Теория и технология программирования

Основы программирования на языке Java

Типы, операции, управляющие конструкции

Система типов Java: знакомство в сравнении с языком C++

- Программа как множество классов
- Безопасная система типов
- Обработка массивов и контроль выхода за границы
- Обработка строк
- Возможности C++, отсутствующие или измененные в Java

Программа как множество классов

- Объектно-ориентированная модель организации вычислительного процесса
- Понятие о ссылочных типах
- Импорт внешних классов. Понятие пакета
- Библиотека классов
- Управление памятью. Сборка мусора

Понятие о ссылочных типах

- В Java все объектные переменные являются ссылками!

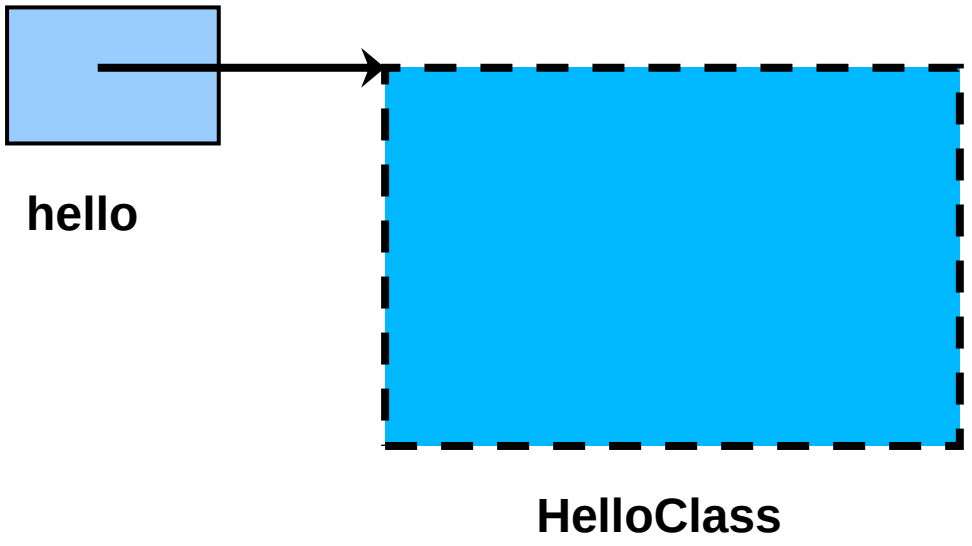
Основные виды абстракций ООП

- АТД - class
- Экземпляр класса – object, class instance

Классы и объекты (модель Java)

- Объекты всегда размещаются в динамической памяти

```
HelloClass hello = new HelloClass;
```



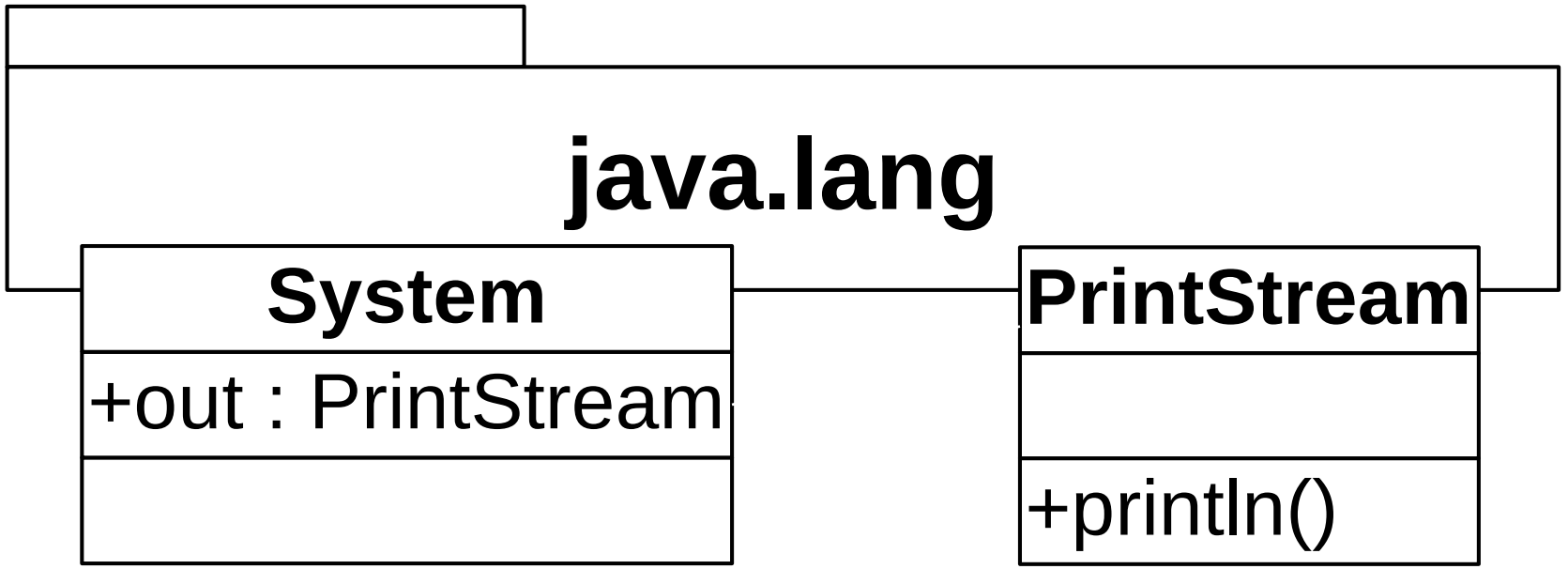
Импорт внешних классов

Demo: HelloNetBeans

```
import java.util.*;
class HelloClass {
    public void hi() {
        System.out.println( "Hello NetBeans!" );
        System.out.print( "It's  nice day: " );
        System.out.println( new Date() );
    }
    public void bye() {
        System.out.println( "Bye!" );
    }
}
```

Библиотека классов (модель Java)

