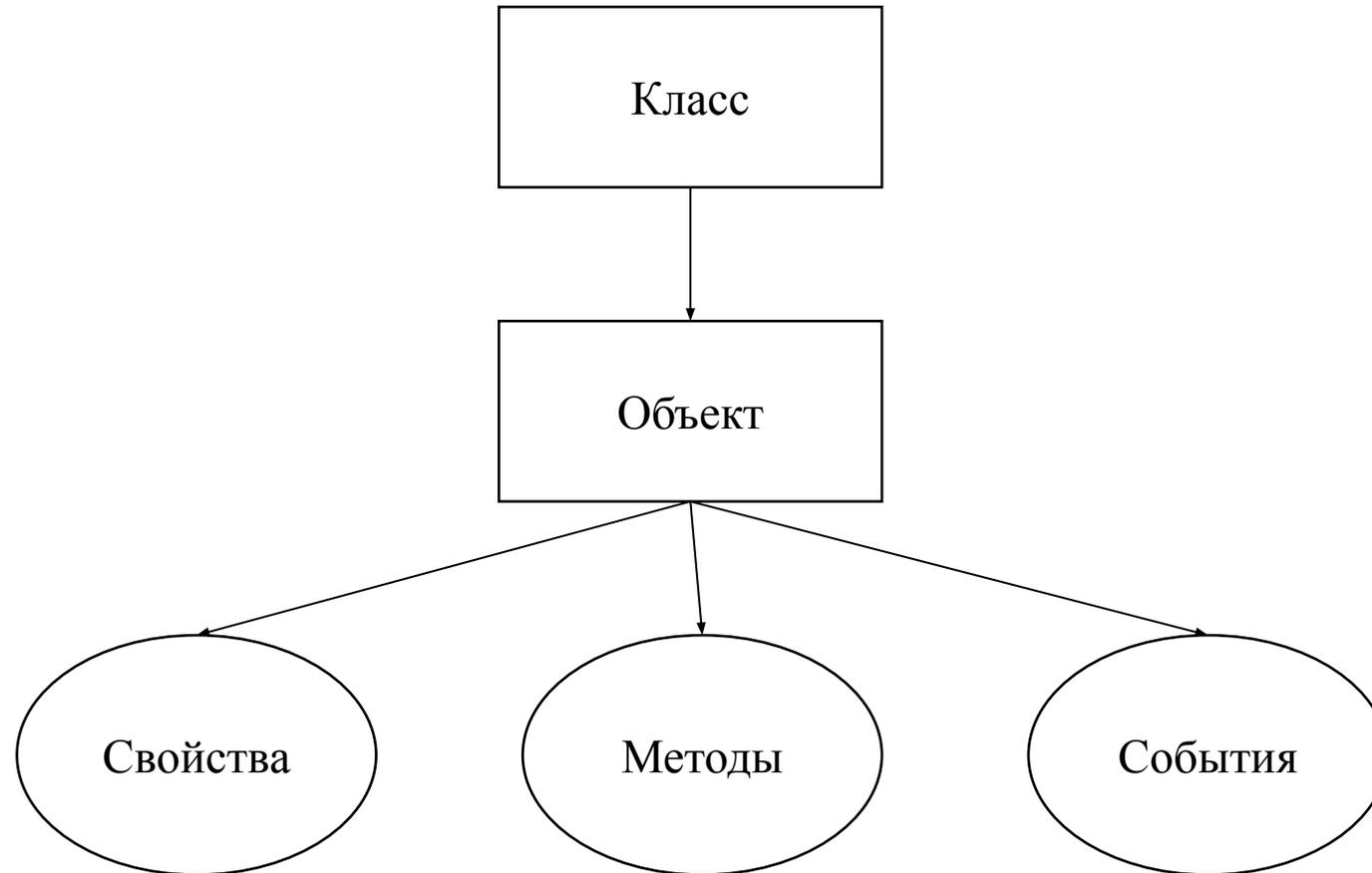


Объектно-ориентированное программирование в JAVA

Объектно-ориентированное программирование

- ООП – это парадигма программирования, в которой базовым является понятие объекта
 - Объект имеет:
 - Состояние
 - Поведение
 - Уникальность
 - Объект умеет:
 - Получать сообщения
 - Обработать данные
 - Отправлять сообщения
- Программа в ходе работы представляет собой набор взаимодействующих объектов

