Java

Лекция 1

Вступление в объектно-ориентированное программирование

Kyiv 2021

Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Объектно-ориентированное программирование разрабатывалось с целью связать поведение сущности с её данными и описать объекты реального мира и бизнес-процессы с помощью программного кода. Таким образом использование этой методологии позволяет описать предметную область задачи в более привычной форме, используя при этом модель того или иного объекта и его поведения.

Вопросы и ответы к ним

Я уже успешно писал программы на Java без использования объектно-ориентированного программирования. Обязательно ли нужно его использовать?

Действительно можно писать программы и без использования объектно-ориентированного программирования (в дальнейшем ООП). В общем случае применение ООП не обязательно.

В каких случаях применение ООП оправданно?

Применение ООП логически оправданно при разработке программ, которые манипулируют данными в виде объектов предметной области. Например, если вы пишете интернет-магазин, то у вас будут присутствовать описания товаров, заказов и т.д.. В этом случае применение ООП очень даже оправданно.

Популярно ли ООП в разработке ПО?

ООП чрезвычайно популярно.

Модельное описание объектов окружающего мира

Модель - абстрактное представление реальности в какой-либо форме (например, в математической, физической, символической, графической или дескриптивной), предназначенное для представления определённых аспектов этой реальности и позволяющее получить ответы на изучаемые вопросы.

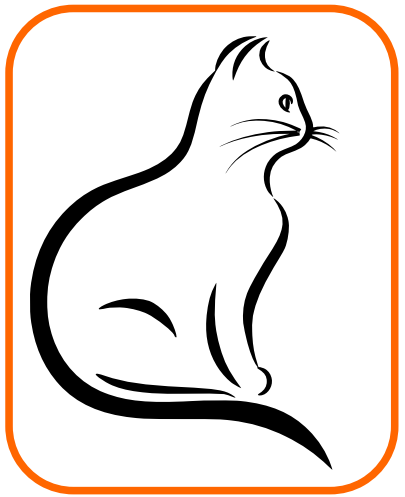
Например мы хотим описать такой объект окружающего нас мира как кота. Полное описание его вы дать не сможете(точный молекулярный состав вам вряд ли известен, а без него описание не полное), но это и не нужно. Вы довольно точно опишите кота указав его цвет, вес, имя. Вы использовали модель для его описания. И именно использование модельного описания объектов предметной области и упрощает применение объектно ориентированного программирования.

Класс

Класс - универсальный, комплексный тип данных, состоящий из набора «полей» (переменных более элементарных типов) и «методов» (функций для работы с этими полями). Он является моделью сущности с внутренним и внешним интерфейсами для оперирования своим содержимым.

По своей сути класс и является способом описания модели с помощью программного кода данного языка программирования. И если вы в своей программе будете использовать такое модельное описание того или иного объекта (кота, книги, товара или заказа) то использовать для этого нужно именно класс).

Пример проектирования класса



Предположим, что в программе нужно описать кота. Готового типа переменных для этого нет. Поэтому сначала выделим нужные нам элементы модели.

Характеристики:

- Имя
- Вес
- Цвет

Поведение:

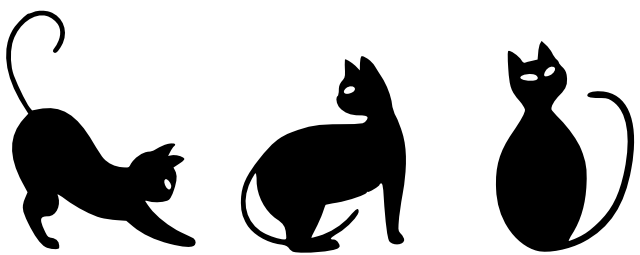
- Мяукать
- Прыгать

Требуемые характеристики (Имя, Вес, Цвет) описываются с помощью полей, а поведение (мяукать, прыгать) описываются с помощью методов. А саму модель опишем с помощью класса. Описание класса приведет к созданию пользовательского типа данных который можно будет использовать в дальнейшем в программе.

Объект (экземпляр класса)

Объект или экземпляр класса - совокупность одновременно хранимых в локальном участке памяти данных, описывающих определенное состояние и поведение, имеющая определенные свойства (атрибуты) и операции над ними (методы). В Java объекты создаются только на основе классов. Объекты принадлежат одному или нескольким классам, которые определяют поведение (являются моделью) объекта. Класс описывает свойства и методы, которые будут доступны у объекта, построенного по описанию, заложенному в классе.

И если класс это модель, то объект(экземпляр класса) это переменная созданная на его основе. И таких объектов можно будет создать произвольное количество.



Вопросы и ответы к ним

Возможность создания пользовательских типов данных (классов) и переменных на их основе (экземпляров классов) это и есть ООП?

Нет. Возможность создания пользовательских типов данных еще не означает, что язык реализует ООП.

Есть ли критерии, которым должен соответствовать язык программирования, и которые указывают на то, что ООП реализовано?

Да. Для этого в языке должно быть реализовано несколько базовых принципов ООП:

- Абстракция;
- Инкапсуляция;
- Наследование;
- Полиморфизм.