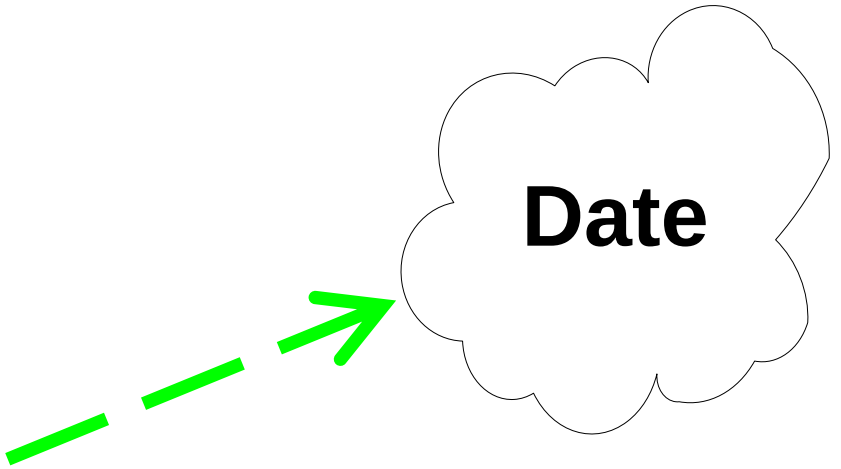
- иерархия пакетов



Управление памятью

```
System.out.println( new Date() );
```

Управление памятью



Никто не ссылается

Управление памятью. Сборка мусора

Garbage Collector в какой-то момент удаляет объект, на который нет ссылок

Встроенные типы

Тип	Размер (бит)	Минимальное значение	Максимальное значение	Класс-оболочка
`boolean`	-	-	-	`Boolean`
`char`	16	`Unicode 0`	`U2^16-1`	`Character`
`byte`	8	`-128`	127	`Byte`
`short`	16	`-2^15`	`2^15-1`	`Short`
`int`	32	`-2^31`	`2^31-1`	`Integer`
`long`	64	`-2^63`	`2^63-1`	`Long`
`float`	32	`IEEE754`	`IEEE754`	`Float`
`double`	64	`IEEE754`	`IEEE754`	`Double`
`void`	-	-	-	`Void`

Булевский тип: замечания для программистов на C++

- Все операции отношения порождают значения булевского типа
- Все условные конструкции работают с булевским типом
- Не совместимы с целыми типами
- Размер объекта не определен
- Можно выводить в выходной поток

Приведение типов

- Приведение – это изменение типа (автоматическое или явное).

- Явное приведение позволяет :
  - > сделать тип преобразования более точным.
  - > форсировать преобразование, если оно не может быть выполнено автоматически.

Расширяющее преобразование

- Результирующий тип имеет больший диапазон значений, чем исходный тип

```
int x = 200;
long y = (long)x;
long value = (long)200; //необязательно

Компилятор делает это автоматически
```

Сужающее преобразование

- Результирующий тип имеет больший диапазон значений, чем исходный тип.

```
long value = 1000L;
int value2 = (int)value; //обязательно

Иногда это единственный способ сделать
код компилируемым
```

Приведение ссылочных типов

- Java позволяет проводить приведение от любого примитивного типа к любому примитивному типу, за исключением boolean.
- Для классов действуют несколько другие правила приведения.

