Урок 2

Создание классов и объектов

Практическая реализация ООП.

Создание классов и объектов

Описывая поведение объекта (например, автомобиля), мы строим его модель. Как правило, она не может описать объект полностью — реальные сущности слишком сложны для этого. Именно поэтому приходится отбирать только те характеристики, которые важны для решения поставленной задачи.

Для описания грузоперевозок центральной характеристикой будет грузоподъемность автомобиля. Чтобы описать автомобильные гонки, это несущественно, зато важен метод набора скорости (что для грузоперевозок не имеет особого значения).

Надо абстрагироваться от некоторых конкретных деталей объекта и выбрать правильную степень абстракции. Слишком высокая даст только приблизительное описание, не позволит правильно моделировать поведение объекта. Чересчур низкая степень абстракции сделает модель слишком сложной, перегруженной деталями, и потому непригодной.

Реализуем класс **Cat**, который будет определять состояние типа «Кот»: он будет иметь свойства **name** (кличка), **color** (цвет) и **age** (возраст). Имя класса должно совпадать с именем файла, в котором он объявлен: класс Cat будет находиться в файле **Cat.java**:

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

Класс определяет новый тип данных — **Cat**. Поскольку класс является лишь чертежом, этот код не приводит к появлению объектов.

Экземпляры класса создаются при помощи оператора **new**. Он динамически выделяет оперативную память для объекта. Общая форма этого оператора имеет следующий вид:

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

Здесь **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. Он определяет действия, которые выполняются при создании объекта класса. В большинстве классов явно объявляются свои конструкторы. Если ни один не указан, Java автоматически формирует конструктор по умолчанию.

Чтобы создать объект класса **Cat**, нужно воспользоваться оператором наподобие такого:

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

После выполнения этого оператора объект **cat1** будет содержать ссылку на экземпляр класса **Cat**. Переменная типа **Cat** (как и любого другого класса) будет иметь ссылочный тип, а не примитивный (с которым мы в основном работали до этого).

Каждый объект класса **Cat** будет содержать собственные значения полей **name**, **color** и **age**. Для доступа к этим полям служит операция-точка, которая связывает имя объекта и поля. Чтобы присвоить полю **color** объекта **cat1** значение ***White***, нужно выполнить следующий оператор:

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

Операция-точка служит для доступа к полям и методам объекта. Рассмотрим 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);
    }
}
```

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

Нужно помнить и о другом аспекте поведения объектов как сущностей ссылочного типа. Поскольку переменная определенного класса является ссылкой, при присваивании другой переменной поведение будет отличаться от присваивания у переменных примитивных типов. Рассмотрим код:

```
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** — то есть 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 объект. Присваивание переменной **cat1** значения переменной **cat2** не привело к выделению области памяти или копированию объекта, а только к тому, что переменная **cat2** ссылается на тот же объект, что и **cat1**. Таким образом, любые изменения, внесенные в объект по ссылке **cat2**, окажут влияние на объект, на который ссылается переменная **cat1** — ведь 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 имена, цвет и возраст. Глядя на строку с добавлением объекта **cat1**, понимаем, почему после имени класса надо указывать круглые скобки. В действительности оператор **new** вызывает конструктор класса. Если он не определен явно, создается конструктор по умолчанию.

Параметризированные конструкторы

В предыдущем примере конструктор позволяет создавать только одинаковых котов, а это не имеет особого смысла. Чтобы 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;
    }
}
```

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 правилом, а сделано, чтобы внутри конструктора можно было легко отличить имя поля и передаваемого параметра. Пример добавления 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

Как и для обычных методов, для конструкторов возможна перегрузка:

```
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 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, указанных при выполнении оператора **new**.

Ключевое слово this

Чтобы метод ссылался на вызвавший его объект, в Java определено ключевое слово **this**. Им можно пользоваться в теле любого метода для ссылки на текущий 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;
   }

   public String getCatName(){
       return this.name;
   }
}
```