Лекция 3. Описание классов, модификаторы доступа.

NetCracker®

Имена используются в программе для доступа к объявленным ранее элементам языка. Имена имеют:

- пакеты;
- классы (конструкторы);
- интерфейсы;
- элементы ссылочных типов:
  - поля;
  - методы;
  - внутренние классы и интерфейсы;
- аргументы:
  - методов;
  - конструкторов;
  - обработчиков ошибок;
- локальные переменные.

Пакеты в Java - это способ логически группировать классы.

Пакеты:

- классы,
- интерфейсы,
- вложенные пакеты.

Имена:

- простые , состоят из одного идентификатора (они определяются во время объявления)
- составные , состоят из последовательности идентификаторов, разделенных точкой.

Составное имя пакета = Полное имя пакета, в котором он располагается . Собственное простое имя

(например, java.lang, java.lang.reflect)

Для ссылочных типов, где элементами являются поля и методы, а также внутренние типы (классы и интерфейсы):

Составное имя = Простое/Составное имя типа или переменной объектного типа . Имя переменной

(например, java.lang.Math.PI)

Модуль компиляции хранится в текстовом .java-файле состоит из трех частей:

- объявление пакета;
- import-выражения;
- объявления верхнего уровня.

Объявление пакета (напр.,java/lang/Object.java):

```
package java.lang;
```

Область видимости объявления типа - пакет, в котором он располагается. Внутри этого пакета допускается обращение к типу по его простому имени. Из всех других пакетов необходимо обращаться по составному имени (полное имя пакета плюс простое имя типа, разделенные точкой), либо с помощью импортирующих выражений.

Import-выражения позволяют импортировать типы в модуль компиляции и далее обращаться к ним по простым именам.

Существует два вида выражений:

- импорт одного типа;
- импорт пакета.

- При указании короткого имени (без указания пакета) компилятор ищет классы:

1. Сначала среди явно импортированых классов:
   ```
   import java.util.List;
   ```
2. Потом среди явно статически импортированных членов класса:
   ```
   import static java.awt.Color.WHITE;
   ```
3. Потом среди классов, объявленых в текущем пакете.
4. Потом среди классов, расположеных в импортированых пакетах:
   ```
   import java.io.*;
   ```
5. Потом среди статических членов классов, которые были импортированы:
   ```
   import static java.lang.Math.*;
   ```



```
[модификаторы] class SomeClass
   [extends ParentClass]
   [implements Interface1, Interface2] {
```

- внутренние классы
- внутренние интерфейсы
- **поля**
- **методы**
- **конструкторы**
- статический инициализатор
- динамический инициализатор

```
}
```


В Java модификаторы доступа указываются для:

- типов (классов и интерфейсов) объявления верхнего уровня;
- элементов ссылочных типов (полей, методов, внутренних типов);
- конструкторов классов.

Модификаторы доступа возможны для различных элементов языка:

- Пакеты всегда доступны, поэтому у них нет модификаторов доступа, любой существующий в системе пакет может быть использован из любой точки программы.
- Типы (классы и интерфейсы) верхнего уровня объявления. При их объявлении есть

всего две возможности: указать модификатор public или не указывать его.

- Массив имеет тот же уровень доступа, что и тип, на основе которого он объявлен.
- Элементы типов и конструкторы объектных типов обладают всеми возможными значениями уровня доступа. Все элементы интерфейсов являются public.

Уровни доступа:

- public;
- private;
- protected;
- <default>.

- **public** или <default>

  Модификатор доступа к классу

- **abstract**

  Признак абстрактности класса

- **final**

  Класс не допускает наследование

- **strictfp**

  Java использует strictfp для гарантирования неизменности результатов операций с плавающей точкой на всех платформах.

- *public*. *К*ласс с признаком *общедоступности.* В любом коде позволяется объявлять ссылки на объекты класса и обращаться к его членам, отмеченным как public. Если модификатор public не задан, класс будет доступен только в контексте *пакета,* которому принадлежит.
- *abstract*. Создавать экземпляры такого класса запрещено. Класс неполный, с наличием в его объявлении *абстрактных* методов (с модификатором abstract), которые должны быть реализованы в производных классах.
- *final*. Класс, определенный как final, *не* допускает наследования.
- *strict floating point*. Операции с плавающей запятой, предусмотренные методами-членами класса, должны выполняться *точно* и *единообразно* всеми виртуальными машинами Java.

