Урок 5

Введение в ООП

Введение в объектно-ориентированное программирование, классы, объекты, конструкторы, инкапсуляция, модификаторы доступа

Что такое класс

Класс определяет форму и сущность объекта и является логической конструкцией, на основе которой построен весь язык Java. Наиболее важная особенность класса состоит в том, что он определяет новый тип данных, которым можно воспользоваться для создания объектов этого типа, т.е. класс - это шаблон (чертеж), по которому создаются объекты, а объекты - экземпляры класса. Для определения формы и сущности класса указываются данные, которые он должен содержать, а также код, воздействующий на эти данные. Ниже представлена упрощённая форма объявления класса:

```
модификатор_доступа class имя_класса {
    тип_переменной поле1;
    тип_переменной поле2;
    ...
    тип_переменной полеN;
    тип_метода имя_метода1(список_параметров) {
        // тело метода
    }

    ...

    тип_метода имя_методаN(список_параметров) {
        // тело метода
    }
}
```

Переменные, определённые в классе, называются полями экземпляра, поскольку каждый объект класса содержит собственные копии этих переменных. Таким образом, данные одного объекта отделены и отличаются от данных другого объекта. Код содержится в теле методов. Поля экземпляра и методы, определённые в классе, называются членами класса. В большинстве классов действия над полями осуществляются через методы, определенные в этом классе.

Первый класс

Ниже приведён код класса Cat, который определяет три поля: name (кличка), color (цвет) и age (возраст), и не содержит пока никаких методов. Следует заметить, что имя класса должно совпадать с именем файла в котором он объявлен, т.е. класс Cat должен находится в файле Cat.java

```
public class Cat {
    String name;
    String color;
    int age;
}
```

Как пояснялось выше, класс определяет новый тип данных, в данном случае это Cat. Поскольку класс является лишь чертежом, приведённый выше код не приводит к появлению каких-либо объектов. Чтобы создать объект класса Cat, нужно воспользоваться оператором наподобие следующего:

```
Cat cat1 = new Cat(); // создать объект класса Cat
```

После выполнения этого оператора объект cat1 станет экземпляром класса Cat. Всякий раз, когда получается экземпляр класса, создаётся объект, который содержит собственную копию каждой

переменной экземпляра, определённой в данном классе. Таким образом, каждый объект класса Cat будет содержать собственные копии полей name, color и age. Для доступа к этим полям служит операция-точка, которая связывает имя объекта с именем поля. Например, чтобы присвоить полю color объекта cat1 значение *White*, нужно выполнить следующий оператор:

```
cat1.color = "White";
```

В общем, операция-точка служит для доступа к полям и методам объекта. Ниже приведён полноценный пример программы, в которой используется класс Cat.

```
public class CatDemo {
    public static void main(String[] args) {
        Cat cat1 = new Cat();
        Cat cat2 = new Cat();
        cat1.name = "Barsik";
        cat1.color = "White";
        cat1.age = 4;
        cat2.name = "Murzik";
        cat2.color = "Black";
        cat2.age = 6;
        System.out.println("Cat1 name: " + cat1.name + " color: " + cat1.color + " age: " + cat1.age);
        System.out.println("Cat2 name: " + cat2.name + " color: " + cat2.color + " age: " + cat2.age);
    }
}
```

Результат работы программы:

```
Cat1 name: Barsik color: White age: 4
Cat2 name: Murzik color: Black age: 6
```

При наличии двух объектов класса Cat, каждый из них будет содержать собственные копии полей name, color и age, т.е. изменение полей одного объекта не повлияет на поля другого. Данные из объекта cat1 изолированы от данных, содержащихся в объекте cat2.

Создание объектов

При создании класса появляется новый тип данных, который можно использовать для создания объектов данного типа. Но создание объектов класса представляет собой двухэтапный процесс. Сначала следует объявить переменную типа класса. Эта переменная не определяет объект. Она является лишь переменной, которая может ссылаться на объект. Затем нужно получить конкретную, физическую копию объекта и присвоить её этой переменной. Это можно сделать с помощью оператора new, который динамически резервирует память для объекта и возвращает ссылку на него. В общих чертах эта ссылка представляет собой адрес объекта в памяти, зарезервированной оператором new. Затем эта ссылка сохраняется в переменной.

```
public static void main(String[] args) {
    Cat cat1;
    cat1 = new Cat();
}
```

В первой строке кода переменная cat1 объявляется как ссылка на объект типа Cat и пока ещё не ссылается на конкретный объект. В следующей строке выделяется память для объекта, а переменной cat1 присваивается ссылка на него. После выполнения второй строки кода переменную cat1 можно использовать так, как если бы она была объектом типа Cat.