Объектно-ориентированное программирование



Ссылка на материал для дополнительного изучения

Принципы ООП

- Инкапсуляция — это свойство системы, позволяющее объединить данные и методы в классе, и скрыть детали реализации от пользователя.

```
class A {  
    private int a;  
    private int b;  
  
    private void doSomething() { //скрытый метод  
        //actions  
    }  
  
    public int getSomething() { //открытый метод  
        return a;  
    }  
}
```

[Ссылка на материал для дополнительного изучения](#)

Принципы ООП

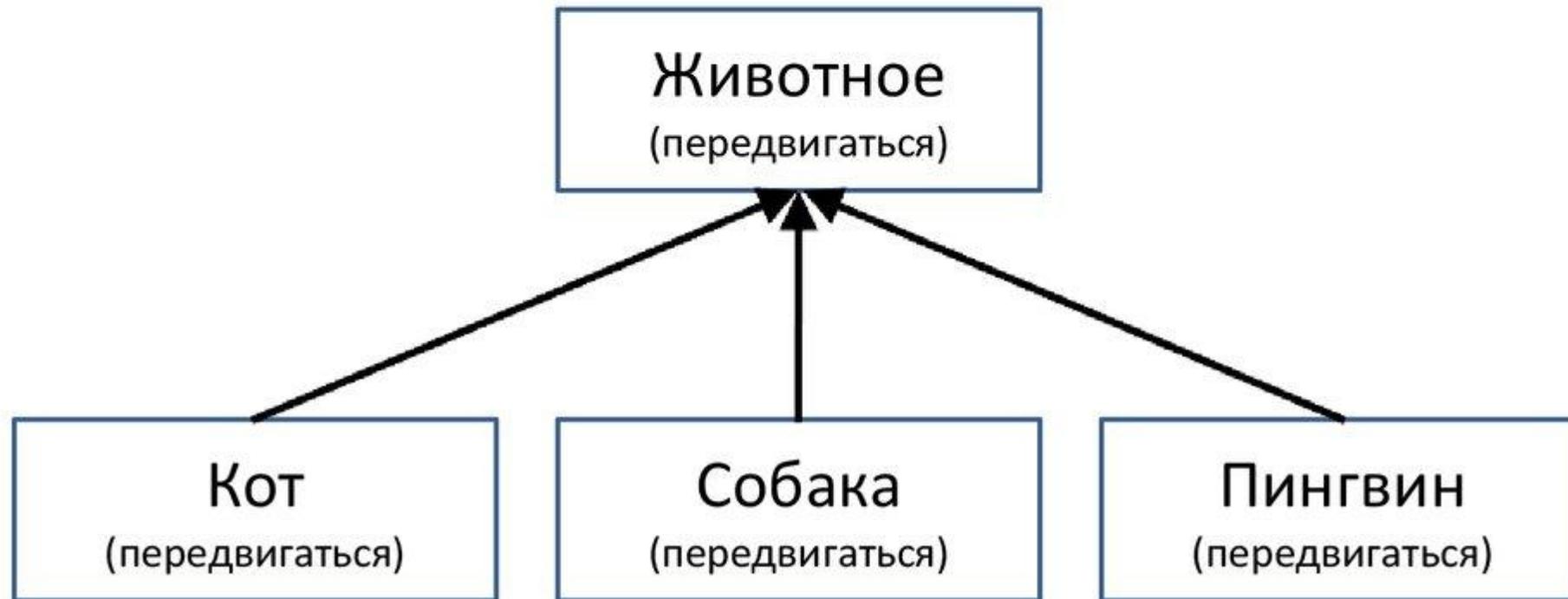
- Наследование – механизм создания новых классов на основе существующих
- При наследовании дочернему классу передаются поля и методы родительского класса

```
class Transport {  
    //  
}
```

```
class Car extends Transport {  
    //  
}
```

Принципы ООП

- Полиморфизм – возможность использования одних и тех же методов для объектов разных классов, только реализация этих методов будет индивидуальной для каждого класса



Классы и объекты в JAVA

- Класс - это шаблон для создания объекта.
- Он определяет структуру и поведение, которые будут совместно использоваться набором объектов.
- Он содержит переменные и методы, которые называются элементами класса, членами класса.

```
public class Название {  
    // описание класса  
}
```

