Оглавление

[Пример создания класса и его экземпляра 2](#_Toc56941222)

[✡  НАСЛЕДОВАНИЕ✡ 3](#_Toc56941223)

[ПЕРЕОПРЕДЕЛЕНИЕ 4](#_Toc56941224)

[ПЕРЕГРУЗКА 5](#_Toc56941225)

[ПАКЕТЫ 5](#_Toc56941226)

[АБСТРАКТНЫЕ КЛАССЫ 6](#_Toc56941227)

[ВЛОЖЕННЫЕ КЛАССЫ 6](#_Toc56941228)

[АНОНИМНЫЕ КЛАССЫ 8](#_Toc56941229)

[Абстрактные классы 9](#_Toc56941230)

[Интерфейсы 10](#_Toc56941231)

# Пример создания класса и его экземпляра

* Пример:

class Cat {

int claws\_lenght;

int age;

}

Cat barsic = new Cat();

barsic.age=15;

barsic.claws\_lenght = 0.4

}

Пример конструктора:

конструктор создается внутри класса, должен называться также, как и класс в одном классе может быть много конструкторов

public class Noone{

int height;

float weight;

Noone (int a, float b){

this.height = a;

this.weight = b;

}

Noone (){}

}

Великолепно, теперь используем конструктор:

Noone crip = new Noone(44, 21);

(barsic instanceof Cat) == true

Если бы Cat наследовал Animal, то

(barsic instanceof Animal) == true

# ✡  НАСЛЕДОВАНИЕ✡

Если я ставлю модификатор final, то этот класс нельзя наследовать.

Class Cat extends Animal {

/тело класса

}

В таком случае Cat “наследует” переменные, методы и конструкторы у класса Animal.

super – инструмент для работы с методами и конструкторами суперкласса, а еще обращения к его полям

public Student (int a, int b, int course) {

super (int a, int b);

this.course = course;

}

Можем так сделать, только если мы создали в главном классе конструктор, который принимает 2 переменные.

Если хотим из суперкласса вызвать метод, пишем

super.method\_name();

# ПЕРЕОПРЕДЕЛЕНИЕ

Общие слова: метод выглядит вот так:

Тип\_возвращения имя (принимаемые значения)// Если мы меняем что – либо, то мы создаем новый метод в случае, если мы не меняем имя, то мы тоже создаем новый метод, реализуя механизм перегрузки.

* Переопределить статический метод нельзя.
* Переопределенный метод также обязан быть не static.
* Если метод имеет модификатор final, его нельзя переопределить.
* При переопределении желательно писать @Override

У нас есть класс Animal, имеющий следующий метод:

void say\_sound() {

System.out.print(“I don’t know what sound I need to do”);

}

В классе Cat, который наследует Animal, пишем:

@Override

void say\_sound(){

System.out.print(“mew”);

}

((@Override писать не обязательно, но если написать, то компилятор тебя проверит (а вдруг ты метод назвал не совсем так, и теперь это новый метод)))

* Нельзя переписать приватный метод.
* Переопределяя protected метод можем поставить модификатор доступа protected или public.
* Переопределяя public метод обязательно ставим модификатор public.

Если Student extends Human, то можно написать

Human john = new Student ();

Отличие от Human john = new Human(); лишь в том, что наш объект теперь имеет переопределенные методы.

# ПЕРЕГРУЗКА

У нас есть метод sum:

public static int sum(int a, int b){

return a + b;

}

Неплохо, но метод работает только для 2-х вводных.

А если мы хотим 3, нам новую создавать? Нет.

public static int sum(int a, int b, int c) {

return a + b + c;

}

public static float sum(float a, float b) {

return a + b;

}

# ПАКЕТЫ

Если мы создаем много классов, и, соответственно, много файлов, то:

1) В исходном коде каждого файла пишем package 112211,

2) Складываем их в папку 112211,

3)В cmd идешь в директорию, где лежит эта папка

4) Пишешь javac 112211\lab0.java\*

5) Пишешь java 112211\lab0\*

\*lab0.java – файл с методом main

\*\*обычно пишем javac lab0.java|||| java lab0

Чтобы получить jar пишем jar cvf папка\_нейм.jar Main.class

Или jar -ecf Main Main.jar Main.class

Для запуска пишем java -jar Main.jar

# АБСТРАКТНЫЕ КЛАССЫ

public abstract class Thing {

abstract public void do\_sth() {}

}

Если в классе есть абстрактные методы, то и класс должен быть абстрактным.

Теперь все классы, наследующие Thing обязаны будут переопределить метод do\_sth

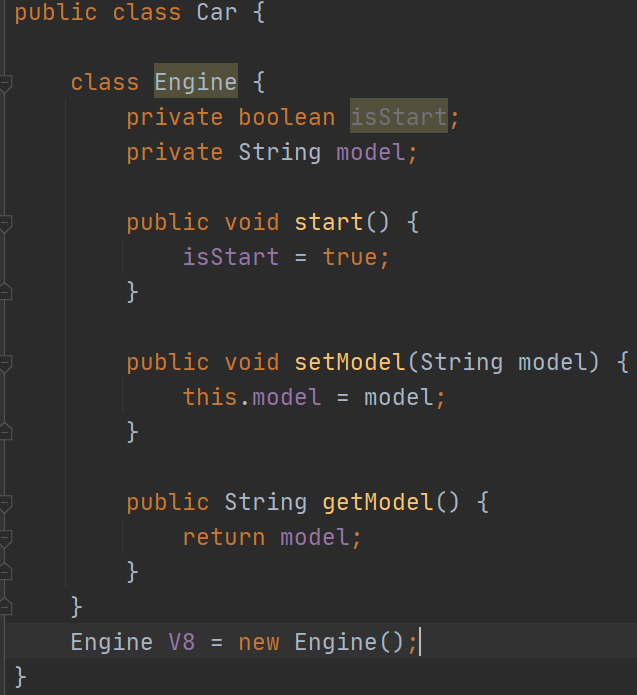
Создать экземпляр абстрактного класса нельзя, максимум можно так:

Thing s1garetka = new Sigareta();

# ВЛОЖЕННЫЕ КЛАССЫ

То, что мы делаем с классом внутри него – это его дело.

