абстрактный класс связывает между собой и объединяет классы. Но, один и тот же интерфейс могут реализовать классы, у которых вообще нет ничего общего.
1. Классы могут реализовывать сколько угодно интерфейсов, но наследоваться можно только от одного класса.
1. Переменные, объявленные в интерфейсе Java, по умолчанию являются окончательными. Абстрактный класс может содержать переменные, не являющиеся конечными.
1. Абстрактный класс может обеспечить реализацию интерфейса. Интерфейс не может обеспечить реализацию абстрактного класса.

Полиморфизм в Java (Polymorphism)

- Полиморфизм означает наличие множества форм.
- Полиморфизм позволяет нам выполнять одно и то же действие разными способами.
- Полиморфизм позволяет определить один интерфейс и иметь несколько реализаций.

В Java делится на два типа:

Полиморфизм времени компиляции (**overloading**)

Полиморфизм времени выполнения (**overriding**)

Полиморфизм времени компиляции (overloading)

(Overloading) Перегрузка методов.

- Когда имеется несколько методов с одинаковыми именами, но разными параметрами - называются перегруженными.
- Методы могут быть перегружены изменением количества аргументов и/или изменением типа аргументов.

Полиморфизм времени выполнения (overriding)

(Overriding) Переопределение метода -> @Override

- Версия выполняемого метода будет определяться объектом, который используется для его вызова.
 - Позволяет подклассу или дочернему классу предоставлять конкретную реализацию метода.
- Правила переопределения метода:
1. Модификатор доступа для переопределяющего метода может разрешить больший, но не меньший доступ, чем переопределенный метод.
 2. final методы не могут быть переопределены: если мы не хотим, чтобы метод был переопределен, мы объявляем его как final
 3. Статические методы не могут быть переопределены.
 4. private методы не могут быть переопределены.
 5. Переопределяющий метод должен иметь тот же тип возвращаемого значения.
 6. Вызов переопределенного метода из подкласса: мы можем вызвать метод родительского класса в переопределяющем методе, используя ключевое слово super
 7. Переопределение и конструктор: мы не можем переопределить конструктор, поскольку родительский и дочерний классы никогда не могут иметь конструктор с одинаковым именем