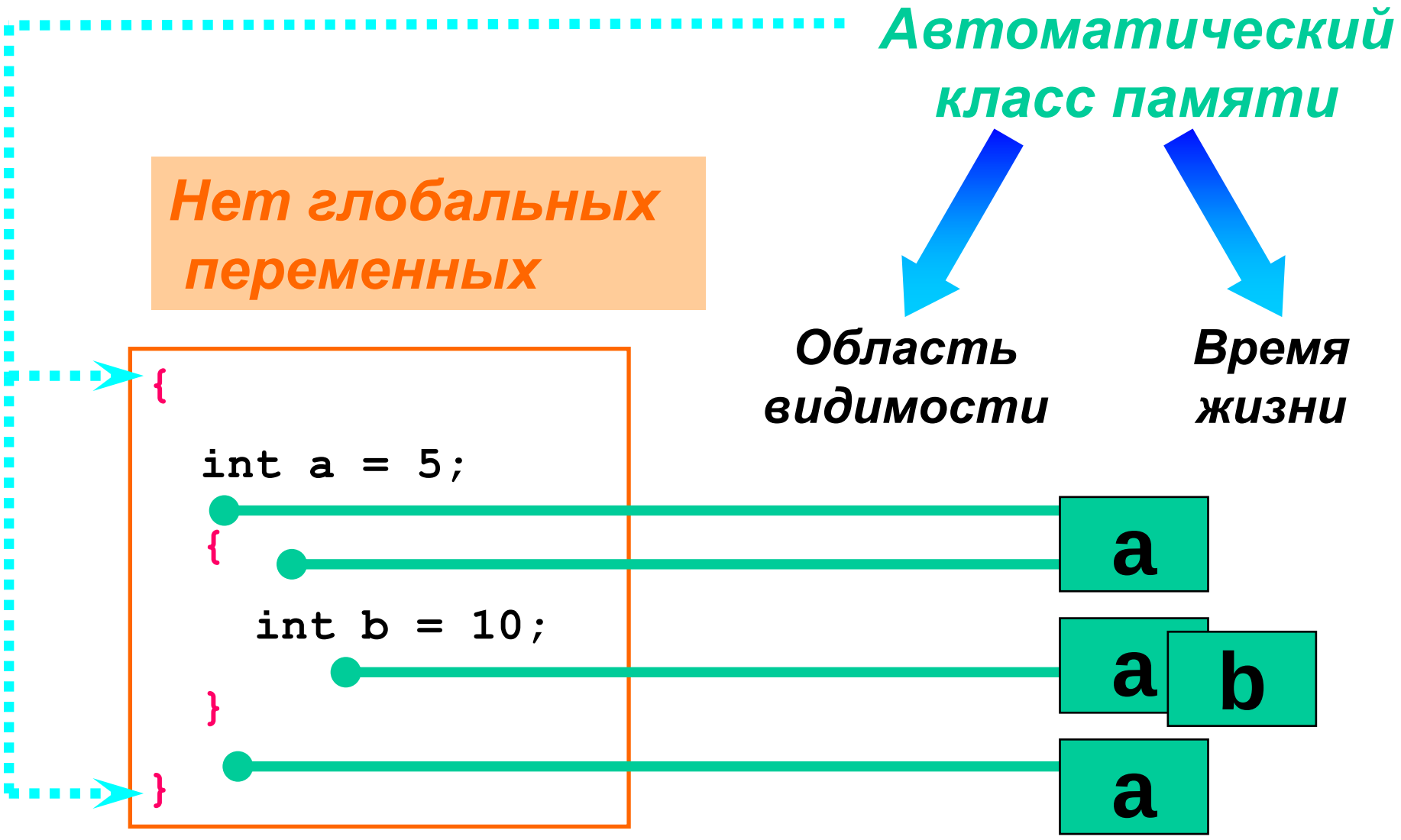
Литералы

- Целочисленные литералы
- Литералы с плавающей точкой
- Булевские литералы
- Символьные литералы
- Строковые литералы

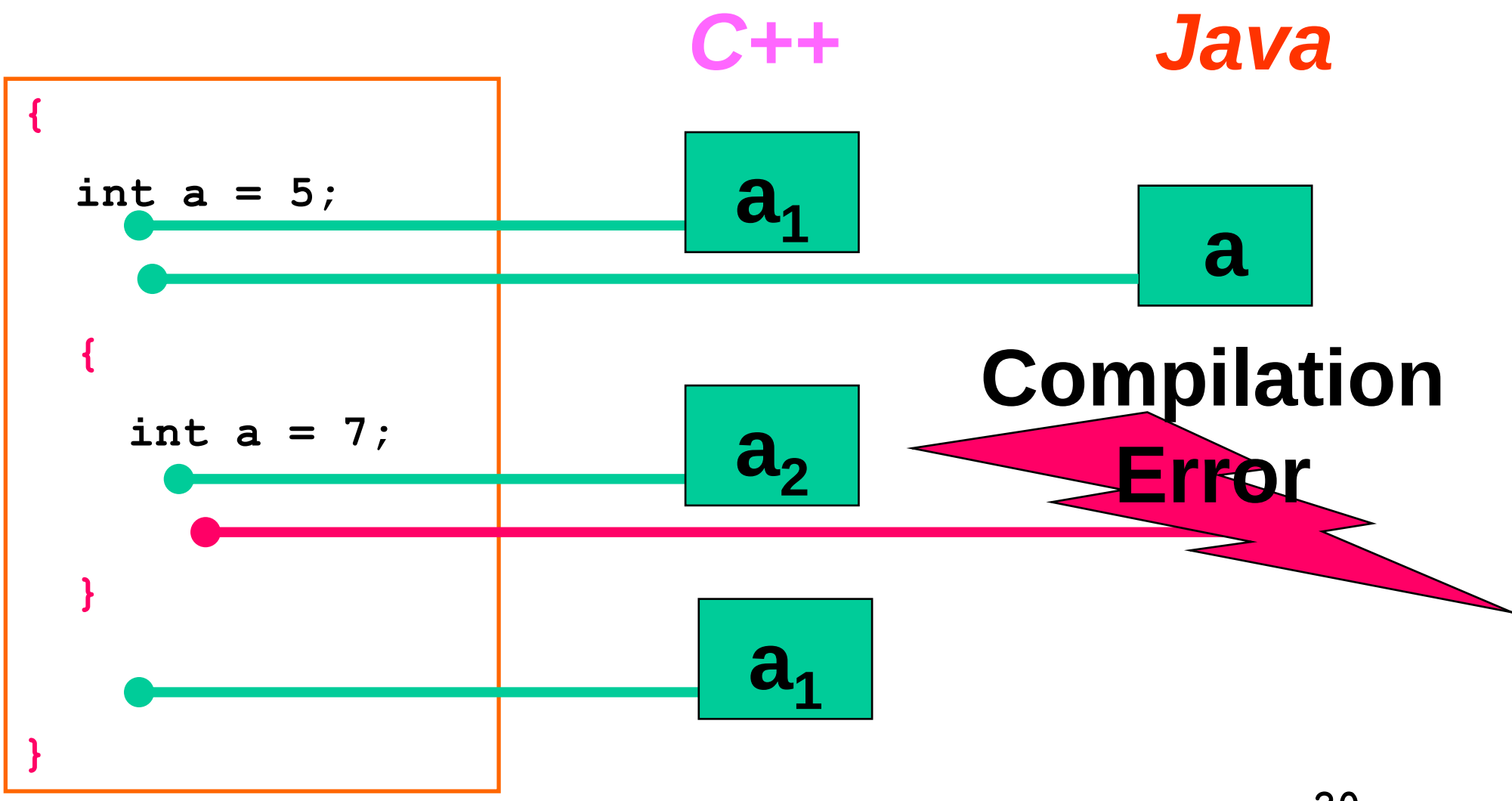
Символы и кодировки

- Прописные русские буквы в кодировке Unicode занимают диапазон:
  - от ' \u0410 ' — заглавная буква А ,
  - до ' \u042F ' — заглавная Я.
- Строчные буквы:
  - от ' \u0430 ' — а ,
  - до ' \044F ' — я .
- В какой бы форме ни записывались символы, компилятор переводит их в Unicode, включая и исходный текст программы.
- Компилятор и исполняющая система Java работают только с кодировкой Unicode.