**Ссылка на материал
для дополнительного изучения**

Хорстман К. «JAVA Библиотека профессионалов Том 1. Основы Одиннадцатое издание» Глава 4.1.1

```
public class Home  
{  
      
    ТЕЛО КЛАССА  
}
```

```
public class Home  
{  
      
    ПЕРЕМЕННАЯ A  
  
      
    ПЕРЕМЕННАЯ Z  
  
      
    МЕТОД 1  
  
      
    МЕТОД N  
}
```

Классы и объекты в JAVA

Класс может содержать:

- Поля
- Методы
- Вложенные классы и интерфейсы

```
public class Body {  
    public long idNum;  
    public String name;  
    public Body orbits;  
  
    public static long nextID = 0;  
}
```

Классы и объекты в JAVA

Модификаторы объявления класса

`public`

Признак общедоступности класса (класс виден вне пакета)

`abstract`

Признак абстрактности класса (класс не полностью реализует поведение)

`final`

Завершенность класса (класс не допускает наследования)

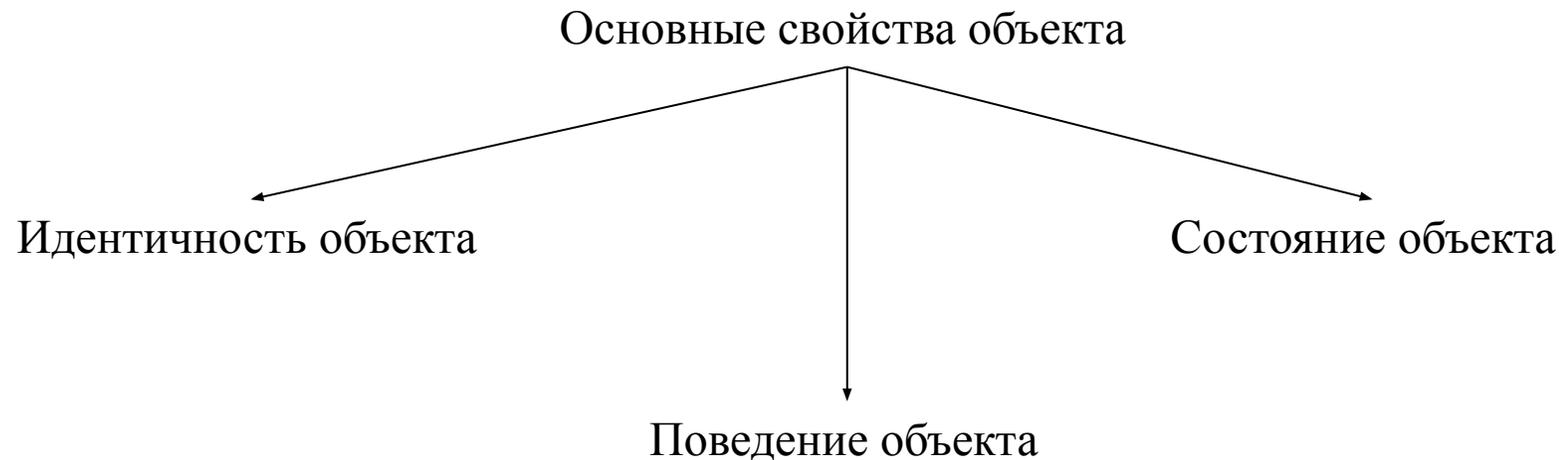
`strictfp`

Повышенные требования к операциям с плавающей точкой

(результаты операций одинаковые на различных платформах)

Классы и объекты в JAVA

- Объект класса - это область памяти, которая содержит переменные, объявленные в классе (поля класса)



**Ссылка на материал
для дополнительного изучения**

Хорстман К. «JAVA Библиотека профессионалов Том 1.
Основы Одиннадцатое издание» Глава 4.1.2

Классы и объекты в JAVA

- Создание объектов класса представляет из себя двухэтапный процесс:

1. Объявление переменной типа класса:

```
Box = myBox;
```

2. Создание объекта:

