Урок 3

Инкапсуляция

Знакомство с одним из трех основополагающих принципов ООП.

Основополагающие принципы ООП

ООП базируется на трех основных принципах: наследовании, инкапсуляции и полиморфизме.

Инкапсуляция (encapsulation) — это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса). При использовании объектно-ориентированного подхода не принято применять прямой доступ к свойствам класса из методов других классов. Задействуют специальные методы класса, чтобы получать и изменять его свойства.

Внутри объекта данные и методы могут обладать различной степенью доступности. Открытые члены класса составляют внешний интерфейс объекта. Это та функциональность, которая доступна другим классам. Закрытыми обычно объявляются все свойства класса, а также вспомогательные методы, которые являются деталями реализации — от них не должны зависеть другие части системы.

Скрывая реализации за внешним интерфейсом класса, можно менять его внутреннюю логику, не затрагивая код остальных компонентов системы. Это свойство называется модульность.

Обеспечивая доступ к свойствам класса только через его методы, проще контролировать корректные значения полей — ведь прямое обращение к свойствам отслеживать невозможно, и им могут присвоить некорректные значения.

Кроме того, не составит труда изменить способ хранения данных. Если информация станет храниться не в памяти, а в долговременном хранилище (файловой системе или базе данных), потребуется изменить только ряд методов одного класса, а не вводить эту функциональность во все части системы.

Наконец, программный код, написанный с использованием данного принципа, легче отлаживать. Чтобы узнать, кто и когда изменил свойство интересующего нас объекта, достаточно добавить вывод отладочной информации в тот метод, посредством которого осуществляется доступ к свойству этого объекта. При использовании прямого доступа к свойствам объектов программисту пришлось бы добавлять вывод отладочной информации во все участки кода, где используется интересующий объект.

Модификаторы доступа

Разберемся с назначением ключевых слов **private** и **public** в языке Java, ведь именно с их помощью реализуется принцип инкапсуляции.

Способ доступа к члену класса (методу или свойству) определяется модификатором доступа, присутствующим в его объявлении. Некоторые аспекты управления доступом связаны главным образом с наследованием и пакетами — их рассмотрим позднее.

В Java определяются следующие модификаторы доступа: **public**, **private** и **protected**, а также уровень доступа, предоставляемый по умолчанию (**default**). **Public** член класса доступен из любой части программы. А компонент, объявленный как **private**, недоступен для находящихся за пределами его класса. Если в объявлении члена класса отсутствует явно указанный модификатор доступа, он доступен для других классов и подклассов из данного пакета. Такой уровень доступа используется по умолчанию. Чтобы элемент был доступен за пределами его текущего пакета, но только классам, непосредственно производным от данного, элемент должен быть объявлен **protected**.

Модификатор доступа предшествует остальной спецификации типа члена:

```
public int a;
protected char b;
private void cMethod(float x1, float x2) {
    // ...
}
```

Доступ к данным объекта должен осуществляться только через методы, определенные в его классе. Поле экземпляра может быть открытым, но по веским причинам. Для доступа к данным с модификатором **private** обычно используются геттеры и сеттеры. Геттер позволяет узнать содержимое поля — как правило, его имя такое же, как у поля, для которого он создан, с добавлением слова **get** в начале. Тип геттера и поля тоже должны совпадать. Сеттер используется для изменения значения поля, объявляется как **void** и именуется по аналогии с геттером (вместо **get** в начале имени метода ставится **set**). Сеттер позволяет добавлять ограничения на изменение полей. В примере ниже показано, как с помощью сеттера запретить указывать коту отрицательный возраст. Если для поля сделать только геттер, вне класса оно будет доступно только для чтения.

```
public class Cat {
    ...
    private String name;
    private int age;

    public void setAge(int age) {
        if (age > 0)
            this.age = age;
        else
            System.out.println("Введен некорректный возраст");
    }
    public int getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```