Основные принципы используемые в ООП

Для того, что бы язык программирования полностью реализовывал объектно-ориентированную парадигму программирования необходимо наличие реализации таких принципов:

- **Абстракция**
- **Инкапсуляция**
- **Наследование**
- **Полиморфизм**

Java — язык реализующий объектно-ориентированную парадигму программирования.

Абстракция

Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой. В ООП рассматривают лишь абстракцию данных (нередко называя её просто «абстракцией»), подразумевая набор наиболее значимых характеристик объекта, доступных остальной программе.

Полное описание кота с помощью языком программирования выполнить невозможно, поэтому мы заменяем полное описание модельным. При этом мы выбираем только те характеристики которые важны в контексте текущей задачи.

Мы взяли только имя, вес и цвет это и есть пример абстракции.

Инкапсуляция

Инкапсуляция — в информатике размещение в одном компоненте данных и методов, которые с ними работают. Также это понятие может быть дополнено механизмом скрытия внутренней реализации от других компонентов. Например, доступ к скрытой переменной может предоставляться не напрямую, а с помощью методов для чтения (геттер) и изменения (сеттер) её значения.

Таким образом в общем случае в разных языках программирования термин «инкапсуляция» относится к одной или обеим одновременно следующим нотациям:

1) Механизм языка, позволяющий ограничить доступ одних компонентов программы к другим;
2) Языковая конструкция, позволяющая связать данные с методами, предназначенными для обработки этих данных.

Наследование

Наследование - свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.

Класс, от которого производится наследование, называется базовым, родительским или суперклассом.

Новый класс называется потомком, наследником, дочерним или производным классом.

Полиморфизм

Полиморфизм подтипов (в ООП называемый просто «полиморфизмом») — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Например кот и кролик могут совершать прыжки. И если попросить каждого из них прыгнуть, то они совершат прыжок. Каждый по разному, но прыгнет. И хотя это объекты разных классов они могут выполнить одно и тоже действие это и есть полиморфизм.

Вопросы и ответы к ним

Нужно ли мне сразу запоминать эти термины и пытаться понять?

Желательно да. Но мы будем постоянно к ним возвращаться и раскрывать более широко, так что время еще есть.

Java отвечает всем этим критериям?

Да. Java является языком реализующим ооп. Иногда Java приводят в качестве своеобразного эталона в таком случае.

Как описываются классы в Java

Для описания классов в Java используется ключевое слово **class**. В теле класса описываются поля, и методы класса. Поля описываются в виде переменных примитивного или ссылочного типа. Написание методов вам должно быть знакомо. Интересной особенностью является то, что все методы класса имеют полный доступ к полям класса.

Наряду с пользовательскими методами класс содержит реализацию стандартных методов(методы класса Object) в явном или не явном виде. Относитесь к ним как к методам которые каждый класс получает по умолчанию.

Вопросы и ответы к ним

Где можно описывать новые классы?

Хороший вопрос. В Java новые классы можно описывать используя разные программные единицы:

1) Новый класс описывается в новом файле. Имя класса должно совпадать с именем файла. Пожалуй самый распространенный способ;
2) Новый класс можно описать в файле с уже существующим классом. Модификатор нового класса не может быть public;
3) Новый класс можно описать в теле другого класса. Такие классы называются внутренними или вложенными;
4) Новый класс можно описать в любой локальной области видимости (методе, условном операторе, цикле).

Пример нового класса Cat

```
package sample1;

public class Cat {
    String name;
    int weight;
    String color;

    public String meow() {
        return "meow meow";
    }

    public void jump() {
        System.out.println("hop");
    }

    public String toString() {
        return "Cat [name=" + name + ", weight=" + weight + ", color=" + color + "]";
    }
}
```

Поля

Примеры методов

Методы класса имеют полный доступ к полям класса

Как создать новый объект пользовательского класса

```
package sample1;

public class Main {

    public static void main(String[] args) {

        Cat cat1 = new Cat();

    }
}
```

Ссылка типа Cat

Создание нового объекта типа Cat

Для работы со ссылочными типами данных нужно объявить ссылку соответствующего типа