Объявление поля:

[модификаторы] <тип> <имя>[=<значение>] {, <имя>[=<значение>]}* ;

double sum = 2.5 + 3.7;

private int a, b, c = 5, d;

Модификаторы полей:

- Модификатор видимости (public, private, protected, <default>)
- static - статические поля , являются общими для всех объектов класса и называются переменными класса.
- final - неизменяемые поля
- transient - несохраняемые поля
- volatile - многопоточные поля

Объявление метода:

```
[модификаторы] <тип> <имя>(<тип> <имя> [, <тип> <имя>]*)
   [throws Exception1, Exception2] {
   <тело метода>
}
```

Модификаторы:

- *Модификатор видимости*
- `abstract` - *абстрактный метод*
- `static` - *статический метод*
- `final` - *непереопределяемый метод*
- `synchronized` - *безопасный метод при работе с потоками*
- `native` - *метод написанный на нативном коде*
- `strictfp` - *стандартизированные вычисления с плавающей точкой*

Модификаторы:

abstract. Предусматривает только объявление метода. Метод должен быть определен в классах-наследниках .

static. Метод может обращаться и изменять статические поля класса, но не может обращаться к полям – членам класса, не описанным как ***static***. К таким методам можно обращаться через имя класса, не создавая экземпляр класса.

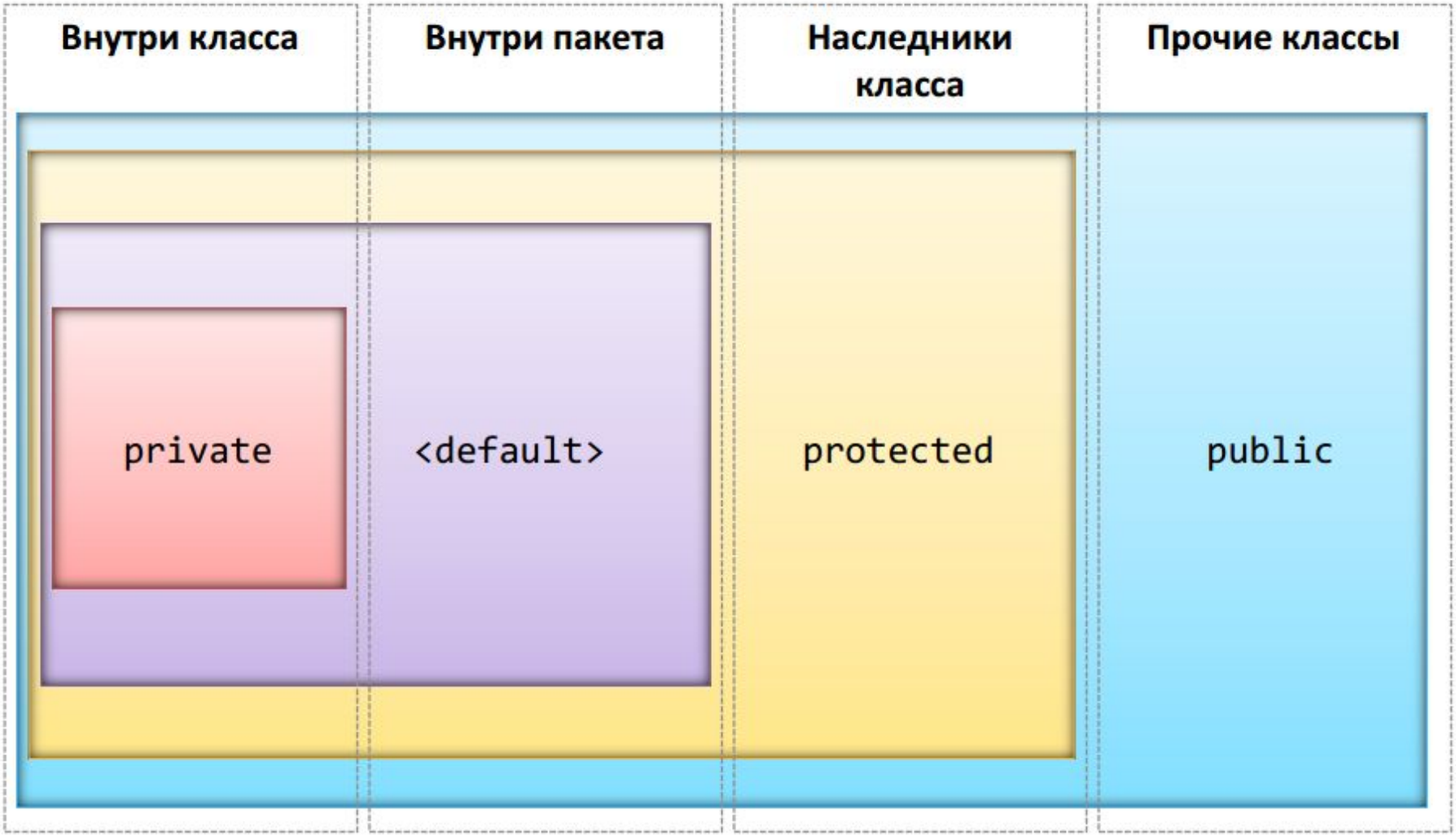
final. Метод нельзя переопределять в наследниках. Можно считать, что все методы final-класса, а также все private-методы любого класса являются final.