Области видимости для объектов встроенных типов



Области видимости для объектов встроенных типов



Контроль преобразований

- Все явные присваивания и инициализации контролируются на корректность преобразований типов
- Все неявные присваивания тоже контролируются
- Нет автоматического приведения типов в случае конфликта типов

Арифметические операции

- Операнды должны иметь числовой тип
- Можно использовать операции на типе char
- % определена и для плавающих типов

+	Сложение
+=	Присваивание со сложением
-	Вычитание
-=	Присваивание с вычитанием
*	Умножение
*=	Присваивание с умножением
/	Деление
/=	Присваивание с делением
%	Остаток от деления
%=	Присваивание с вычислением остатка
++	Инкремент
--	Декремент

Поразрядные (bitwise) операции

- Действуют на индивидуальные биты
- Применяются для целочисленных операндов

~	Отрицание (унарная операция)
&	Поразрядное И
|	Поразрядное ИЛИ
^	Поразрядное исключающее ИЛИ
>>	Сдвиг вправо с размножением знака
>>>	То же, старший бит заполняется нулем
<<	Сдвиг влево
&=	Присваивание с поразрядным И (так же для остальных бинарных операций)

Операции отношений

- Реализуют отношения равенства, неравенства, строгого и нестрогого порядка
  - Формируют значение boolean
    - Для данных любых типов

==	Равно
!=	Не равно

    - Только для числовых типов

>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Логические операции

- Работают только со значениями boolean

&	Логическое И
&&	Логическое И (укороченная форма)
|	Логическое ИЛИ
||	Логическое ИЛИ (укороченная форма)
^	Исключающее ИЛИ
==	Равно
!=	Не равно
!	Отрицание (унарная операция)
?:	Тернарная условная операция

Приоритет операций

- Определяет порядок выполнения операций при вычислении выражений

1	( ) [ ] .
2	++ -- ~ !
3	* / %
4	+ -
5	>> >>> <<
6	> >= < <=
7	== !=

8	&
9	^
10	|
11	&&
12	||
13	?:
14	= op=

Массивы и контроль границ

- Работа с массивами – источник множества ошибок управления памятью при программировании на C/C++
- В Java все массивы инициализированы

- ***Невозможно обратится к памяти за пределами пространства, занимаемого массивом***

Доступ к элементам массива

- Доступ осуществляется по индексу.
- Размер хранится в немодифицируемом поле массива length

```
int items[] = new int[10];
int sum = 0;
// ...
for (int i = 0;
     i < items.length;
     i++) {

       sum = sum + items[i];

}
```

Многомерный массив

- Двумерный массив как математический объект – это матрица

	0	1	2	3
0	20	14	-7	9
1	16	-2	99	1
2	4	0	-2	73

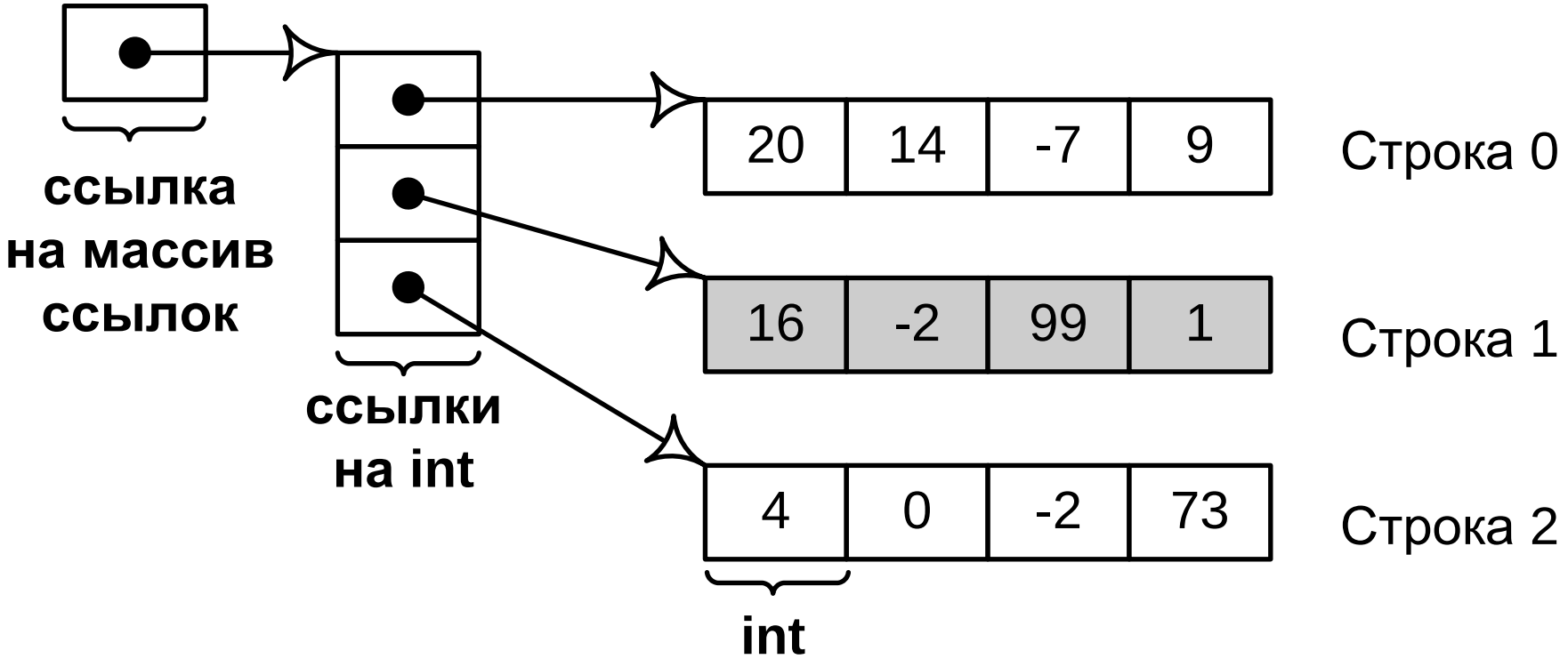
Число строк (**numStr**)

Число столбцов (**numCol**)

Элементы (**numStr*numCol**)

Многомерный массив: размещение данных в памяти

- Хранение организовано в виде ступенчатого (jagged) массива



Массив массивов

- Поэтому каждая строка массива может иметь отличную от других длину

```
int twoDim [][] = new int[4][];
twoDim[0] = new int [10];
twoDim[1] = new int [20];
twoDim[2] = new int [30];
twoDim[3] = new int [100];
```

