1. Наследование

> **Наследование** — это механизм, позволяющий создавать новый класс на основе кода уже имеющегося функционала в другом классе **без необходимости дублировать его**.

> В программировании принято говорить о наследовании и как о **механизме**, и как о **принципе ООП**, предписывающем пользоваться этим механизмом.

Возьмем, для примера, приложение для учета печатной продукции издательства.

Мы получили такой проект и поняли, что сначала нам нужно реализовать объекты для каждого типа продукции. Их два: книга и журнал.

```
class Book { // Создаем класс «Книга»
   private String name; // Строка «Название»
   private String authorName; // Строка «Имя автора»
   private int pageQuantity; // Строка «Количество страниц»
   private String content; // Строка «Содержание»

   public void printContent() {
      System.out.println(content);
   }

   ... // Геттеры и сеттеры
}

class Magazine { // Создаем класс «Журнал»
   private String name; // Строка «Название»
   private int pageQuantity; // Строка «Количество страниц»
   private String content; // Строка «Содержание»

   public void printContent() {
      System.out.println(content);
   }

   ... // Геттеры и сеттеры
}
```

Как можно заметить, оба типа печатных изделий имеют много общего: название, количество страниц и содержание есть и там, и там. Отличаются же они только полем

`authorName` , потому что у книги почти всегда есть автор, а журнал выпускается редакцией.

> Дублирование кода значительно **ухудшает читабельность и занимает дополнительные ресурсы**. А у нас как раз складывается ситуация, когда две сущности практически дублируют друг друга.

Наследование позволяет нам вынести общий для двух объектов код в отдельную сущность.

Объявим класс, который будет хранить общий код. Назовем его, например, печатным продуктом.

```
class PrintedProduct {
   private String name;
   private int pageQuantity;
   private String content;

   public void printContent() {
      System.out.println(content);
   }

   ... // Геттеры и сеттеры
}
```

С помощью механизма наследования мы можем заставить объекты («Книга» и «Журнал») «унаследовать» эти поля и методы.

> Для этого используется ключевое слово **extends** («расширяет»).

```
class Book extends PrintedProduct {
   private String authorName;
   // Так как у книги, в отличии от журнала, есть конкретный автор,
   // создадим это поле в унаследованном классе

   ... // Геттер и сеттер для authorName
}

class Magazine extends PrintedProduct {
// Так как журнал не имеет каких-то дополнительных полей,
// нам достаточно полей и поведения из родительского класса
}
```

Теперь мы можем создать любой из двух объектов и вызвать метод

`printContent` , который они унаследовали от родителя.

> Следует знать, что **private-члены класса** (поля и методы) **не наследуются**. Это значит, что ими будет управлять родитель, но доступа к ним из наследника не будет.

Когда мы инициализируем поле

`content` через конструктор, единственный способ обратиться к нему из кода класса «Книга» или класса «Журнал» будет через геттеры и сеттеры.

Без сеттера класс «Книга» не сможет изменять поле

`content` — оно находится в классе-родителе и управляется исключительно им.

Инициализация переменных, к которым нет доступа

Как мы инициализируем переменные при создании книги, если к ним нет доступа?

Java неявно подкладывает объект класса-родителя в ваш объект и присваивает его переменной

`super` (по аналогии с `this` , которая ссылается на ваш текущий объект).

По переменной

`super` , соответственно, можно вызвать конструктор родителя.

Нужно запомнить, что **строка с вызовом конструктора родителя присутствует в конструкторе вашего класса-наследника всегда**, просто если ее не пишете вы, то ее пишет Java.

Изменим наш код, добавив в него конструкторы:

```
public class PrintedProduct {
   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;
   }

   public void printContent() {
      System.out.println(content);
   }

   ... // Геттеры и сеттеры
}

public class Book extends PrintedProduct {
   private String authorName;

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

   ... // Геттер и сеттер для authorName
}
```

```
public class Magazine extends PrintedProduct {

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

Первый вариант кода тоже можно назвать рабочим. Так как в нем нет конструктора, Java сама создала бы пустой конструктор, а в нем вызвала бы пустой конструктор класса-родителя. Все поля пришлось бы устанавливать через сеттеры.

Сгенерированный конструктор выглядел бы так:

```
public Book() {
   super();
}
```

Теперь мы можем создать книгу и журнал.

```
public class Main {
   public static void main(String[] args) {
      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

      b.printContent(); // Будет напечатано: War and Peace content
      m.printContent(); // Будет напечатано: Lots of information about Java
}
```

Если наследники должны иметь внутри себя логику по работе с содержимым полей родителя, то эти поля (или методы) должны быть помечены модификаторами `default` (модификатор может отсутствовать, если родитель и наследник лежат в одном пакете) или `protected` .

Запрет наследования

Помимо возможности реализовывать функционал родительского класса в классах наследниках, у нас есть возможность наследование запретить. Этот прием пригодится вам, если потребуется сделать так, чтобы какая-то часть функционала **не** переходила потомкам.

Для того чтобы сделать наследование от определенного класса невозможным, необходимо при его объявлении указать ключевое слово

`final` .

```
public final class Book {
    private String name;
    private String authorName;
    private int pageQuantity;
    private String content;

    ... // Геттеры и сеттеры

}
// Мы не можем унаследоваться от данного класса
```

Также ключевое слово

`final` может использоваться:

- С методами — такие методы невозможно переопределить.
- С переменными — значение таких переменных нельзя изменить после их первоначальной инициализации.