Лексика языка Java

Лекция 2

УНЦ «Инфоком»
Самара
2013

NetCracker®

План лекции

- Структура исходного кода и его элементы
- Типы данных
- Описание классов
  - Общая структура
  - Поля
  - Методы
  - Конструкторы
  - Блоки инициализации
- Точка входа программы

Кодировка

- Java спроектирована быть многоязыковой
- Java ориентирована на Unicode
- Первые 128 символов почти идентичны набору ASCII
- Символы Unicode задаются двухбайтными последовательностями:
  - `\u0000`, `\uFFff`, `\u0410`, `\u044F`
- **Java чувствителен к регистру!**

Исходный код

Исходный код разделяется на:

- Пробелы – разделяют текст на лексемы
  - ASCII-символ SP, \u0020, дес. код 32, пробел классический
  - ASCII-символ HT, \u0009, дес. код 9, горизонтальный таб
  - ASCII-символ FF, \u000c, дес. код 12, перевод страницы
  - ASCII-символ LF, \u000a , дес. код 10, символ новой строки
  - ASCII-символ CR, \u000d, дес. код 13, возврат каретки
- Комментарии
- Лексемы

Исходный код

```
double a = 1, b = 1, c = 6;
double D = b * b - 4 * a * c;

if (D >= 0) {
    double x1 = (-b + Math.sqrt (D)) / (2 * a);
    double x2 = (-b - Math.sqrt (D)) / (2 * a);
}
```

```
double a=1,b=1,c=6;double D=b*b-4*a*c;if(D>=0)
{double x1=(-b+Math.sqrt(D))/(2*a);double
x2=(-b-Math.sqrt(D))/(2*a);}
```

Лексемы

- Идентификаторы
- Служебные (ключевые) слова
  `class`, `public`, `const`, `goto`
- Литералы
- Разделители
  `{ } [ ] ( ) ; . ,`
- Операторы
  `= > < ! ? : == && ||`

Идентификаторы

- Имена, задаваемые элементам языка для упрощения доступа к ним
- Можно записывать символами Unicode
- Состоят из букв и цифр, знаков `_` и `$`
- Не допускают совпадения со служебными словами, литералами `true`, `false`, `null`
- Длина имени не ограничена

Служебные (ключевые) слова

abstract	double	int	strictfp	
boolean	else	interface	super	
break	extends	long	switch	
byte	final	native	synchronized	
case	finally	new	this	
catch	float	package	throw	
char	for	private	throws	
class	goto	protected	transient	
const	if	public	try	
continue	implements	return	void	
default	import	short	volatile	
do	instanceof	static	while	

Типы данных

- **Ссылочные**
  - Предназначены для работы с объектами
  - Переменные содержат ссылки на объекты
  - Ссылка – это не указатель!
  - Тип переменной определяет контракт доступа к объекту
- **Примитивные (простые)**
  - Предназначены для работы со значениями естественных, простых типов
  - Переменные содержат непосредственно значения

Ссылочные типы

- К ссылочным типам относятся типы классов (в т.ч. массивов) и интерфейсов
- Переменная ссылочного типа способна содержать ссылку на объект, относящийся к этому типу
- Ссылочный литерал `null`

Примитивные типы

- Булевский (логический) тип
  - `boolean` – допускает хранение значений `true` или `false`
- Целочисленные типы
  - `char` – 16-битовый символ Unicode
  - `byte` – 8-битовое целое число со знаком
  - `short` – 16-битовое целое число со знаком
  - `int` – 32-битовое целое число со знаком
  - `long` – 64-битовое целое число со знаком
- Вещественные типы
  - `float` – 32-битовое число с плавающей точкой (IEEE 754-1985)
  - `double` – 64-битовое число с плавающей точкой (IEEE 754-1985)

Литералы

- Булевы
  `true false`
- Символьные
  `'a' '\n' '\\' '\377' '\u0064'`
- Целочисленные
  `29  035  0x1D  0X1d 0xffffL`
  - По умолчанию имеют тип `int`
- Числовые с плавающей запятой
  `1.  .1  1e1 1e-4D 1e+5f`
  - По умолчанию имеют тип `double`
- Строковые
  `"Это строковый литерал"  ""`

Целочисленные литералы

- десятичный вид
  - цифры 0-9
- восьмеричный
  - префикс 0, цифры 0-7
- шестнадцатеричный вид
  - префикс 0x или 0X, цифры 0-9, буквы A-F

```
1000000L    123l       -200
0777777L    00L        -0100L
0X7fffFL    0xC0B0L      0xCafe
```