Пример – поиск значения -1 в двумерном массиве

```
static void findMinusOne(int[][] arr2d) {
L1:
    for (int i=0; i<arr2d.length; i++) {
        for (int j=0; j<arr2d[i].length; j++) {
            if (arr2d[i][j]==-1) {
                System.out.println("i="+i+" j="+j);
                break L1; // break с меткой
            }
        }
    }
}
```

Операции над массивами

- Определены внутри класса **java.util.Arrays** (данный класс содержит только функции, причем все они статические)
  - import java.util.Arrays;

- Arrays.equals(arr1, arr2) – сравнение на поэлементное равенство
  - arr1==arr2 – сравнение ссылок на равенство
- Arrays.binarySearch(arr, elem) – бинарный поиск элемента
- Arrays.copyOf(arr, length) – создать копию массива заданной длины
- Arrays.fill(arr, elem) – заполнение элементом
- Arrays.sort(arr) – сортировка
- И так далее! См. JavaDoc

Ошибки времени выполнения

- Обращение к несуществующему индексу массива отслеживается виртуальной машиной во время исполнения кода

```
public class Main {
    public static void main(String[] args) {
        int array [] = new int[]{1, 2, 3};
        System.out.println(array[3]);
    }
}
```

Попытка обратиться к несуществующему индексу блокируется исполнительной системой

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 3
        at Main.main(Main.java:27)
```

Ошибки времени выполнения

- Попытка поместить в массив неподходящий элемент пресекается виртуальной машиной

```
Object x[] = new String[3];
//попытка поместить в массив содержимое
//несоответствующего типа
x[0] = new Integer(0);
```

```
Нарушение типов также приводит к ошибке времени выполнения

Exception in thread "main" java.lang.ArrayStoreException:
java.lang.Integer
        at Main.main(Main.java:22)
```

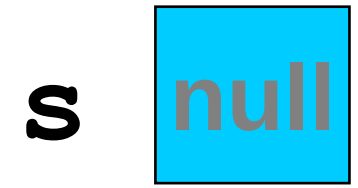
Строки символов

- Класс **java.lang.String** представляет собой хранилище символов и функциональность для работы с ними.
- Строка имеет фиксированную длину и не завершается специальным символом.
- Объект класса **String** – неизменяем.

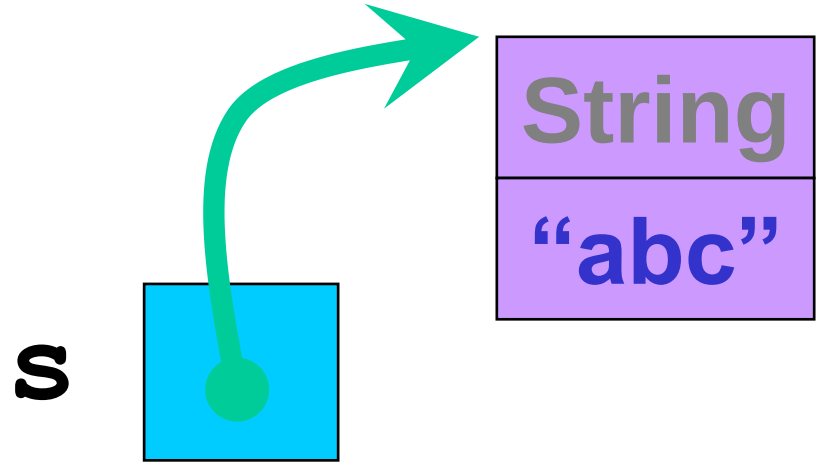
Строки символов

- Строка символов – ссылочный тип

```
String s;
```



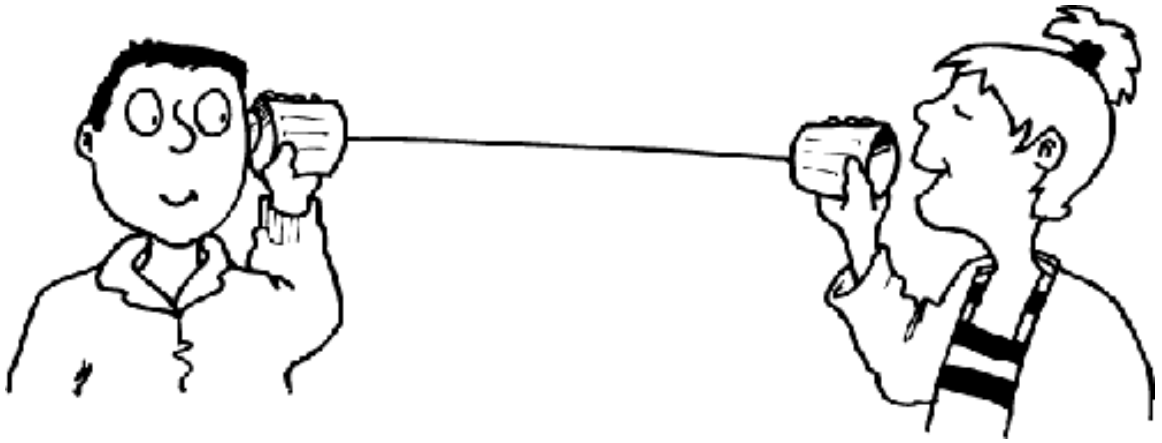

```
String s = "abc";
```



Строки символов

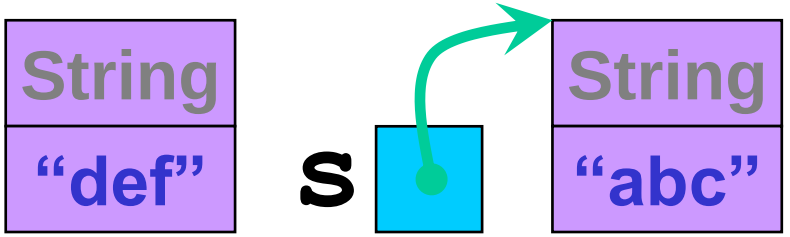
- Операция + поддерживается языком для конкатенации строк
- Перегрузка прочих операций не разрешена