Работа оператора new в Java

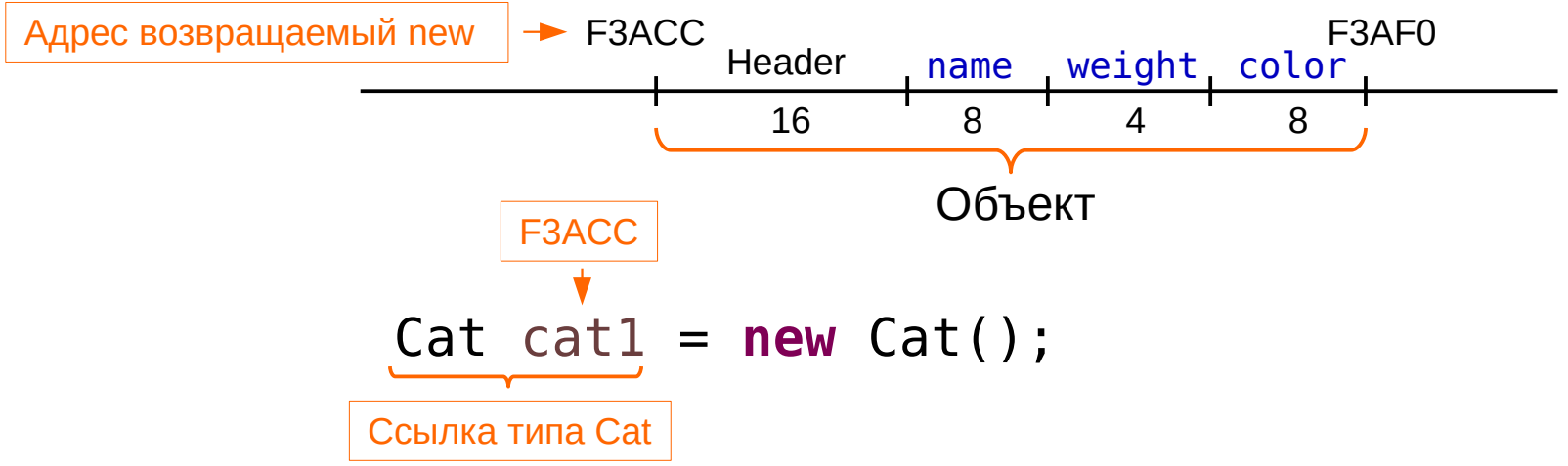
Оператор **new** в Java используется для создания и инициализации объектов. Т.е. именно он проводит выделение места в оперативной памяти (heap) и инициализацию начального значения полей.

Для любого объекта выделяется место:

- **заголовок** (сервисный системный с точки зрения JVM участок памяти). Заголовок состоит из двух частей — **mark word**, который содержит в себе информацию о блокировках, identity hashcode (или biased locking) и сборке мусора, и **class pointer**, который указывает на класс объекта. Т.е. каждый объект имеет ссылку на свой класс;
- Данные самого объекта (поля)

В результате работы оператора new создается объект, проводится его начальная инициализация и возвращается ссылка на него.

Демонстрация работы оператора new



При вызове оператора new выделяется участок к оперативной памяти (от адреса F3ACC и до F3AF0) проводится инициализация и в результате работы возвращается ссылка на указанный участок оперативной памяти. Ссылка cat1 хранит адрес оперативной памяти где находится созданный объект.

Вопросы и ответы к ним

Сколько можно создать объектов?

Количество создаваемых объектов ограничивается объемом оперативной памяти выделенной под кучу.

Можно ли создать две и более ссылки на один и тот же объект?

Да это довольно распространенная практика. И вам довольно часто будет встречаться такая ситуация.

Как получить доступ к полям и методам объекта?

Доступ к полям и методам реализуется с помощью ссылки на объект. Для этого указывается имя ссылки после которой идет «.» после которой идет имя поля или вызов метода.

Пример создания объектов и ссылок на них

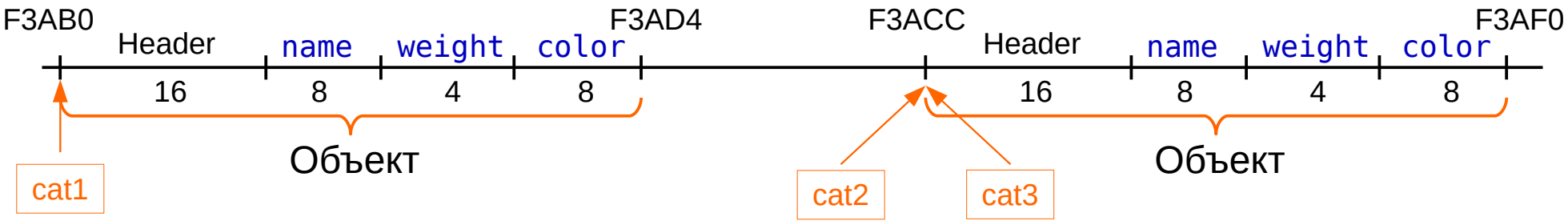
```
public static void main(String[] args) {

    Cat cat1 = new Cat();

    Cat cat2 = new Cat();


    Cat cat3 = cat2;
}
```

Создание объекта и ссылки на него

Создание объекта и ссылки на него

Создание второй ссылки на объект

Создание нескольких объектов и ссылок




```
Cat cat1 = new Cat();
```

```
Cat cat2 = new Cat()
```

```
Cat cat3 = cat2;
```

При присвоении ссылок происходит просто копирование адресов. После подобного присвоения обе ссылки хранят один и тот же адрес в оперативной памяти. Они указывают на один и тот же объект.