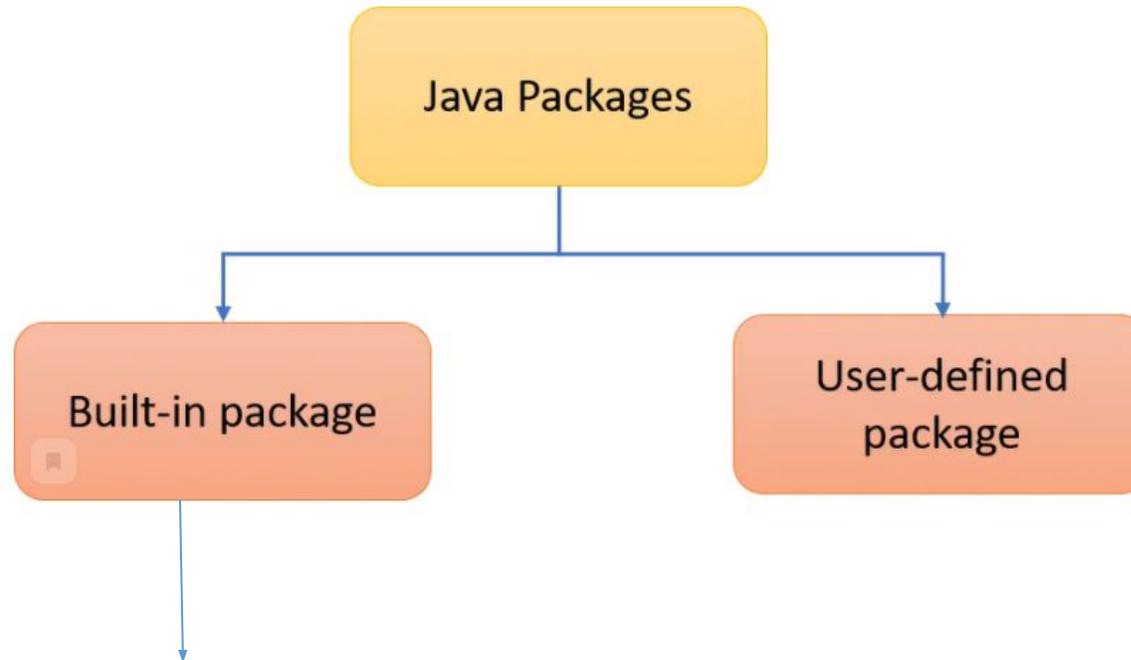
```
myBox = new Box();
```

Пакеты

- Пакеты в Java группируют несколько классов, интерфейсов или пакетов и т. д.
- Преимущества пакетов:
 - Избегает конфликтов именования классов. Это означает, что мы можем использовать одни и те же имена классов в двух разных пакетах
 - Обеспечивает возможность повторного использования путем доступа к классу из одного пакета в другом
 - Простота обслуживания, так как классы будут организованы
 - Обеспечивает защиту доступа для защищенных классов и классов по умолчанию
 - Это помогает в инкапсуляции или сокрытии данных

Ссылка на материал для дополнительного изучения

Пакеты



- java.lang - языковая поддержка
- java.io - операции ввода / вывода
- java.util - служебные классы для реализации структур данных
- java.net - сетевые операции
- java.awt - реализация графического пользовательского интерфейса
- java.applet - создание апплетов

Конструкторы

- Конструктор можно вызвать только в сочетании с операцией `new()`. Его нельзя применить к существующему объекту, чтобы изменить информацию в его полях
- Имя конструктора совпадает с именем класса
- Класс может иметь несколько конструкторов
- Конструктор может иметь один или несколько параметров или же вообще их не иметь
- Конструктор не возвращает никакого значения

! Нельзя присваивать локальным переменным такие же имена, как и полям экземпляра

Ссылка на материал для дополнительного изучения

Ключевое слово var

До версии Java 10 объявление переменных:

```
Employee harry = new Employee("Harry Hacker", 50000, 1989, 10, 1);
```

Начиная с версии Java 10:

```
var harry = new Employee("Harry Hacker", 50000, 1989, 10, 1);
```

- Ключевое слово var можно употреблять в локальных переменных, объявляемых **ТОЛЬКО** в теле метода

[Ссылка на материал для дополнительного изучения](#)

Обработка пустых ссылок на объект

Используется метод `requireNonNullElse()`

```
public Employee(String n, double s, int year, int month, int day)
{
    name = Objects.requireNonNullElse(n, "unknown");
    . . .
}
```

«Жесткий» способ – отвержение аргумента с пустым значением `null`

```
public Employee(String n, double s, int year, int month, int day)
{
    Objects.requireNonNull(n, "The name cannot be null");

    name = n;
    . . .
}
```

[Ссылка на материал для дополнительного изучения](#)

Явные и неявные параметры

Пример:

```
public void raiseSalary(double byPercent)
{
    double raise = salary * byPercent / 100;
    salary += raise;
}
```

У метода `raiseSalary()` имеются два параметра. Первый, называемый **неявным**, представляет собой ссылку на объект типа `Employee`, который указывается перед именем метода. Второй параметр называется **явным** и указывается как число в скобках после имени данного метода.