```
String s = “Do” + “you” + “like” + “Java?”;
```



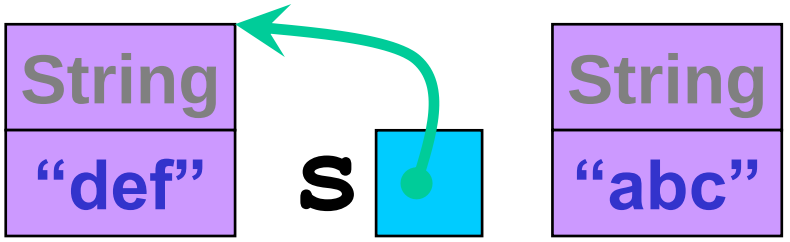
Строки символов

- Строки String неизменяемы
- При выполнении операций над строками порождаются новые объекты (на некоторые из них никто не ссылается)
- Ссылку можно изменить!



Строки символов

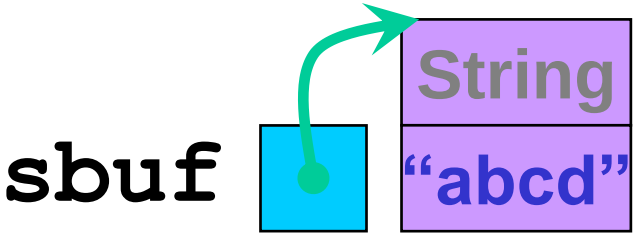
- Строки String неизменяемы
- При выполнении операций над строками порождаются новые объекты (на некоторые из них никто не ссылается)
- Ссылку можно изменить!



Класс StringBuilder

- Используется для работы со строковыми объектами, допускающими изменения

```
StringBuilder sbuf = “abc”;
sbuf.append( “d” );
```



StringBuilder

- Используется, когда требуется создать строку из многих других строк, например:
  - String s = s1 + 23 + s2 + 2.5 + s3;
  - так можно, но неэффективно (будет создано 3 промежуточных строки)!
- Лучше:
  - StringBuilder sb = new StringBuilder();
  - sb.append(s1);
  - sb.append(23);
  - sb.append(s2);
  - sb.append(2.5);
  - sb.append(s3);
  - String s = sb.toString();

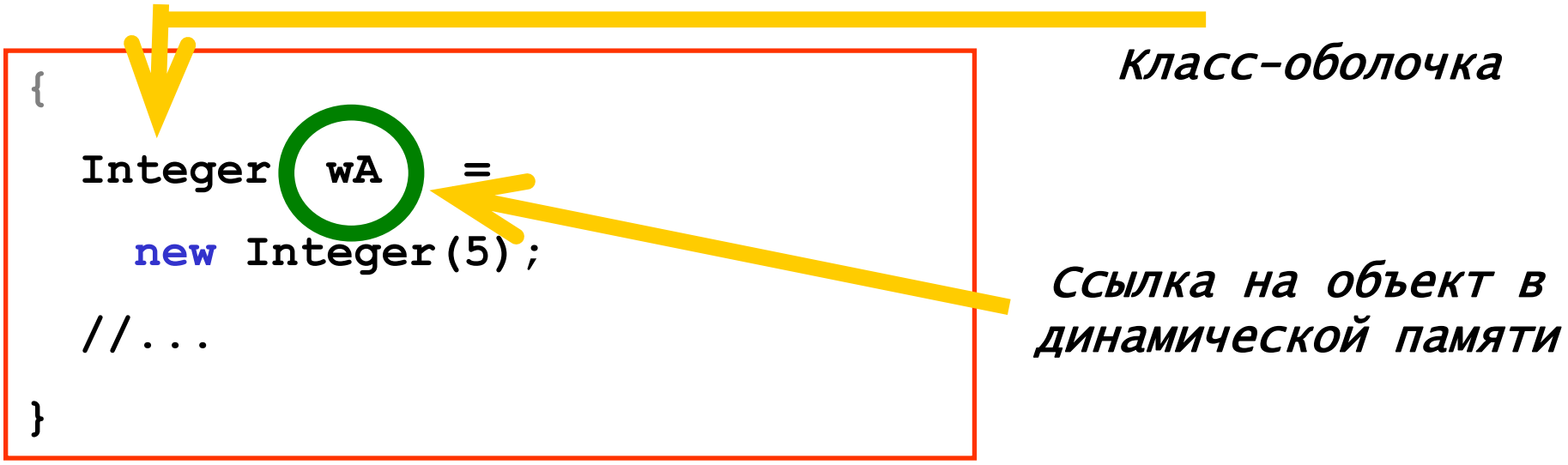
Строки символов и метод main

- Метод **main()** может иметь параметры, передаваемые в виде массива строк

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

Классы-оболочки

- Предоставляют возможность работать с данными размерных типов как с объектами



Demo: TypeWrappers

Классы-оболочки: когда это нужно?

- Чтобы использовать методы класса-оболочки

```
class UseTypeWrappers {
  static void printInt ( int a ) {
    System.out.println( Integer.toBinaryString( a ) );
    System.out.println( Integer.toOctalString( a ) );
    System.out.println( Integer.toHexString( a ) );
  }
}
```

Demo: TypeWrappers

Классы-оболочки: когда это нужно?

- Чтобы использовать методы класса-оболочки для лексического анализа

```
class UseTypeWrappers {
  public static void main( String args[] ) {
    int val = Integer.parseInt( args[0] );
    System.out.println( "val = " + val );
  }
}
```

Demo: TypeWrappers

Классы-оболочки: когда это нужно?

- Чтобы поместить данные размерных типов в коллекции

```
int array[] = { ... };
Stack intStack = new Stack();
for( int i = 0; i < array.length; i++ ) {
  intStack.push( new Integer( array[ i ] ) );
}
```

Demo: TypeWrappers

Классы-оболочки: когда это нужно?

- Чтобы извлечь данные размерных типов из коллекции

```
while( !intStack.empty() ) {
  Integer objVal = (Integer) intStack.pop();
  System.out.println( objVal.intValue() );
}
```

Demo: Autoboxing

Классы-оболочки: автоупаковка и автораспаковка

- Начиная с Java SE 5, не нужно явно создавать объект-оболочку