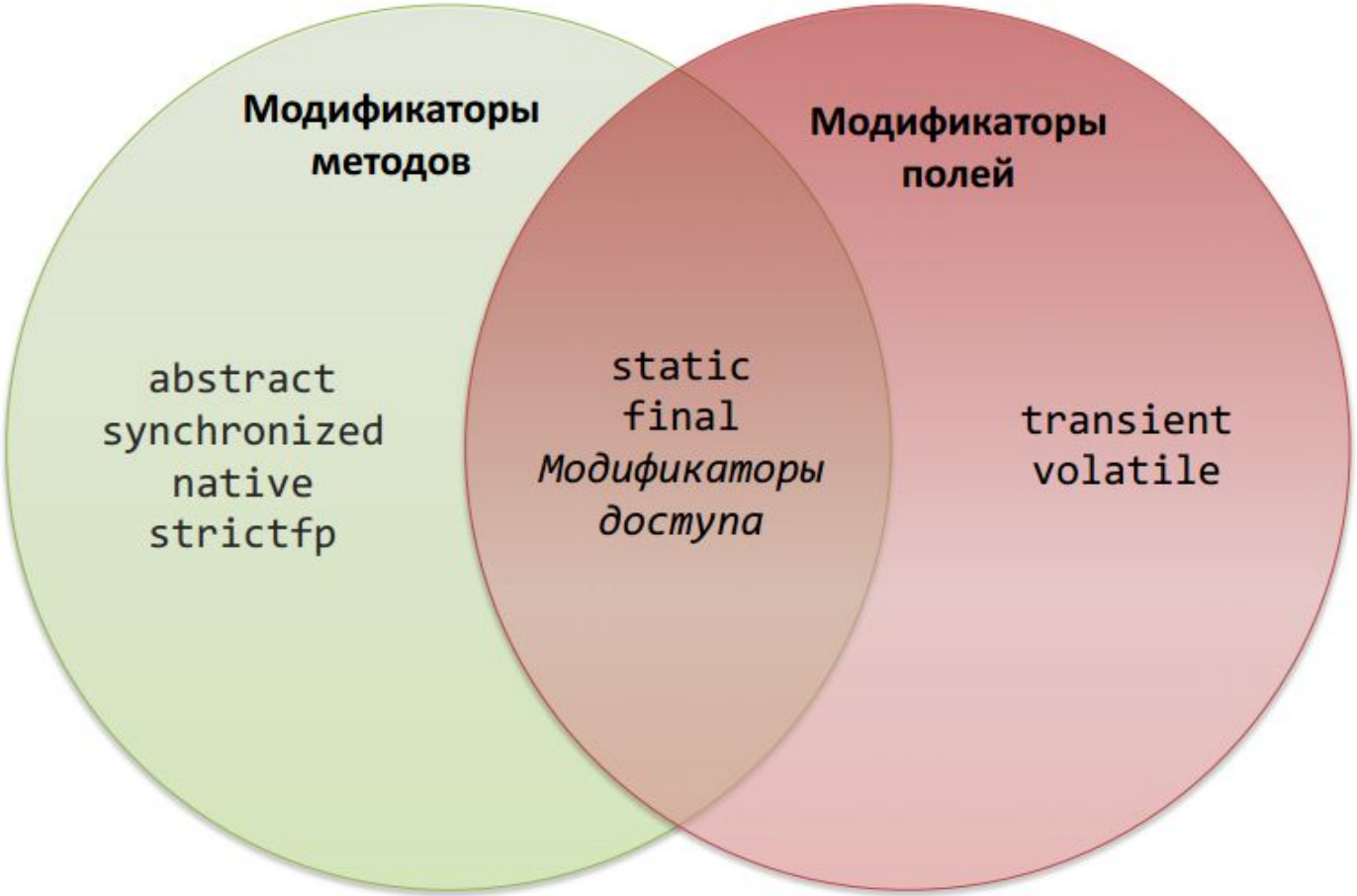
sinchronized. Метод защищен от разрушения данных при попытке использования этих данных несколькими методами сразу.