Доступ к полям и методам

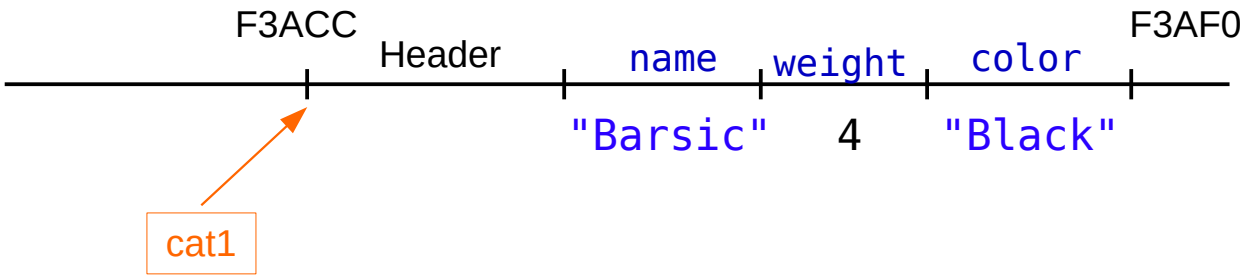
```
Cat cat1 = new Cat();

cat1.name = "Barsic";
cat1.color = "Black";
cat1.weight = 4;

System.out.println(cat1.meow());

System.out.println(cat1.toString());
```

Установка значений полей

Вызов метода

Установка значений полей

```
cat1.name = "Barsic";
cat1.color = "Black";
cat1.weight = 4;
```



Используя ссылку вы можете как читать значения полей объекта, так и устанавливать новые значения. Таким же образом вызываются и методы объекта.

Состояние объекта

Состояние объекта — совокупность значений полей объекта. Так для объекта на который указывает ссылка cat1 состоянием будет:

```
name = "Barsic"
color = "Black"
weight = 4
```

В Java для получение состояния объекта в виде строки используется метод:

```
public String toString()
```

Этот метод является стандартным (принадлежит классу Object) и вызывается когда ссылка на объект используется для получения строки. Например в методе System.out.println.

Наличие этого метода желательно во всех пользовательских классах.

Назначение класса

Именно класс в Java определяет такие параметры:

1) Количество полей у объекта;
2) Тип каждого поля;
3) Название каждого поля;
4) Набор доступных методов.

Вследствие этого у объектов одного и того же класса одинаковое количество полей, их тип и название. Также они обладают одинаковым набором доступных методов. Различается только состояние этих объектов (значение их полей). Поэтому к проектированию класса стоит относится с повышенным вниманием.

Пример с объектами класса Cat

```
Cat cat1 = new Cat();

cat1.name = "Barsic";
cat1.color = "Black";
cat1.weight = 4;

Cat cat2 = new Cat();

cat2.name = "Umka";
cat2.color = "White";
cat2.weight = 5;

System.out.println(cat1);

System.out.println(cat2);
```

Создание первого объекта и ссылки на него

Задание значения полей первого объекта

Создание второго объекта и ссылки на него

Задание значения полей второго объекта

Получение состояния первого объекта

Получение состояния второго объекта

В примере создано два объекта типа Cat. Оба обладают одинаковым набором полей и методов. Значения полей для каждого объекта можно задать разные.

Модификаторы доступа

В Java для ограничения области видимости членов класса применяются модификаторы доступа. Модификаторы доступа определяют возможность доступа к членам класса из других классов (при условии создания объекта). Для модификаторов доступа используются зарезервированные ключевые слова. В случае если модификатор доступа не был использован, то используется модификатор доступа по умолчанию.

Это модификаторы:

- **public** публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором public, видны другим классам из текущего пакета и из внешних пакетов. При условие разрешения видимости за пределами модуля видны во всем проекте;
- **protected** такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах;
- **private** закрытый класс или член класса, противоположность модификатору public. Закрытый класс или член класса доступен только из кода в том же классе.

Модификатор по умолчанию такие поля или методы видны всем классам в текущем пакете.

Вопросы и ответы к ним

Для чего стоит использовать модификатор public?

Данный модификатор стоит использовать для всего, что предназначено для явного использования во всей программе. Методы класса которые будут постоянно использоваться. Внимание! Полей это не касается. В Java не рекомендовано использование доступных за пределами класса полей.

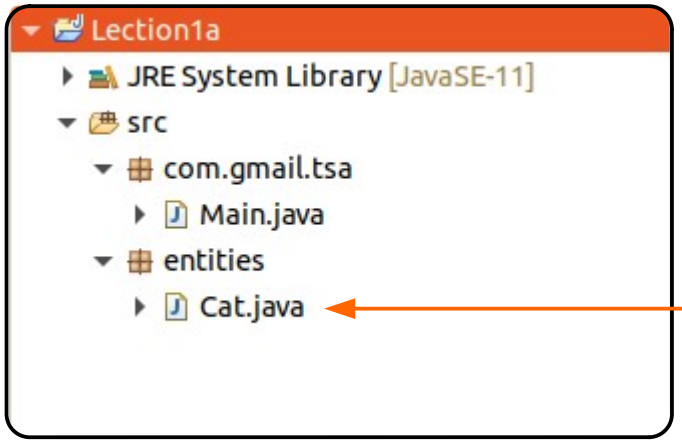
Для чего стоит использовать модификатор private?

Для членов класса которые не должны быть доступны из вне, методы реализация которых не должна быть доступна за пределами класса. Также рекомендуется все поля класса объявлять с модификатором private.

Часто ли применяются модификатор protected и модификатор по умолчанию?

Модификатор protected применяется для классов функциональность которых должна быть доступна только в их наследниках (редкий случай). Модификатор по умолчанию, используется чаще всего при написании частей утилит. Сама утилита описывается в отдельном пакете, и при описании ее частей (которые не должны быть доступны за его пределами) используется модификатор по умолчанию.