Дробные литералы

- целая часть;
- десятичная точка (ASCII -символ точка);
- дробная часть;
- порядок (ASCII -буква **E** в произвольном регистре и целое число с опциональным + или -)
- окончание-указатель типа

```
7.2        5.25D     4.
8.       3.14F      .2
.9       .7f       3e3
7e-10      5d      5f
```

Символьные литералы

- один символ из набора Unicode в одиночных кавычках (апострофах)
- допускается специальная запись для описания символа через его код
- Есть поддержка ввода символа через восьмеричный код (совместимости с C)

```
'a'     '\u0041'    '\n'
' '     '\u0410'    '\r'
'M'     '\u0391'    '\\'
```

Символьные литералы

- Для записи специальных символов используются обозначения

```
\b   \u0008   backspace BS - забой
\t   \u0009   horizontal tab HT - табуляция
\n   \u000a   linefeed LF - конец строки
\f   \u000c   form feed FF - конец страницы
\r   \u000d   carriage return CR - возврат
каретки
\"   \u0022   double quote " - двойная кавычка
\'   \u0027   single quote ' - одинарная кавычка
\\   \u005c   backslash \ - обратная косая черта
\16-тиричный код символа, от \u0000 до \u00ff
```

Строковые литералы

- состоят из набора символов и записываются в двойных кавычках
- длина литерала может быть нулевой или сколь угодно большой
- каждый строковый литерал является экземпляром класса `String`
- для создания литерала из нескольких строк используются символы `\n` и/или `\r`, либо оператор конкатенации строк `+`

Строковые литералы

- Примеры строковых литералов

```
""      // литерал нулевой длины
"\""      //литерал, состоящий из одного
символа "
"а это?"  //простой строковый литерал

"Hello, world!\r\nHello!" //литерал из 2-х
строк

"Длинный текст " +
   "с переносом"  //выражение из 2-х литералов
```

Описание класса

Класс может содержать:

- поля,
- методы,
- вложенные классы и интерфейсы.

```
class Body {
  public long idNum;
  public String name;
  public Body orbits;

  public static long nextID = 0;
}
```

Модификаторы объявления класса

- `public`
  Признак общедоступности класса
- `abstract`
  Признак абстрактности класса
- `final`
  Завершенность класса (класс не допускает наследования)
- `strictfp`
  Повышенные требования к операциям с плавающей точкой

Поля класса

- Объявление поля:

```
[модификаторы] <тип> {<имя> [=
<инициализирующее выражение>]};
```

```
double sum = 2.5 + 3.7;
public double val = sum + 2 * Math.sqrt(2);
```

- Если поле класса явно не инициализируются, ему присваивается значение по умолчанию его типа (`0`, `false` или `null`)

Поля класса

- Модификаторы полей:
  - модификаторы доступа
  - `static`
    поле статично (принадлежит контексту класса)
  - `final`
    поле не может изменять свое значение после инициализации
  - `transient`
    поле не сериализуется (влияет только на механизмы сериализации)
  - `volatile`
    усиливает требования к работе с полем в многопоточных программах

Методы

- Объявление метода:

```
[модификаторы] <тип> <сигнатура>
[throws исключения] {<тело>}
```

```
class Primes {
  static int nextPrime(int current) {
    <Вычисление простого числа в теле метода>
  }
}
```

Модификаторы методов

- Модификаторы доступа
- `abstract`
  абстрактность метода (тело при этом не описывается)
- `static`
  статичность метода (метод принадлежит контексту класса)
- `final`
  завершенность метода (метод не может быть переопределен при наследовании)

Модификаторы методов

- `synchronized`
  синхронизированность метода (особенности вызова метода в многопоточных приложениях)
- `native`
  «нативность» метода (тело метода не описывается, при вызове вызывается метод из native-библиотеки)
- `strictfp`
  повышенные требования к операциям с плавающей точкой

Особенности методов

- Для нестатических методов вызов через ссылку на объект или в контексте объекта

```
reference.method();
methodReturningReference().method();
```

- Для статических методов вызов через имя типа, через ссылку на объект или в контексте класса

```
ClassName.staticMethod();
reference.staticMethod();
staticMethodReturningReference().method();
```