native. Метод не имеет реализации на Java. Он должен быть написан на другом языке (C/C++, Fortran ит.д.) и добавлен в систему в виде загружаемой динамической библиотеки.

Throws. Метод не будет обрабатывать описанные исключения, их должен будет обработать вызывающий метод.

Если метод не возвращает никакого значения, указывается ключевое слово void, в *теле метода* обязательно должно встречаться return -выражение.





Если два и более методов класса имеют одно имя, но их параметры не совпадают, то такие методы называют перегруженными.

Сигнатура определяется именем *метода* и его аргументами (количеством, типом, порядком следования).

Например,

```
class Point {
        void get() {}
                void get(int x) {}
     void get(int x, double y) {}
      void get(double x, int y) {}
}
```


Сигнатура метода =
<*имя*> + <*тип_параметра1*> + <*тип_параметра2*> + …

Перегружаемые методы должны иметь разные сигнатуры

```
public void add(String a)            :: add(String)
public void add(int a)               :: add(int)
protected int add(String a, int b)   :: add(String,int)
public void sub(String a)            :: sub(String)
public int add(String b)             :: add(String)
```

Конструктор — это именованный блок кода, отвечающий за инициализацию объекта. Он носит имя класса и является специальным методом и может иметь параметры, которые необходимо указывать в скобках при создании объекта.

Конструктор – это метод, который автоматически вызывается при создании объекта класса и выполняет действия по инициализации объекта, вызывается не по имени, а только вместе с ключевым словом **new** при создании экземпляра класса. Конструктор не возвращает значение, но может иметь параметры и быть перегружаемым.

Иногда необходимо использовать в теле метода ссылку на объект, который его вызывает. Для этого существует специальная ссылка this.


```java
public class Man {

    private String name;

    public String getName() {
        return name;
    }

    public Man(String name) {
        this.name = name;
    }

    public Man() {
        this("Anonymous");
    }

    public void sayHello() {
        System.out.println("Hello, my name is "+ getName());
    }

    public static void main(String[] args) {
        Man man = new Man("Marry");
        Man unknown = new Man();
        unknown.sayHello();
    }
}
```



```
class Cow {

    static final String NAME = "Vacca Vulgaris";

    static String sayHello(String name) {
        return "Hello, " + name;
    }

    int legCount = 4;

    void eatGrass() {
        // niam-niam
    }
}

Cow.sayHello(Cow.NAME);
Cow cow = new Cow();
cow.legCount = 3;
cow.eatGrass();
```


Блок инициализации будет выполняться при создании объекта. Статическая инициализация предусматривает обращение только к статическим элементам этого класса.

```
class Body {
    public static final int MAGIC;
    static {
        MAGIC = 42;
    }
    private static long nextId = 0;
    public long idNum;
    {
        prepare();
        idNum = nextId++;
    }
    public String name = "No name";
    public Body orbitis = null;

    Body (String name, Body orbitis) {
        this.name = name;
        this.orbitis = orbitis;
    }
}
```

- Эккель Б. Философия Java. –Эккель Б. Философия Java. – СПб.: Питер, 2009. – 640 с.
- http://www.intuit.ru/studies/courses/16/16/info
- Шилдт Г. Java. Полное руководство. – СПб.: Вильямс, 2012. – 1104 с.
- Шилдт Г. Полный справочник по Java. Java SE 6 Edition. – СПб.: Вильямс, 2007. – 1040 с.
- Шилдт Г., Холмс Д. Искусство программирования на Java. – СПб.: Вильямс, 2005. – 333 с.
- Шилдт Г. Java. для начинающих. – СПб.: Вильямс, 2009. – 720 с.

Q&A

Thank you!