Явные параметры перечисляются в объявлении метода, например `double byPercent`

Ссылка на материал для дополнительного изучения

Статические поля. Модификатор static

Поле с модификатором доступа static существует в одном экземпляре, для всего класса. Но если поле не статическое, то каждый объект содержит его копию

Пример:

```
class Employee
{
    ...
    private int id;

    private static int nextId = 1;
}
```

[Ссылка на материал для дополнительного изучения](#)

Статические методы

Статические методы – методы, которые не оперируют объектами.

Применение:

- когда методу не требуется доступ к данным о состоянии объекта, поскольку все необходимые параметры задаются явно
- когда методу требуется доступ лишь к статическим полям класса

```
public static int getNextId()  
{  
    return nextId; // вернуть статическое поле  
}
```

[Ссылка на материал для дополнительного изучения](#)

Метод main()

Метод main() объявляется как статический:

```
public class Application
{
    public static void main(String[] args)
    {
        // здесь создаются объекты
    }
}
```

Метод main() не оперирует никакими объектами. На самом деле при запуске программы еще нет никаких объектов. Статический метод main() выполняет и конструирует объекты, необходимые программе

[Ссылка на материал для дополнительного изучения](#)

Задание

Создайте пример наследования, реализуйте класс `Student` и класс `Aspirant`, аспирант отличается от студента наличием некой научной работы.

Класс `Student` содержит переменные: `String firstName`, `lastName`, `group`. А также `double averageMark`, содержащую среднюю оценку.

Создать переменную типа `Student`, которая ссылается на объект типа `Aspirant`.

Создать метод `getScholarship()` для класса `Student`, который возвращает сумму стипендии. Если средняя оценка студента равна 5, то сумма 100, иначе 80.

Переопределить этот метод в классе `Aspirant`. Если средняя оценка аспиранта равна 5, то сумма 200, иначе 180.

Создать массив типа `Student`, содержащий объекты класса `Student` и `Aspirant`.

Вызвать метод `getScholarship()` для каждого элемента массива.

Задание

Создать класс `Animal` и расширяющие его классы `Dog`, `Cat`, `Horse`. Класс `Animal` содержит переменные `food`, `location` и методы `makeNoise`, `eat`, `sleep`. Метод `makeNoise`, например, может выводить на консоль "Такое-то животное спит". `Dog`, `Cat`, `Horse` переопределяют методы `makeNoise`, `eat`. Добавьте переменные в классы `Dog`, `Cat`, `Horse`, характеризующие только этих животных. Создайте класс `Ветеринар`, в котором определите метод `void treatAnimal(Animal animal)`. Пусть этот метод распечатывает `food` и `location` пришедшего на прием животного. В методе `main` создайте массив типа `Animal`, в который запишите животных всех имеющихся у вас типов. В цикле отправляйте их на прием к ветеринару.

Задание

Создайте класс Phone, который содержит переменные number, model и weight.

Создайте три экземпляра этого класса.

Выведите на консоль значения их переменных.

Добавить в класс Phone методы: receiveCall, имеет один параметр – имя звонящего. Выводит на консоль сообщение “Звонит {name}”. getNumber – возвращает номер телефона. Вызвать эти методы для каждого из объектов.

Добавить конструктор в класс Phone, который принимает на вход три параметра для инициализации переменных класса - number, model и weight.

Добавить конструктор, который принимает на вход два параметра для инициализации переменных класса - number, model.

Добавить конструктор без параметров.

Вызвать из конструктора с тремя параметрами конструктор с двумя.

Добавьте перегруженный метод receiveCall, который принимает два параметра - имя звонящего и номер телефона звонящего. Вызвать этот метод.

Создать метод sendMessage с аргументами переменной длины. Данный метод принимает на вход номера телефонов, которым будет отправлено сообщение. Метод выводит на консоль номера этих телефонов.