1. Полиморфизм

Чтобы код был лаконичным, не содержал дубляжа, но при этом работал с самыми разными данными, мы можем использовать один из главных принципов ООП — **полиморфизм**.

Само слово происходит от греческих слов *«много»* и *«формы»* и описывает механизм работы с разными типами объектов как с одним, при этом не «раздувая» и не дублируя код.

> **Полиморфизм** — это способность системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

> **Как это работает?**
>
> Потомки одного родительского класса имеют в том числе и общий родительский функционал.
>
> Поэтому несколько объектов с общим родителем мы можем объединить в **массив** и работать с ними так, как будто работаем с их общим родителем.

Посмотрим на примере.

Создадим родительский класс PrintedProduct.

```
public class PrintedProduct {

    // Добавим несколько полей,
    // которые являются общими для классов-наследников
    private String name;
    private int pageQuantity;
    private String content;

    public PrintedProduct(String name, int pageQuantity, String content) {
        this.name = name;
        this.pageQuantity = pageQuantity;
        this.content = content;
    }

    // Опишем метод, который выводит информацию в консоль
    public void printContent() {
        System.out.println(content);
    }

    ... // Геттеры и сеттеры класса PrintedProduct

}
```

Далее создадим первый класс-наследник.

```
public class Book extends PrintedProduct {

    // Добавим уникальное поле
    private String authorName;

    public Book(String name, int pageQuantity, String authorName, String content
        super(name, pageQuantity, content);
        this.authorName = authorName;
    }

    public String getAuthorName() {
        return authorName;
    }

}
```

Теперь создадим еще одного наследника, который не содержит уникальных элементов.

```
public class Magazine extends PrintedProduct {

    public Magazine(String name, int pageQuantity, String content) {
        super(name, pageQuantity, content);
    }

}
```

Далее в классе с методом main реализуем следующий код.

```
public class Main {

    public static void main(String[] args) {

        PrintedProduct[] products = new PrintedProduct[] {
            new Book("War and Peace", 1000, "Lev Tolstoy", "War and Peace content"
            new Magazine("Java Magazine", 100, "Lots of information about Java"),
            new Magazine("Java Magazine2", 150, "Lots of information about Java, 2
        };

        for (int i = 0; i < products.length; i++) {
            PrintedProduct p = products[i];
            p.getAuthorName(); // Будет ошибка
            // Независимо от того, книга лежит или журнал в ячейке i,
            // мы имеем ссылку типа PrintedProduct,
            // а в нем о таком методе никто не знает
            p.printContent(); // Метод сработает верно,
            // так как метод printContent существует в PrintedProduct
        }

    }

}
```

Независимо от того, книга лежит в массиве или журнал, мы сможем работать исключительно с тем набором полей и методов, что объявлены в их общем родителе `PrintedProduct` .

Метод в качестве аргумента может принимать объект родительского класса: при вызове метода можно ему в скобки передать любого наследника данного родителя. Все классы-наследники будут выполняться с одной логикой.

Давайте создадим еще один класс с методом main и реализуем в нем метод

`printContent` , но уже с передаваемым в него аргументом типа PrintedProduct:

```
public class Main2 {

   public static void main(String[] args) {

   }

   // Создадим метод вне метода main
   public static void printContent(PrintedProduct product) {
      product.printContent();
   }

}
```

Теперь мы можем его вызвать с помощью любого объекта, который был унаследован напрямую или через цепочку наследований (`c` наследует `b` , `b` наследует `a`).

```
public class Main2 {

   public static void main(String[] args) {

      // Создадим 2 объекта
      Book b = new Book("War and Peace", 1000, "Lev Tolstoy", "War and Peace co
      Magazine m = new Magazine("Java Magazine", 100, "Lots of information abou

      // И передадим их в качестве параметра в метод
      printContent(b);
      printContent(m);

   }

   public static void printContent(PrintedProduct product) {
      product.printContent();
   }

}
```

Подведем итоги

> Любой прямой или косвенный наследник класса может быть использован в качестве экземпляра своего родителя.

За набор полей и методов отвечает тип ссылки (т. е. тип, который указан левее от оператора `=` при создании объекта).

За то, что выполняется при вызове этих методов, отвечает код после `=` .

Помните, что наследовать можно исключительно от одного класса. Множественное наследование в Java невозможно.