Пример применения модификаторов доступа




```java
public class Cat {
	public String name;
	protected int weight;
	private String color;

	public String meow() {
		return "meow meow";
	}

	public void jump() {
		System.out.println("hop");
	}

}
```

В примере показаны поля и методы с различными модификаторами доступа. Сам класс описан в пакете entities. Объект класса Cat создан в классе Main в пакете com.gmail.tsa. В таком случае доступ будет только к полю name и к методам meow и jump (они описаны с модификатором public).

Вопросы и ответы к ним

Что такое классы наследники (для применения с модификатором protected)?

Ответ на этот вопрос мы получим чуть позже когда будем изучать тему наследование.

Для чего делать поля private ведь к ним нельзя теперь получить доступ?

Это часть реализации механизма сокрытия он будет рассмотрен в этой лекции позже. Пока, что будем объявлять поля с модификатором по умолчанию.

Модификатор static

static — модификатор, применяемый к полю, блоку, методу или внутреннему классу. Данный модификатор указывает на привязку к текущему классу.

Особенности при применении:

- **Поле**. Такое поле становится статическим, это означает что теперь это поле существует в единственном экземпляре. Доступ к это полю возможен и через ссылку на объект и через сам класс. В таком случае говорят, что поле принадлежит классу.
- **Метод**. Такой метод может быть вызван без создания объекта. Доступен через ссылку на объект и через имя класса. Не имеет доступа к не статическим членам своего класса.

Как устроенна работа классов

Перед началом работы вашего приложения происходит загрузка классов используемых в нем. Для этого используются загрузчики классов(ClassLoaders). Класс для использования в вашей программе загружается в виде объекта типа Class. Эти объекты загружаются в специальную область памяти называемую MetaSpace. Для каждого класса создается ровно один объект типа Class. И именно на этот объект и есть ссылка в header каждого объекта.

Все поля которые вы объявите с модификатором static также будут хранится в этой области. При создании объекта это поле не будет создаваться, вы просто получите ссылку на данные из MetaSpace.

Все объявленные методы также хранятся в части MetaSpace (Method Area). У методов с модификатором static нет связи с созданным объектом. Именно поэтому из статических методов можно обратится только к статическим членам класса. У обычных методов такая связь есть.

Статическое поле класса

```
public class Cat {

    static String name;

    int weight;
    String color;

    public String meow() {
        return "meow meow";
    }

    public void jump() {
        System.out.println("hop");
    }

    public String toString() {
        return "Cat [name=" + name + ", weight=" + weight + ", color=" + color + "]";
    }
}
```

Статическое поле класса

Пример использования статических полей

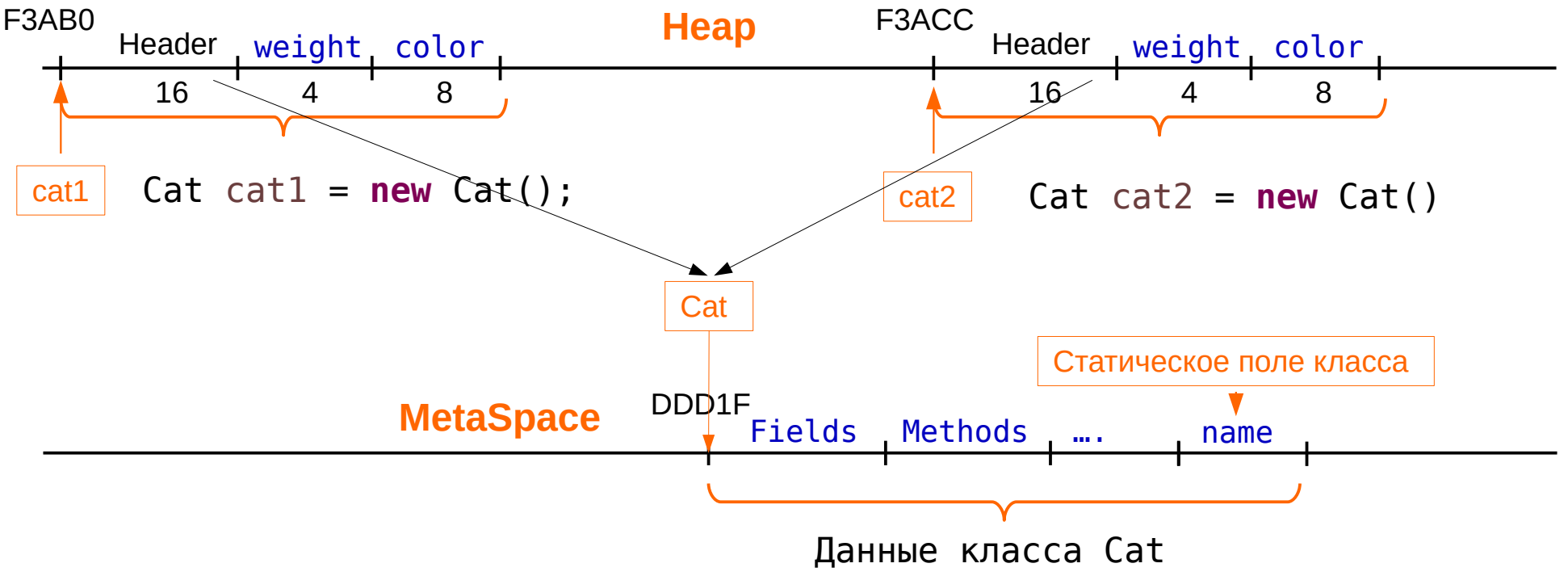
```
public class Main {

    public static void main(String[] args) {

        Cat cat1 = new Cat();

        cat1.name = "Barsic";

        Cat cat2 = new Cat();

        cat2.name = "Vaska";

        Cat.name = "Umka";

        System.out.println(cat1.name);
        System.out.println(cat2.name);

    }

}
```