- Наличие круглых скобок при вызове обязательно, т.к. они являются оператором вызова метода

Особенности методов

- На время выполнения метода управление передается в тело метода
- Возвращается одно значение простого или объектного типа

```
return someValue;
```

- Аргументы передаются по значению, т.е. значения параметров копируются в стек:
  - для примитивных типов копируются сами значения
  - для ссылочных типов копируется значение ссылки
- Перегруженными являются методы с одинаковыми именами и различными сигнатурами

Создание объектов

- Создание ссылки и создание объекта – **различные** операции
- Используется оператор **new**, он возвращает ссылку на объект
- После оператора указывается имя конструктора и его параметры

```
Body sun;
sun = new Body();
sun.idNum = Body.nextID++;
sun.name = "Sun";
sun.orbits = null;

Body earth = new Body();
earth.idNum = Body.nextID++;
earth.name = "Earth";
earth.orbits = sun;
```

Конструкторы

- Память для объекта выделяет оператор **`new`**
- Конструкторы предназначены для формирования начального состояния объекта
- Правила написания конструктора сходны с правилами написания методов
- Имя конструктора совпадает с именем класса

Конструкторы

- Для конструкторов разрешено использование **только** модификаторов доступа
- При написании конструктор **не имеет** возвращаемого типа
- Оператор возврата `return` прекращает выполнение текущего конструктора
- Конструкторы могут быть перегружены
- Конструкторы могут вызывать друг друга с помощью ключевого слова `this()` в первой строке конструктора

Конструкторы

- Если в классе явно не описан ни один конструктор, автоматически создается т.н. конструктор по умолчанию, не имеющий параметров
- Если в классе описан хотя бы один конструктор, то автоматически конструктор по умолчанию не создается
- Также конструктором по умолчанию называют конструктор, не имеющий параметров

Конструкторы

```
class Body {
  public long idNum;
  public String name = "No Name";
  public Body orbits = null;

  private static long nextID = 0;

  Body() {
    idNum = nextID++;
  }
  Body(String name, Body orbits) {
    this();
    this.name = name;
    this.orbits = orbits;
  }
}
```

Деструкторы?

- В ряде языков деструкторы выполняют действия, обратные действию конструкторов: освобождают память, занимаемую объектом, и «деинициализируют» объект (освобождают ресурсы, очищают связи, изменяют состояние связанных объектов)
- Если после вызова деструктора где-то осталась ссылка (указатель) на объект, ее использование приведет к возникновению ошибки
- В Java деструкторов нет, вместо них применяется механизм автоматической сборки мусора

Автоматическая сборка мусора

- В случае нехватки памяти для создания очередного объекта виртуальная машина находит недостижимые объекты и удаляет их
- Процесс сборки мусора можно инициировать принудительно
- Для явного удаления объекта следует утратить все ссылки на этот объект и инициировать сбор мусора
- Взаимодействие со сборщиком осуществляется через системные классы `java.lang.System` и `java.lang.Runtime`

Модификаторы доступа

- `private`
  Доступ только в контексте класса
- `(package, default, none)`
  Доступ для самого класса и классов в том же пакете
- `protected`
  Доступ в пределах самого класса, классов-наследников и классов пакета
- `public`
  Доступ есть всегда, когда доступен сам класс

Блоки инициализации

- Если некоторые действия по инициализации должны выполняться в любом варианте создания объекта, удобнее использовать блоки инициализации
- Тело блока инициализации заключается в фигурные скобки и располагается на одном уровне с полями и методами
- При создании объекта сначала выполняются инициализирующие выражения полей и блоки инициализации (в порядке их описания в теле класса), а потом тело конструктора

Блоки инициализации

```
class Body {
  public long idNum;
  public String name = "No Name";
  public Body orbits = null;

  private static long nextID = 0;

  {
    idNum = nextID++;
  }

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

Статическая инициализация

```
class Primes {
  static int[] knownPrimes = new int[4];

  static {
    knownPrimes[0] = 2;
    for (int i=1; i<knownPrimes.length; i++)
      knownPrimes[i] = nextPrime(i);
  }