Подробное рассмотрение оператора new

Оператор new динамически выделяет оперативную память для объекта. Общая форма этого оператора имеет следующий вид:

```
Переменная_класса = new Имя_класса();
```

где переменная класса обозначает переменную создаваемого класса, а имя_класса - конкретное имя класса, экземпляр которого получается. Имя класса, за которым следуют круглые скобки, обозначает конструктор данного класса. Конструктор определяет действия, выполняемые при создании объекта класса. В большинстве классов, явно объявляются свои конструкторы, если ни один из явных конструкторов не указан, то Java автоматически сформирует конструктор по умолчанию.

При присваивании переменные ссылок на объекты действуют иначе, чем можно было бы предположить.

```
public static void main(String[] args) {
    Cat cat1 = new Cat();
    Cat cat2 = cat1;
}
```

На первый взгляд может показаться, что переменной cat2 присваивается ссылка на копию объекта cat1, т.е. переменные cat1 и cat2 будут ссылаться на разные объекты в памяти, но это не так. На самом деле cat1 и cat2, будут ссылаться на один и тот же объект. Присваивание переменной cat1 значения переменной cat2 не привело к выделению области памяти или копированию объекта, лишь к тому, что переменная cat2 ссылается на тот же объект, что и переменная cat1. Таким образом, любые изменения, внесенные в объекте по ссылке cat2, окажут влияние на объект, на который ссылается переменная cat1, поскольку это один и тот же объект в памяти.

Конструкторы

Конструктор позволяет инициализировать объект непосредственно во время его создания. Его имя совпадает с именем класса, в котором он находится, а синтаксис аналогичен синтаксису метода. Как только конструктор определен, он автоматически вызывается при создании объекта перед окончанием выполнения оператора new.

```
public class Cat {
    private String name;
    private String color;
    private int age;
    public Cat() {
        System.out.println("Это конструктор класса Cat") ;
        name = "Barsik";
        color = "White";
        age = 2;
    }
}
public class MainClass {
    public static void main(String[] args) {
        Cat cat1 = new Cat();
    }
}
```

Теперь при создании объектов класса Cat, все коты будут иметь одинаковые имя, цвет и возраст. Глядя на строку с созданием объекта cat1, должно быть ясно, почему после имени класса требуется указывать круглые скобки. В действительности оператор new вызывает конструктор класса. Если конструктор класса не определен явно, то создается конструктор по умолчанию.

Параметризированные конструкторы

В предыдущем примере конструктор позволяет создавать только одинаковых котов, что не имеет особого смысла. Для того, чтобы можно было с помощью конструктора создавать отличающиеся объекты, достаточно добавить в конструктор набор параметров.

```
public class Cat {
    private String name;
    private String color;
    private int age;
    public Cat(String _name, String _color, int _age) {
        name = _name;
        color = _color;
        age = _age;
    }
}
```

Как правило, удобно, чтобы имя параметра совпадало с именем поля, которое он заполняет. В данном случае так и есть, только перед именами параметров стоит нижнее подчеркивание (это не является каким-то правилом), это сделано для того, чтобы внутри конструктора можно было легко отличить имя поля от имени передаваемого параметра. Ниже приведён пример создания двух объектов класса Cat:

```
public static void main(String[] args) {
    Cat cat1 = new Cat("Barsik", "Brown", 4);
    Cat cat2 = new Cat("Murzik", "White", 5);
}
```

Перегрузка конструкторов

Наряду с перегрузкой обычных методов возможна перегрузка и конструкторов.

```
public class Cat {
    private String name;
    private String color;
    private int age;
    public Cat(String _name, String _color, int _age) {
        name = _name;
        color = _color;
        age = _age;
    }
    public Cat(String _name) {
        name = _name;
        color = "Unknown";
        age = 1;
    }
    public Cat() {
        name = "Unknown";
        color = "Unknown";
        age = 1;
    }
}
```

В этом случае допустимы будут следующие варианты создания объектов:

```
public static void main(String[] args) {
    Cat cat1 = new Cat();
    Cat cat2 = new Cat("Barsik");
    Cat cat3 = new Cat("Murzik", "White", 5);
}
```

Соответствующий перегружаемый конструктор вызывается в зависимости от аргументов, указываемых при выполнении оператора new.

Ключевое слово this

Иногда требуется, чтобы метод ссылался на вызвавший его объект. Для этой цели в Java определено ключевое слово this. Им можно пользоваться в теле любого метода для ссылки на текущий объект, т.е. объект у которого был вызван этот метод.

```
public class Cat {
    private String name;
    private String color;
    private int age;
    public Cat(String name, String color, int age) {
        this.name = name;
        this.color = color;
        this.age = age;
    }
}
```

Эта версия конструктора действует точно также, как и предыдущая. Ключевое слово this применяется в данном случае для того чтобы отличить имя параметра, от имени поля объекта.