Доступ к статическому полю используя ссылку на объект

Доступ к статическому полю используя ссылку на объект

Доступ к статическому полю используя ссылку класс

Объяснение работы статических полей



Статические поля класса хранятся вместе с данными класса и существуют в единственном экземпляре. Как и методы класса.

Неявные преобразования выполняемые компилятором при работе со статическими полями

При компиляции обращение к статическому полю посредством ссылки заменяется на обращение посредством имени класса.

Конструкции вида

```
cat1.name = "Barsic";
```

Автоматически и во всем коде заменяются на конструкции вида

```
Cat.name = "Barsic";
```

По сути происходит замена ссылки на объект ссылкой на класс. Это же касается и вызова статических методов.

Статический метод класса

```
public class Cat {
     static String name;
     int weight;
     String color;

     public String meow() {
          return "meow meow";
     }

     public void jump() {
          System.out.println("hop");
     }

     public static void printCatHello() {
          System.out.println("Hello kitty");
     }

     public String toString() {
          return "Cat [name=" + name + ", weight=" + weight + ", color=" + color + "]";
     }
}
```

Статическое поле класса

Статический метод класса

Статический метод класса

```
public class Main {

    public static void main(String[] args) {

        Cat cat1 = new Cat();

        cat1.printCatHello();

        Cat.printCatHello();

    }

}
```

Доступ к статическому методу используя ссылку на объект

Доступ к статическому методу используя ссылку класс

Правила при работе со статическими полями и методами

Статическое поле существует в единственном экземпляре и хранится в MetaSpace вместе с данными класса. Значение этого поля будет одинаковым для всех объектов. При доступе к статическому полю ссылка на объект всегда заменяется ссылкой на класс.

Статический метод можно вызывать без создания объекта. Статический метод не имеет доступ к не статическим членам своего класса. При доступе к статическому методу ссылка на объект всегда заменяется ссылкой на класс.

Вопросы и ответы к ним

Как то сложновато все это. Обязательно знать это наизусть?

Желательно, но не обязательно. Достаточно просто запомнить следствия из этого.

Часто ли используют статические поля?

Нет. Статические поля используют редко. Они могут пригодиться для хранения глобального значения в проекте, но это не рекомендованная практика. Поэтому при применении статических полей стоит задуматься о корректности такого подхода. Исключение составляют константы.

Часто ли используют статические методы?

Довольно часто. Это прежде всего сервисные методы. Яркий тому пример классы Math, Collections, Arrays и статические методы описанные в них. Например Math.sqrt, Arrays.sort.

Ссылка this

Ссылка this может быть использована в теле не статического метода класса и только в нем. Это ссылка автоматически создается компилятором и указывает на текущий объект. Эта ссылка содержит тот же адрес, что и ссылка посредством которой данный метод был вызван.

Ссылка this используется для того, что бы метод мог ссылаться на объект для которого он был вызван.

Ссылка this

```
public class Cat {
    String name;
    int weight;
    String color;

    public void setName(String name) {
        this.name = name;
    }
    public String meow() {
        return "meow meow";
    }

    public void jump() {
        System.out.println("hop");
    }

    public String toString() {
        return "Cat [name=" + name + ", weight=" + weight + ", color=" + color + "]";
    }
}
```

Использование ссылки this в методе класса

Вызов метода в котором используется this

```
public class Main {

    public static void main(String[] args) {

        Cat cat1 = new Cat();

        cat1.setName("Vaska");

        System.out.println(cat1.name);
    }

}
```

Вызов метода в котором использована ссылка this

При таком вызове ссылке this в теле метода setName компилятор присваивает значение ссылки по которой этот метод вызван. Таким образом this == cat1, они содержат одинаковые адреса. И как следствие установка поля name по ссылке this, эквивалентно установке поля name по ссылке cat1.

Как происходит подобное присвоение?