  //nextPrime() declaration etc.
}
```

- Статический блок инициализации выполняет инициализацию контекста класса
- Вызов статического блока инициализации происходит в процессе загрузки класса в виртуальную машину

Понятие о пакетах

- Способ логической группировки классов
- Комплект ПО, который можно распространять независимо и применять вместе с другими пакетами
- Членами пакетов являются:
  - классы,
  - интерфейсы,
  - вложенные пакеты,
  - дополнительные файлы ресурсов

Функциональность пакетов

- Позволяют группировать взаимосвязанные классы и интерфейсы в единое целое
- Способствуют созданию пространств имен, позволяющих избежать конфликтов идентификаторов, относящихся к разным типам
- Обеспечивают дополнительные средства защиты элементов кода
- Образуют иерархическую систему

Способы реализации и доступ к пакетам

- Пакеты могут быть реализованы:
  - в виде структуры каталогов с файлами классов,
  - в виде jar-архива.
- Путь к используемым пакетам указывается:
  - непосредственно при запуске JVM,
  - через переменную окружения CLASSPATH (по умолчанию CLASSPATH="").

Понятие имени

- Имена задаются посредством идентификаторов, указывают на компоненты программы

- Пространства имен
  - пакеты
  - типы
  - поля
  - методы
  - локальные переменные и параметры
  - метки

- Имена бывают составные (`java.lang.Double`) и простые (`Double`)

Душераздирающий, но корректный код

Пример зависимости имени от контекста

```
package Reuse;

class Reuse {
  Reuse Reuse (Reuse Reuse) {
    Reuse:
    for(;;) {
      if (Reuse.Reuse(Reuse) == Reuse)
        break Reuse;
    }
    return Reuse;
  }
}
```

Понятие модуля компиляции

- Модуль компиляции хранится в `.java` файле и является единичной порцией входных данных для компилятора.

- Состоит из:
  - объявления пакета
    ```
    package mypackage;
    ```
  - выражений импортирования
    ```
    import java.net.Socket;
    import java.io.*;
    ```
  - объявлений верхнего уровня – классов и интерфейсов

Объявление пакета

- Первое выражение в модуле компиляции (например, для файла `java/lang/Object.java`)
  - `package java.lang;`
- При отсутствии объявления пакета модуль компиляции принадлежит безымянному пакету (не имеет вложенных пакетов)
- Пакет доступен, если доступен модуль компиляции с объявлением пакета

Объявление пакета

```
package space.stars;
class Sun {
   ...
}
```

- Если файл модуля компиляции доступен JVM, то пакеты также доступны
- Если пакет доступен, то область его видимости – все модули компиляции
  - ограничений на доступ к пакетам нет
  - пакеты `java.lang` и `java.io` доступны всегда

Выражения импорта

- Доступ к типу из данного пакета – по простому имени типа
- Доступ к типу из других пакетов – по составному имени пакета и имени типа
  - сложности при многократном использовании
- import-выражения упрощают доступ
  - импорт одного типа (`import java.net.URL;`)
  - импорт пакета с типами (`import java.net.*;`)

Выражения импорта

- Попытка импорта пакета, недоступного на момент компиляции, вызовет ошибку
- Дублирование импорта игнорируется
- Нельзя импортировать вложенный пакет
  - `import java.net; //ошибка компиляции`
- При импорте типов пакета вложенные пакеты не импортируются!

Выражения импорта

- Алгоритм компилятора при анализе типов:
  - выражения, импортирующие типы
  - другие объявленные типы
  - выражения, импортирующие пакеты
- Если тип импортирован явно невозможны:
  - объявление нового типа с таким же именем
  - доступ по простому имени к одноименному типу в текущем пакете

Выражения импорта

- Импорт пакета не мешает объявлять новые типы или обращаться к имеющимся типам текущего пакета по простым именам
  - поиск типа сначала в текущем пакете
- Импорт конкретных типов дает возможность при прочтении кода сразу понять, какие внешние типы используются
  - но эти типы могут и не использоваться

Объявление верхнего уровня