```
class UseAutobox {
  public static void main( String args[] ) {
    Integer objVal = 100; // autoboxing
    int val = objVal; // autounboxing
    System.out.println(objVal + "=" + val );
    //...
  }
}
```

Demo: Autoboxing

Классы-оболочки: автоупаковка и автораспаковка

- Аналогично обстоит дело с параметрами и возвращаемыми значениями методов

```
class UseAutobox {
  static int some( Integer objVal ) {
    return objVal;
  }
  public static void main( String args[] ) {
     Integer objVal1 = some( 200 );
     ++objVal1;
     System.out.println( objVal1 );
  }
}
```

Demo: Autoboxing

Классы-оболочки: автоупаковка и автораспаковка

- То же справедливо и для коллекций

```
int array[] = { 10, 20, 30 };


Stack intStack = new Stack();
for( int i = 0; i < array.length; i++ ) {
    intStack.push( array[ i ] );
}


while( !intStack.empty() ) {
    System.out.println( intStack.pop()   );
}
```

Многоразрядные числа с высокой точностью

- Предназначены для проведения расчетов без потери точности.
- Не имеют примитивных аналогов.
- **BigInteger** – для представления длинных целых чисел.
- **BigDecimal** – для представления больших чисел с плавающей точкой.

Demo: BigIntegerSample

BigInteger: пример использования

- Инициализация и вычисления

```
public static void main( String args[] ) {
    BigInteger sum = new BigInteger( "0" );

    for( int i=0; i<args.length; i++ ) {
        BigInteger next = new BigInteger( args[i] );
        System.out.println( next.toString() );
        sum = sum.add( next );
    }
    //...
}
```

Demo: BigIntegerSample

BigInteger: пример использования

- Использование методов

```
public static void main( String args[] ) {
    //...
    System.out.println( "Result information" );
    System.out.println( "sum = " + sum.toString() );
    System.out.println( "bit counted: " + sum.bitCount() );
}
```

BigInteger: пример использования

- Вариант for-each цикла for

```
public static void main( String args[] ) {
    BigInteger sum = new BigInteger( "0" );

    for( String x: args ) {
        System.out.println( x );
        sum = sum.add( new BigInteger( x ) );
    }
    //...
}
```

Управляющие конструкции языка Java (самостоятельная работа)

- Условные инструкции (инструкции выбора)
  - if
  - Switch (теперь и для String)
- Циклы
  - while
  - do while
  - for (вариант for each)
- Инструкции перехода
  - break (в том числе – с меткой)
  - continue (в том числе – с меткой)
  - return

Управляющие конструкции языка Java

- Пример
  - Задача: определить, упорядочены ли элементы массива в каком-нибудь направлении (отношение порядка нестрогое)
    - Сначала находим первую пару несовпадающих элементов и определяем потенциально возможную упорядоченность (по возрастанию или убыванию)
    - С учетом найденного направления проверяем упорядоченность оставшейся части

Упражнения для самостоятельной работы

- Напишите программу, которая выводит на консоль значения параметров командной строки
- Напишите программу вычисляющую расстояние, которое проходит луч света за заданное число секунд. Число секунд задается посредством аргумента командной строки
- Напишите программу, определяющую наименьший набор банкнот и монет для получения заданной суммы. Денежную систему выбирайте по вашему усмотрению.

Возможности C++, отсутствующие или измененные в Java

- В Java нет указателей
- В Java нет глобальных переменных
- Параметры методов всегда неизменны, если это объекты встроенных типов (нет связывания по ссылке)
- Строка – это не массив символов
- Не поддерживается перегрузка операций

Ввод-вывод в Java

- Классы ввода-вывода в Java находятся в пакете java.io
  - > import java.io.*;
  - > // или, если нужен конкретный класс
  - > import java.io.PrintStream;

- Что такое **System.in** и **System.out**?
  - > это объекты класса **InputStream** и **PrintStream**
  - > как и все классы – ссылочный тип

Основные возможности PrintStream

- print(var) – вывод var в поток (в случае System.out на консоль)
- println(var) – вывод var в поток с переводом строки
- printf(fmt, ...) или format(fmt, ...) – форматированный вывод, аналог функции printf в языке C
- close() – закрытие потока

Создание нового потока вывода

```
// файловый поток
PrintStream fout = new PrintStream("out.txt");
// или с указанием кодировки
PrintStream fout866 = new PrintStream("out.txt", "CP866");
// или консольный с
   // указанием кодировки
PrintStream out866 = new PrintStream(
      System.out, true, "CP866");
```

Основные возможности InputStream/InputStreamReader

```
char[] buf = new char[50];
int symNum = in.read(buf);
// или
int symNum = in.read(buf, 0, 50);
// и все!
// А как же чтение чисел, например?
```

Создание нового потока ввода

```
InputStreamReader reader = new
      InputStreamReader(System.in,
        "CP866");
// или
FileInputStream fstream = new
      FileInputStream("in.txt");
InputStreamReader freader = new
      InputStreamReader(fstream,
        "CP1251");
```

Разбор введенных данных

```
// Преобразование в строку
String str = new String(buf);
// Длина строки будет совпадать с длиной массива
// Урезание лишних символов
str = str.substring(0, symNum);
// Удаление пробелов в начале и в конце
str = str.trim();
```

Разбор получившейся строки

```
// Разбить строку по пробелам
// 12 34 56 => { "12", "34", "56" }
String[] splitted;
try {
   splitted = str.split(" ");
} catch (PatternSyntaxException e) {
   // Если разбиение не получилось
   out866.println("Неверный формат ввода!");
   return;
}
```

Разбор подстрок