Инкапсуляция

Инкапсуляция связывает данные с манипулирующим ими кодом и позволяет управлять доступом к членам класса из отдельных частей программы, предоставляя доступ к данным только с помощью определенного ряда методов, что позволяет предотвратить злоупотребление этими данными. Так, если класс реализован правильно, он создаёт своего рода "черный ящик", которым можно пользоваться, но его внутренний механизм защищен от повреждений.
Способ доступа к члену класса определяется модификатором доступа, присутствующим в его объявлении. Некоторые аспекты управления доступом связаны главным образом с наследованием и пакетами, и будут рассмотрены позднее. В Java определяются следующие модификаторы доступа: public, private и protected, а также уровень доступа, предоставляемый по умолчанию. Любой public член класса доступен из любой части программы. А любой компонент, объявленный как private, недоступен для компонентов, находящихся за пределами его класса. Если в объявлении члена класса отсутствует явно указанный модификатор доступа, он доступен для подклассов и других классов из данного пакета. Этот уровень доступа используется по умолчанию. Если же требуется, чтобы элемент был доступен за пределами его текущего пакета, но только классам, непосредственно производным от данного класса, такой элемент должен быть объявлен как protected.

Модификатор доступа предшествует остальной спецификации типа члена.

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

Как правило, доступ к данным объекта должен осуществляться только через методы, определенные в классе этого объекта. Поле экземпляра вполне может быть открытым, но на то должны иметься веские основания. 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 также должен совпадать с типом поля. Сеттер используется для изменения значения поля, объявляется как void, и именуется по аналогии с геттером (только вместо get в начале имени метода стоит слово set). Кроме того, сеттер позволяет добавлять ограничения на изменение полей, в примере ниже, с помощью сеттера не получится указать коту отрицательный возраст. Если для поля сделать только геттер, то вне класса это поле 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;
    }
}
```

В Java представлено три модификатора доступа: private, public, protected:

	private	**Модификатор отсутствует**	**protected**	**public**
Один и тот же класс	+	+	+	+
Подкласс, производный от класса из того же самого пакета	-	+	+	+
Класс из того же самого пакета, не являющийся подклассом	-	+	+	+
Подкласс, производный от класса другого пакета	-	-	+	+
Класс из другого пакета, не являющийся подклассом, производный от класса данного пакета	-	-	-	+

Дополнительные вопросы

Сборка «мусора». Поскольку выделение оперативной памяти для объектов осуществляется динамически с помощью оператора new, в процессе выполнения программы необходимо периодически и удалять объекты из памяти. В Java освобождение оперативной памяти осуществляется автоматически и называется сборкой «мусора». В отсутствие любых ссылок на объект считается, что этот объект больше не нужен и занимаемую им память можно освободить. Во время выполнения программы сборка «мусора» выполняется только изредка и не будет выполняться лишь потому, что один или несколько объектов больше не используются.

Ключевое слово static. Иногда возникает необходимость создать поле класса, общее для всех объектов этого класса, или метод, который можно было бы использования без создания объектов класса, в котором прописан этот метод. Обращение к такому полю или методу должно осуществляться через имя класса. Для этого в начале объявления поля или метода ставится ключевое слово static. Когда член класса объявлен как static (статический), он доступен до создания любых объектов его класса и без ссылки на конкретный объект. Наиболее распространённым примером статического члена служит метод main(). При создании объектов класса, копии статических полей не создаются, и все объекты этого класса используют одно и то же статическое поле.

На методы, объявленные как static, накладывается следующие ограничения:

- Они могут непосредственно вызывать только другие статические методы;
- Им непосредственно доступны только статические переменные;
- Они никоим образом не могут делать ссылки типа this или super.

Домашнее задание

1. Создать класс "Сотрудник" с полями: ФИО, должность, email, телефон, зарплата, возраст;

Конструктор класса должен заполнять эти поля при создании объекта;

Внутри класса «Сотрудник» написать метод, который выводит информацию об объекте в консоль;

Создать массив из 5 сотрудников:

Пример:
Person[] persArray = new Person[5]; // Вначале объявляем массив объектов
persArray[0] = new Person("Ivanov Ivan", "Engineer", "ivivan@mailbox.com", "892312312", 30000, 30); // потом для каждой ячейки массива задаем объект
persArray[1] = new Person(...);
...
persArray[4] = new Person(...);

С помощью цикла вывести информацию только о сотрудниках старше 40 лет;

Дополнительные материалы

1. Кей С. Хорстманн, Гари Корнелл Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. - М.: Вильямс, 2014. - 864 с.
2. Брюс Эккель Философия Java // 4-е изд.: Пер. с англ. – СПб.: Питер, 2016. – 1168 с.
3. Г. Шилдт Java 8. Полное руководство // 9-е изд.: Пер. с англ. - М.: Вильямс, 2015. - 1376 с.
4. Г. Шилдт Java 8: Руководство для начинающих. // 6-е изд.: Пер. с англ. - М.: Вильямс, 2015. - 720 с.