```
package first;
class MyFirstClass {
}
interface MyFirstInterface {
}
```

- Область видимости типа – пакет
- Доступ к типу извне его пакета
  - по составному имени
  - через выражения импорта
- Разграничение (модификаторы) доступа

Объявление верхнего уровня

- В модуле компиляции может быть максимум один `public` тип
- Имя типа и имя файла должны совпадать
- Другие не-`public` типы модуля должны использоваться только внутри этого модуля
- Как правило, один модуль компиляции содержит один тип

Правила именования

- Пакеты
  - `java.lang`
  - `javax.swing`
  - `ru.ssau.infokom`
  - `com.sun.xml.internal.ws.protocol.xml`
- Типы
  - `Student`
  - `Cloneable`
  - `Serializable`
  - `ArrayIndexOutOfBoundException`
- Поля
  - `value`
  - `enabled`
  - `distanceFromShop`

Правила именования

- Методы
  - `getValue(), setValue(…)`
  - `isEnabled(…)`
  - `length()`
  - `toString()`
- Поля-константы
  - `PI`
  - `SIZE_MIN, SIZE_MAX, SIZE_DEF`
- Локальные переменные
  - `byte b; char c; int i,j,k; long l; double d;`
  - `Object o; String s;`
  - `Exception e`

Точка входа программы

- Метод
- Статический
- Доступный
- С параметрами-аргументами
- Без возвращаемого значения

```
class Echo {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i] + " ");
    System.out.println();
  }
}
```

Комментарии

- Не влияют на итоговый бинарный код
- Используются для ввода пояснений
- Бывают двух видов
  - Строчные, одна строка
  - Блочные, несколько строк

Комментарии

- `// Комментарий`
  Символы после `//` и до конца текущей строки игнорируются
- `/* Комментарий */`
  Все символы, заключенные между `/*` и `*/`, игнорируются
- `/** Комментарий */`
  Комментарии документирования

Комментарии

```
int bonus = 100500; // эм, комментарии?
```

```
/*
   Особенный цикл, начинаться с единицы
   из-за особенностей алгоритма
*/
for (int i=1; i<10; i++) {
...
}
```

```
float s = 2*Math.PI/*getRadius()*/;
// Закомментировано для отладки
```

Комментарии

```
// Текст /*…*/ будет частью строки s
String s = "text/*just text*/";
```

```
// Ошибка, комментарий разбил имя метода
circle.get/*comment*/Radius();
```

```
// Комментарий может разделять вызовы функций:
circle./*comment*/getRadius();
```

```
// Комментарий может заменять пробелы:
int/*comment*/x=1;
```

Комментарии

```
// еще один честный комментарий ниже
/* начало комментария /* // /** завершение: */
```

```
// Ошибка компилятора на 7 строке
1. /*
2.     comment
3.     /*
4.     more comments
5.     */
6.     finish:
7. */
```

Комментарии документирования (javadoc)

- Начинаются с `/**`, заканчиваются `*/`
- В строках начальные символы `*` и пробелы перед ними игнорируются
- Перед объявлением классов, интерфейсов, полей, методов и конструкторов
- Допускают использование HTML-тэгов, кроме заголовков
- Специальные тэги `@see`, `@param`, `@deprecated`

Комментарии документирования (javadoc)

```
/**
    * Вычисление модуля целого числа.
    * Этот метод возвращает
    * абсолютное значение аргумента x.
    */
int getAbs(int x) {
    if (x>=0)
        return x;
    else
        return -x;
}
```

Комментарии документирования (javadoc)

```
/**
    * Первое предложение - краткое описание метода.
    * <p>
       * <blockquote><pre>
       * if (condition==true) {
       *    x = getWidth();
       * }
       * </pre></blockquote>
    * А так описывается HTML-список:
       * <ul><li>Вот наклонный шрифт <i>курсив</i>,
       * <li>Вот шрифт стиля <b>bold</b></ul>
    */
public void calculate (int x, int y) {
    ...
}
```

Комментарии документирования (javadoc)

Первое предложение – краткое описание метода.

```
if (condition==true) {
    x = getWidth();
}
```

А так описывается HTML-список:

* Вот наклонный шрифт *курсив*,
* Вот шрифт стиля **bold**

Спасибо за внимание!

NetCracker®

Дополнительные источники

- Арнолд, К. Язык программирования Java [Текст] / Кен Арнолд, Джеймс Гослинг, Дэвид Холмс. – М. : Издательский дом «Вильямс», 2001. – 624 с.
- Вязовик, Н.А. Программирование на Java. Курс лекций [Текст] / Н.А. Вязовик. – М. : Интернет-университет информационных технологий, 2003. – 592 с.
- Хорстманн, К. Java 2. Библиотека профессионала. Том 1. Основы [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010 г. – 816 с.
- Эккель, Б. Философия Java [Текст] / Брюс Эккель. – СПб. : Питер, 2011. – 640 с.
- JavaSE at a Glance [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/overview/index.html, дата доступа: 21.10.2011.
- JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html, дата доступа: 21.10.2011.