Каждый не статический метод класса обладает одним дополнительным неявным параметром. Этот параметр всегда идет первым (внимание он неявный и поэтому в явном виде в списке параметров метода его нет), он и представляет собой ссылку this. Т.е. тип этого параметра ссылка тип которой совпадает с типом текущего класса, а название зарезервированное слово this.

```
public void setName(String name) {
    this.name = name;
}
```

⟺

```
public void setName(Cat this, String name) {
    this.name = name;
}
```

При вызове же компилятор фактически подставляет ссылку по которой произвели вызов метода в качестве этого первого параметра.

```
cat1.setName("Vaska");
```

⇨

```
cat1.setName(cat1,"Vaska");
```

Для чего используют ссылку this

Ссылка this чаще всего используется в следующих случаях:

1) Разрешения конфликта имен поля класса и формального параметра метода класса. В таком случае использование ссылки this явно указывает на поле класса;
2) Для реализации возможности методу добавить куда либо или вернуть ссылку на текущий объект;
3) Для вызова конструктора текущего класса.

Конструктор

Конструктор — специальный метод основное предназначение которого начальная установка значений полей объекта. Конструктор вызывается после работы оператора new при создании объекта.

Если в классе не описан конструктор в явном виде, то он автоматически генерируется компилятором. Если конструктор описан в явном виде в классе, то автоматически он не генерируется.

Правила описания конструктора:

- Имя конструктора должно совпадать с именем класса;
- У конструктора отсутствует тип возвращаемого значения;
- Конструктор не может иметь иных модификаторов, кроме модификаторов доступа. Он не может быть например native, final и т. д.

Конструктор не явный статический метод класса. Поэтому конструкторы можно вызывать из статических методов класса.

Пример вызова конструктора по умолчанию

Вызов конструктора по умолчанию

```
Cat cat1 = new Cat();
```

Конструктор по умолчанию — конструктор с модификатором public и без параметров. Именно такой генерируется компилятором с случае отсутствия явно описанного конструктора в классе. В классе Cat нет явно описанного конструктора, поэтому был создан конструктор по умолчанию и его вызов и приведен в примере.

Явное описание конструктора в классе

```
public class Cat {
    String name;
    int weight;
    String color;

    public Cat(String name, int weight, String color) {
        this.name = name;
        this.weight = weight;
        this.color = color;
    }
…...
```

Конструктор

Явное описание конструктора в классе. Данный конструктор предназначен для удобной инициализации всех полей класса Cat.

Внимание! При явном описании конструктора класса, конструктор по умолчанию больше не создается компилятором автоматически. И поэтому его вызов сопровождается ошибкой компиляции.

Описание нескольких конструкторов в классе

```
public class Cat {
    String name;
    int weight;
    String color;

    public Cat(String name, int weight, String color) {
        this.name = name;
        this.weight = weight;
        this.color = color;
    }

    public Cat() {

    }
…...
```

Конструктор с параметрами

Конструктор по умолчанию

Так как конструктор это метод, то его можно перегружать. В одном классе можно описать произвольное количество конструкторов.

Использование конструкторов

```
public class Main {

    public static void main(String[] args) {

        Cat cat1 = new Cat();
        cat1.name = "Vaska";
        cat1.color = "Black";
        cat1.weight = 4;

        Cat cat2 = new Cat("Umka", 5, "White");

        System.out.println(cat1);
        System.out.println(cat2);
    }
}
```

Использование конструктора по умолчанию

Использование конструктора с параметрами

Вопросы и ответы к ним

Использование конструктора с параметрами удобнее использования конструктора по умолчанию. Так стоит ли писать конструктор по умолчанию?

Для классов которые служат для описания сущностей с которыми вы будете работать это делать необходимо. Это упростит создание объектов этих типов разнообразными библиотеками и фреймворками.

Можно ли вызывать из одного конструктора другой?

Можно. Для этого используется синтаксис вида this(список параметров). Где список параметров совпадает со списком параметров конструктора который вы хотите вызвать.

Описание нескольких конструкторов в классе

```
public class Cat {
    String name;
    int weight;
    String color;

    public Cat(String name, int weight, String color) {

        this(name, weight);
        this.color = color;
    }

    public Cat(String name, int weight) {
        this.name = name;
        this.weight = weight;
    }

    public Cat() {

    }
…...
```

Вызов конструктора с двумя параметрами используя this

Конструктор с двумя параметрами

Порядок описания членов класса

Согласно стандартам оформления кода на Java[3] следует описывать члены класса в следующем порядке:

1) Статические поля класса. Порядок описания определяется модификатором доступа public, protected, private.
2) Не статические поля класса. Порядок описания определяется модификатором доступа public, protected, private.
3) Конструкторы класса.
4) Методы. Методы стоит группировать по функциональности. Если один метод класса вызывает другой метод класса, то желательно описать их рядом.

Использование ссылки пользовательского типа

Работа со ссылками пользовательского типа не отличается от работы со ссылками встроенных типов данных (например String).

```
public static String getCatName(Cat cat) {
    return cat.name;
}

public static void renameCat(Cat cat, String newName) {
    cat.name = newName;
}

public static Cat createWhiteCat(String name, int weight) {
    Cat cat = new Cat(name, weight, "white");
return cat;
}
```

Ссылка типа Cat как параметр метода

Ссылка типа Cat как тип возвращаемого значения

Пример объявления массива ссылок пользовательского типа

```
Cat [] cats = new Cat[10];
```

Как и все массивы ссылочного типа, массив ссылок пользовательского типа инициализируется значениями null.