Так вот, можем сделать внутренний класс, потом (уже в основном классе) создать экземпляр, и теперь это ещё одно поле в нашем классе.



Вложенный и внутренний класс видят private поля друг друга.

Есть 3 типа вложенных классов:

1. Нестатический вложенный класс (как правило приватны). Данные классы относятся к объекту, т. е. имеют доступ к его полям. Пример – кусочки машины. Могу создать экземпляр такого класса внутри метода, где вызову его метод.
2. Статический вложенный класс (как правило публичны). Имеет доступ только к статическим полям. Создаем объект следующим образом :

Car.Battery battery = new Car.Battery();

1. Вложенный класс в методе. Если мы сделаем переменную метода (локальную) final, то у нас будет доступ к ней в этом классе.

# АНОНИМНЫЕ КЛАССЫ

Использование анонимных классов оправдано в следующих случаях:

1. Тело класса очень короткое
2. Нужен всего один экземпляр класса
3. Класс используется в месте его создания или сразу после него
4. Имя класса не важно или не облегчает понимание кода.

Создание анонимного класса:

new Tank {

\*тело класса\*

}; // точка с запятой строго обязательны

* **Пример**

new Computer() {

void superHunter() {

this.i7.start(); //типо вместо названия //экземпляра говоришь “этот”  
};

* **Пример**

Вот есть у меня класс Tracer, но мне не нравиться реализация одного метода, что я делаю?

Ближе к месту вызова класса я пишу

Tracer tracer = new Tracer() { // заметь! «{» вместо «;»

\*переопределяю метод и вообще пишу код\*

};

//теперь можем вызвать

tracer.eat();

по факту новый класс (который наследует за Tracer) всё же создается, но мы этого не видим.

* **Пример**

Похожая ситуация – у нас есть интерфейс, и мы хотим его реализовать 1 раз, нам для этого создавать объект? Нет.

AbleToEat ableToEat = new AbleToEat() {

@Override

public void eat(){ \*ну тут конкретная реализация\*}

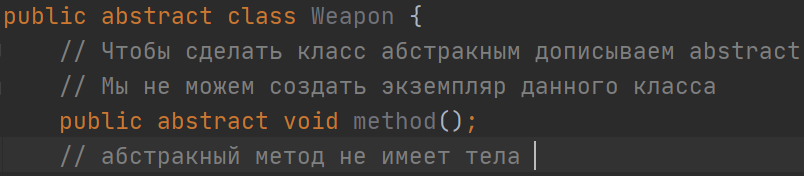
};

ableToEat.eat();

# Абстрактные классы

С помощью абстрактного класса мы **обязываем** наследников реализовать какие – либо методы (абстрактные). Можем еще и обычный метод запилить, тогда вообще всё круто.

В остальном абстрактный класс ничем не отличается от обычного.



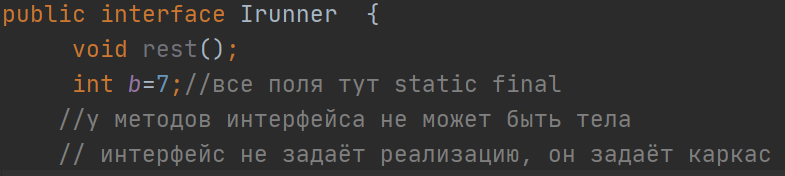
# Интерфейсы

В интерфейсах все методы public abstract, а все поля public static final (и это нельзя изменить).

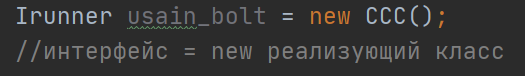
В жабе мы можем реализовывать множество интерфейсов:



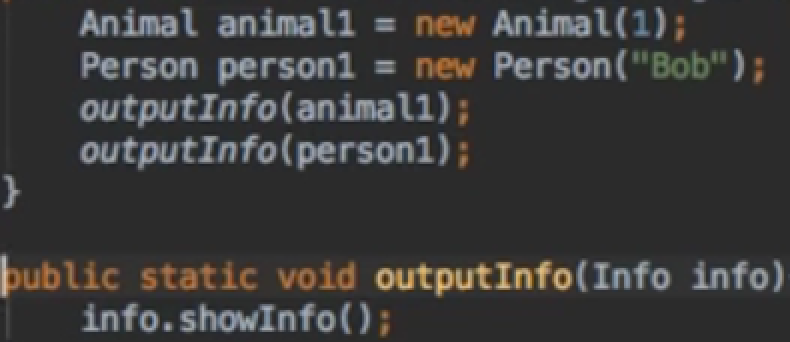
Пример интерфейса:



Могу сделать вот так:



Могу сделать метод, который будет принимать на вход любой объект, реализующий интерфейс:



Если хотим и унаследовать и реализовать:



МОЖЕМ:

1. Мы можем создать много классов (причем не обязательно вложенных) в одном java файле, но только один из них может быть public.

При задании полей класса могу сделать следующее:

public class Bus {

public Strint some\_string;

public Engine V8;

}// В данном случае мы создали экземпляр класса Engine;

Метод класса всегда имеет доступ к его полям

Везде, где используется класс, может быть использован и его наследник.