НАСЛЕДОВАНИЕ

АБСТРАКТНЫЕ КЛАССЫ

ИНТЕРФЕЙСЫ

Наследование позволяет значительно сократить количество дублируемого кода и предоставляет легкую эволюцию объектов.

Суть заключается в том, что дочерний класс наследует свойства и методы родителя.

Таким образом, потомок имеет весь функционал родительского класса без прямого копирования кода, после чего может добавлять свои свойства и методы или расширять методы родителя.

Некоторые правила наследования

1. Ключевое слово extends используется для реализации наследования
2. Java не поддерживает множественное наследование классов
3. Дочерний класс должен расширять, а не полностью изменять функционал родителя
4. Свойства и методы родителя доступны для дочерних классов согласно модификаторам доступа
5. Методы родителя можно расширить или переопределить в дочернем классе
6. Конструкторы родителя не наследуются подклассами
7. Если родитель не имеет конструктора по умолчанию, то подкласс должен иметь явный конструктор

```java
// класс Knight наследует (расширяет) класс BattleUnit
// класс Knight - дочерний класс (или подкласс)
// класс BattleUnit - родитель (или суперкласс)
public class Knight extends BattleUnit{
    private int additionalHealth;

    public Knight(int health, int attack, int additionalHealth) {
        super(health, attack); // вызов конструктора родителя
        setAdditionalHealth(additionalHealth);
    }
    public int getAdditionalHealth() {
        return additionalHealth;
    }
    private void setAdditionalHealth(int additionalHealth){
        if (additionalHealth < 1) {
            throw new IllegalArgumentException("additionalHealth < 1");
        }
        this.additionalHealth = additionalHealth;
    }
    @Override // указывает компилятору на переопределение, реализацию метода
    public void attackEnemy(BattleUnit enemy) {
        super.attackEnemy(enemy); // вызов метода родителя
        // расширения функционала метода attackEnemy
        if (!enemy.isAlive()) setHealth(getHealth() + additionalHealth);
    }
}
```

```java
public class App {
    public static void main(String[] args) {
        Knight knight1 = new Knight(34, 10, 7);
        Knight knight2 = new Knight(28, 11, 11);

        // метод attackEnemy переопределен в классе Knight
        knight1.attackEnemy(knight2);

        // метод isAlive унаследован без изменений
        System.out.println(knight1.isAlive());
        System.out.println(knight2.isAlive());

        // метод объявлен в дочернем классе
        System.out.println(knight1.getAdditionalHealth());
    }
}
```

```java
public class App {
    public static void main(String[] args) {
        /* Можно использовать тип данных родителя (или интерфейса)
           в качестве указания на тип данных.
           В таком случае объектам для вызова будут доступны
           только методы, объявленные в суперклассе (или интерфейсе).
           Если метод переопределен в дочернем классе,
           то будет использоваться его реализация.
        */
         BattleUnit knight1 = new Knight(34, 10, 7);
         BattleUnit knight2 = new Knight(28, 11, 11);

         // метод attackEnemy переопределен в классе Knight
         knight1.attackEnemy(knight2);

         // метод isAlive унаследован без изменений
         System.out.println(knight1.isAlive());

         // метод недоступен, тк не объявлен в суперклассе
         System.out.println(knight1.getAdditionalHealth());
    }
}
```

Абстрактный класс — это класс, который может иметь объявленные, но нереализованные методы. Невозможно создать экземпляр такого класса.

Абстрактные классы используются, чтобы создать класс с общим для подклассов функционалом. При этом подразумевается, что в программе нет необходимости создавать экземпляры данного класса, только подклассов.

Чтобы создать абстрактный класс, нужно дописать ему ключевое слово abstract при объявлении класса.

Интерфейс может иметь только объявления методов и констант, default и static методы с реализацией (java 8). Невозможно создать экземпляр интерфейса.

Обычные классы, которые наследуют абстрактный класс или реализуют интерфейс обязаны иметь реализацию всех абстрактных методов (методов без реализации). Обычный класс может имплементировать больше одного интерфейса.

Сам интерфейс может наследовать (extends) несколько интерфейсов.

Некоторые правила интерфейсов

1. interface используется для создания интерфейса
2. интерфейсы не могут иметь конструкторов
3. по умолчанию любой атрибут интерфейса является public, static и final
4. по умолчанию методы интерфейса являются abstract и public
5. модификатор private доступен, начиная с Java 9
6. static методы с реализацией, доступны, начиная с Java 8
7. default методы с реализацией, доступны, начиная с Java 8 (можно не переопределять)

Некоторые правила интерфейсов

1. интерфейс может реализовать другой интерфейс или интерфейсы, например
   public interface Figures extends PrintAble, CloneAble{}
2. ключевое слово implements используется классами для реализации интерфейса
3. класс, реализующий интерфейс, должен обеспечить реализацию всех его методов, если только это не абстрактный класс

```java
// интерфейс Worker наследует интерфейс AutoCloseable
public interface Worker extends AutoCloseable {
    void start();
    void stop();
    boolean isActive();
    default void printState(){
        if (isActive()) System.out.println("Запущен");
        else System.out.println("Завершен");
    }
}
```

```java
// класс ClientWorker реализует интерфейс Worker
// класс ClientWorker не является абстрактным
// в классе необходимо реализовать абстрактные методы Worker
// в классе необходимо реализовать абстрактные методы AutoCloseable
public class ClientWorker implements Worker {
    @Override
    public void start() {
        // реализация метода
    }

    @Override
    public void stop() {
        // реализация метода
    }

    @Override
    public boolean isActive() {
        // реализация метода
        return false;
    }

    @Override
    public void close() {
        // реализация метода
    }
}
```

```java
public class App {
    public static void main(String[] args) {
        // интерфейсы можно использовать
        // для указания типа данных при объявлении переменной
        // в такой ситуации для вызова объекту будут доступны
        // только методы, объявленные в интерфейсе
        Worker client = new ClientWorker();
        client.start();
        client.stop();
        client.close();
        client.isActive();
        client.printState();

        AutoCloseable client2 = new ClientWorker();
        // доступен только метод close(),
        // который объявлен в интерфейсе AutoCloseable
        client2.close();
    }
}
```