Сокрытие

Сокрытие - принцип проектирования, заключающийся в разграничении доступа различных частей программы к внутренним компонентам друг друга. В Java является составной частью принципа инкапсуляции. Для реализации сокрытия членов класса от внешнего доступа используется модификатор private.

Промышленным стандартом в Java стало сокрытие всех полей класса. Т.е. все поля класса объявляются с модификатором private. Для каждого поля пишется два метода — метод получения и установки. Метод получения возвращает значение private поля. Метод установки задает новое значение private поля.

Методы получения и установки могут также называться

- Академический вариант. Акцессоры(accessors) - методы получения, мутаторы (mutators) - методы установки.
- Сленговое название. Геттеры (getters) - методы получения, сеттеры (setters) - методы установки.

Правило описания метода получения

Метод получения желательно описывать согласно установленному соглашению. Имя метода должно начинаться с приставки get (для полей типа boolean is) после которого должно быть имя поля записанное с большой буквы. Тип возвращаемого значения совпадает с типом используемого поля. Модификатор доступа public.

Метод получения для поля name.

```
public String getName() {
    return name;
}
```

Правило описания метода установки

Метод установки желательно описывать согласно установленному соглашению. Имя метода должно начинаться с приставки set после которого должно быть имя поля записанное с большой буквы. Параметр метода должен совпадать и по имени и по типу с нужным полем. Тип возвращаемого значения void. Модификатор доступа public.

Напишем метод установки для поля name.

```
public void setName(String name) {
    this.name = name;
}
```

Свойство

Свойство - способ доступа к внутреннему состоянию объекта, имитирующий переменную некоторого типа. Обращение к свойству объекта выглядит так же, как и обращение к полю, но, в действительности, реализовано через вызов функции. При попытке задать значение данного свойства вызывается один метод, а при попытке получить значение данного свойства — другой.

В Java свойства не реализованы. Но их общепринятой заметой является совокупность из private поля и метода получения и установки для него. Поэтому если при чтении технической документации вы встречаетесь с термином свойство, то это именно оно.

Пример полного кода класса Cat

```
public class Cat {
        private String name;
        private int weight;
        private String color;

        public Cat(String name, int weight, String color) {
                this.name = name;
                this.weight = weight;
                this.color = color;
        }

        public Cat() {
        }

        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public int getWeight() {
                return weight;
        }
        public void setWeight(int weight) {
                this.weight = weight;
        }
        public String getColor() {
                return color;
        }
        public void setColor(String color) {
                this.color = color;
        }
        public String meow() {
                return "meow meow";
        }
        public void jump() {
                System.out.println("hop");
        }
        public String toString() {
                return "Cat [name=" + name + ", weight=" + weight + ", color=" + color + "]";
        }
}
```

Поля

Конструкторы

Методы получения и установки

Методы логики класса

Метод для получения состояния объекта

Использование объектов типа Cat

```
public class Main {

    public static void main(String[] args) {

        Cat cat1 = new Cat("Umka", 5, "White");

        System.out.println(cat1.getName()); ← Использование метода получения

        cat1.setName("Luska"); ← Использование метода установки

        System.out.println(cat1); ← Использование метода toString()

    }
}
```

Задание для самостоятельной проработки. Основной уровень.

1) Создайте пользовательский класс для описания товара (предположим, это задел для интернет-магазина). В качестве свойств товара можете использовать значение цены, описание, вес товара. Создайте пару экземпляров вашего класса и протестируйте их работу.
2) Описать класс Треугольник. В качестве свойств возьмите длины сторон треугольника. Реализуйте метод, который будет возвращать площадь этого треугольника. Создайте несколько объектов этого класса и протестируйте их.

Задание для самостоятельной проработки. Продвинутый уровень.

1) Создайте класс Phone (Телефон) одним из свойств должен быть его номер. Создайте класс Network (сеть мобильного оператора). В классе Телефон должны быть описаны следующие методы:
   - Регистрация в сети мобильного оператора
   - Метод реализующий исходящий звонок. Данный метод принимает один параметр (описывающий номер мобильного телефона). Логика работы этого метода такова: если текущий телефон не прошел регистрацию в сети, то закончить работу метода с сообщением об этом. Если текущий телефон прошел регистрацию и в сети также зарегистрирован телефон на номер которого совершается вызов, то вызвать метод входящий звонок у того телефона. Если телефон на номер которого вы совершаете вызов в сети не зарегистрирован, то закончить работу метода с сообщением об этом.
   - Метод реализующий входящий звонок. Принимает параметр в виде номера с которого произвели вызов на текущий. Вывести сообщение вида вам звонит номер такой то.

Список литературы

1) James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith «The Java ® Language Specification Java SE 11 Edition» 2018, Oracle America, Inc.
2) Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley, Daniel Smith «The Java ® Virtual Machine Specification Java SE 11 Edition» 2018, Oracle America, Inc.
3) «Java Code Conventions» 1997, Sun Microsystems, Inc.
4) Брюс Эккель. Философия Java. Библиотека программирования. 4-е издание. Спб::Питер 2009.
5) Герберт Шилд. Java 8. Полное руководство; 9-е изд.:Пер. с англ. - М.: ООО "И.Д. Вильямс" 2015.