```
// Преобразование к целым
for (String substr: splitted) {
   int num = 0;
   try {
      num = Integer.parseInt(substr);
      out866.println("Введено число: " + num);
   } catch (NumberFormatException e) {
      out866.println("Введено НЕ число: " + substr);
   }
   // ...
}
```

Более сложный вариант разбора строки

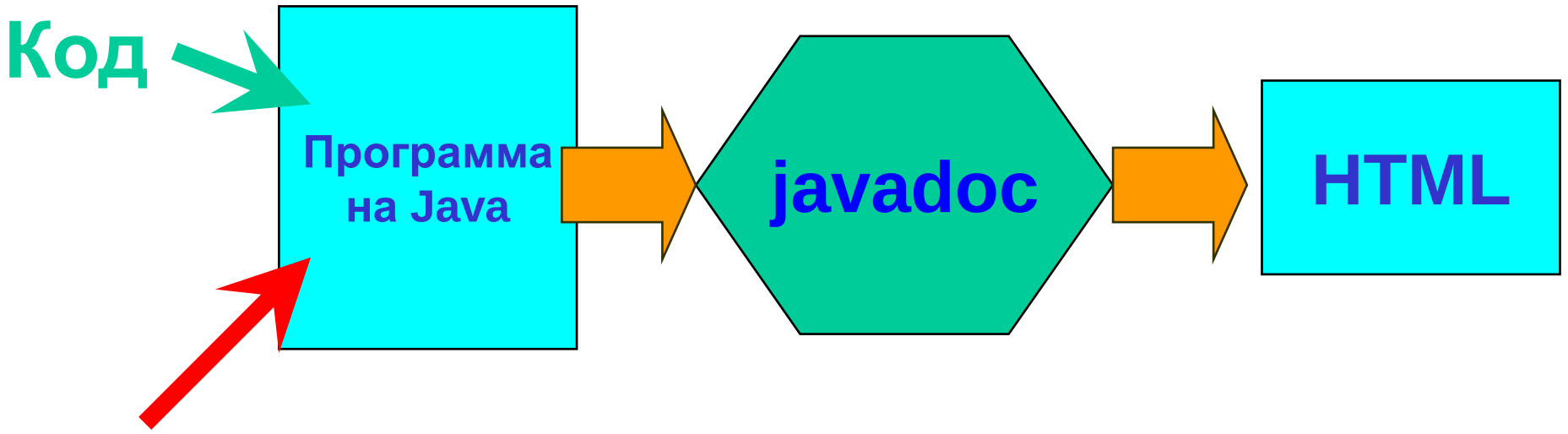
```
StringTokenizer tokenizer = new
      StringTokenizer(str, " \t");
while (tokenizer.hasMoreElements()) {
    String substr = tokenizer.nextToken();
    // ...
}
```

Комментарии и встроенный HTML (самостоятельная работа)

- Комментарии в тексте программы
- Документирующие комментарии
- Встроенный HTML
- Примеры

Документирующие комментарии

- Специальный синтаксис комментариев
- Утилита javadoc



Документирующие комментарии

- Необходимость поддержки кода и документации ведет к необходимости их совмещения
- Преимущества поддержки на уровне языка
  - Единство стиля
  - Четкая структура
- Преимущества javadoc
  - Стандартное оформление
  - Просмотр в любом html-браузере
  - Удобная навигация

Demo: AnagramGame

Документирующие комментарии: пример

- Anagram Game (фрагмент кода)

```
/**
 * Gets the number of words in the library.
 * @return the total number of pairs in the library
 */
public static int getSize() {
    return WORD_LIST.length;
}
/**
 * Checks whether a guess for a word at the given index is correct.
 * @param idx index of the word guessed
 * @param userGuess the user's guess for the actual word
 * @return true if the guess was correct; false otherwise
 */
 public static boolean isCorrect(int idx, String userGuess) {
```

Теги Javadoc

- Утилита javadoc поддерживает некоторые специальные теги

```
@see – ссылка на другой класс, метод или поле

Например:
@see Window
@see java.awt.Window
@see java.awt.Window#isActive
```

Теги класса

- Специальный набор тегов может быть включен в документацию класса

```
@version – указание версии класса
@author – автор класса
@since – с какой версии продукта или
  библиотеки появился класс
```

Теги полей

- При документировании переменных можно использовать только встроенный HTML и @see

```
public class DocumentPrinter {
      /** Идентификатор класса.
        * @see #print()
        */
      public int id;
      public void print() {}
}
```

Теги методов

- При документировании методов имеется существенно больше возможностей

```
/** Посылает документ на печатающее устройство. Возвращает
   значение
  * {@code true} если документ успешно отослан.
  * @see Printer#getDefaultPrinter()
  * @param document – документ, предназначенный для печати.
  * @return {@code true} если документ успешно отослан;
  *         {@code false} в противном случае.
  * @throws IOException в случае ошибки ввода-вывода.
  * @throws PrinterException в случае ошибки печатающего
   устройства.
  * @deprecated рекомендуется использовать метод
  * {@code print(Document, Printer)}.
  */
  public boolean print(Document document) {…}
```

Теги методов

@param	Название параметра метода и комментарий к нему.
@return	Описание возвращаемого значения.
@throws	Тип бросаемого исключения и описание исключительной ситуации.
@deprecated	Данным тегом помечается не рекомендуемый к использованию метод.
@code	Ставится перед названиями классов, ключевых слов и т.п. – для применение стиля code.
@see	Ссылка на место в коде.

Теги методов

- Документирование параметров метода

```
/**
  * @param studID – уникальный идентификатор студента
  * @param discID – уникальный идентификатор дисциплины
  */
  public boolean isLearned(int studID, int discID){
        ...
  }
```

Теги методов

- Документирование возвращаемого значения
  - > Если метод не возвращает ничего (void), данный тег не используется
  - > Для **boolean** указывается, в каком случае формируется значение **true**, а в каком – **false**

```
/**
 * @return - {@code true} если дисциплина была изучена
студентом;
 *           {@code false} в противном случае.
 */
public boolean isLearned(int studID, int discID){
      ...
}
```

Теги методов

- Документирование исключений
  - > Тип исключения
  - > Условия возникновения исключения

```
/**
  * @throws ArgumentValueException при некорректных входных
  * параметрах (идентификатор меньше или равен нулю).
  */
 public boolean isLearned(int studID, int discID){
     ...
 }
```

Упражнения для самостоятельной работы

- Для программ предыдущего упражнения напишите документирующие комментарии в формате, поддерживаемом javadoc
- Убедитесь в правильной генерации html-документации для разработанных программ

Q&A