

# i=i++;

i=i++;

tmp = i;

i = i + 1;

i = tmp;

**Case 1:**

int x=7;

x=x++;

**Byte Code:**

```
0 bipush 7    //Push 7 onto stack
2 istore_1 [x] //Pop 7 and store in x
3 iload_1 [x]  //Push 7 onto stack
4 iinc 1 1 [x] //Increment x by 1 (x=8)
7 istore_1 [x] //Pop 7 and store in x
8 return      //x now has 7
```

**Case 2:**

int x=7;

x=++x;

**Byte Code**

```
0 bipush 7    //Push 7 onto stack
2 istore_1 [x] //Pop 7 and store in x
3 iinc 1 1 [x] //Increment x by 1 (x=8)
6 iload_1 [x]  //Push x onto stack
7 istore_1 [x] //Pop 8 and store in x
8 return      //x now has 8
```



# Обработка строк



## Обработка строк

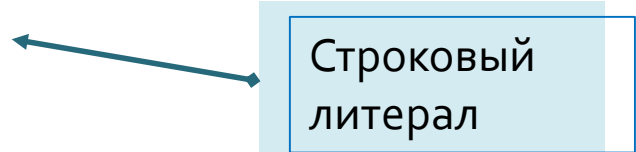
### Класс String

**Строка** – это последовательность символов, представляемая в Java как объект типа **String** (пакет *java.lang*).

Наиболее простой способ создания *строки* – использование *строкового литерала*:

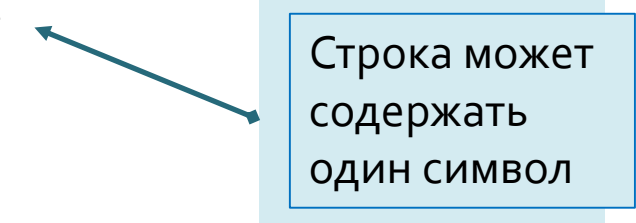
String *str* = “text”;

Строковый  
литерал



String *str\_1* = “H”;

Строка может  
содержать  
один символ



## Обработка строк

### Создание строки с помощью конструктора

<b>String()</b>	- пустая строка;
<b>String(String str)</b>	- строка как копия другой строки;
<b>String(byte[] ach)</b>	- строка 8-разрядных символов;
<b>String(char[] c)</b>	- строка, инициализированная массивом символов;

*Например,*

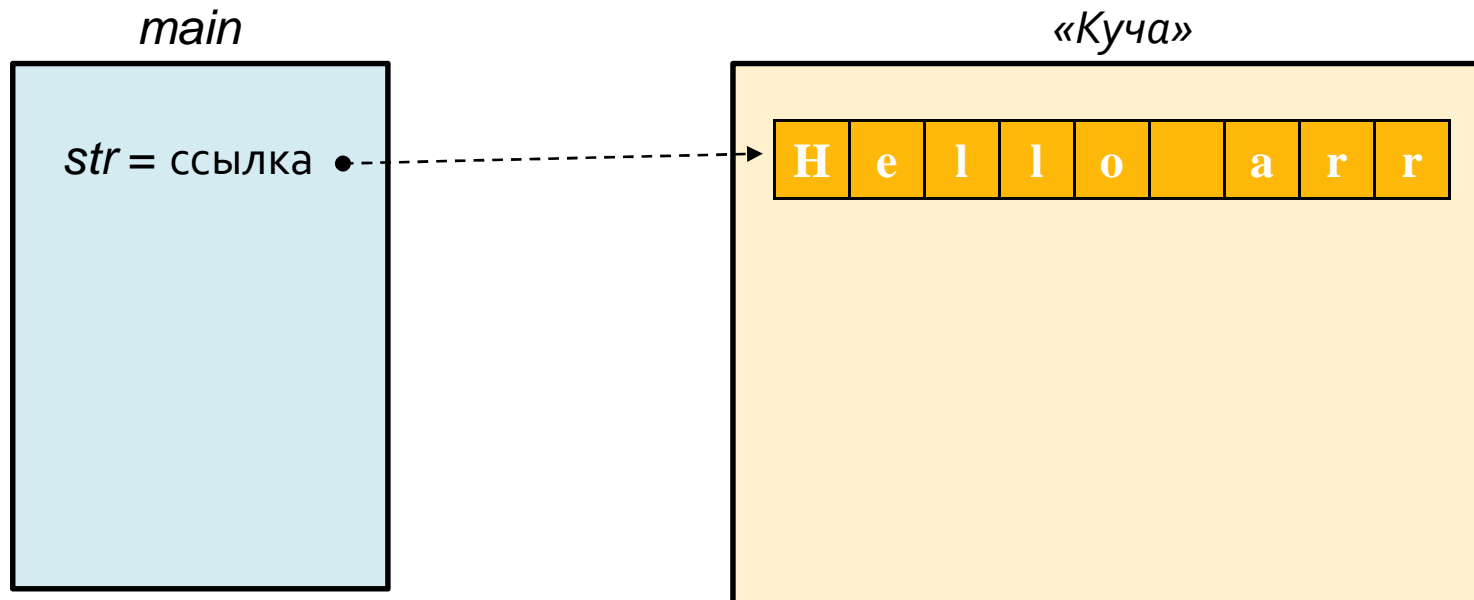
```
String s = new String();  
String s1 = new String(s);  
String s2 = new String("Student");
```

<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

## Обработка строк

Пример 1:

```
public static void main(String[] arg) {  
    char[] helloArray = {'H', 'e', 'l', 'l', 'o', ' ', 'a', 'r', 'r'};  
    String str = new String(helloArray);  
    System.out.println(str);  
}
```



## Обработка строк

### Методы класса String

<i>Прототип метода</i>	<i>Назначение метода</i>
length()	Определение длины строки
concat(String s) или «+»	Соединение двух строк
equals(Object ob)	Сравнение двух строк с учетом регистра
equalsIgnoreCase(String s)	Сравнение двух строк без учета регистра
substring(int n, int m)	Извлечение из строки подстроки длиной (m-n), начиная с позиции n
substring(int n)	Извлечение из строки подстроки, начиная с позиции n
valueOf(<значение>)	Преобразование переменной базового типа к строке
toUpperCase()	Преобразование всех символов строки в верхний регистр
toLowerCase()	Преобразование всех символов строки в нижний регистр
replace(char c1, char c2)	Замена в строке всех вхождений первого символа вторым символом
trim()	Удаление всех пробелов в начале и конце строки
charAt(int pos)	Получение символа по указанной позиции pos
getChars(<параметры>)	Получение строки символов в виде массива двухбайтных значений
indexOf(int ch)	Определить позицию первого вхождения указанного символа в строке
lastIndexOf(char c)	Определить позицию последнего вхождения указанного символа в строке

## Обработка строк

### Объединение строк

String str1 = "Learning ";

String str2 = "java!";

String str3 = str1.concat(str2);

String str4 = str1 + str2;

Первый способ

Второй способ

System.out.println(str3);

System.out.println(str4);

Вывод в консоли:

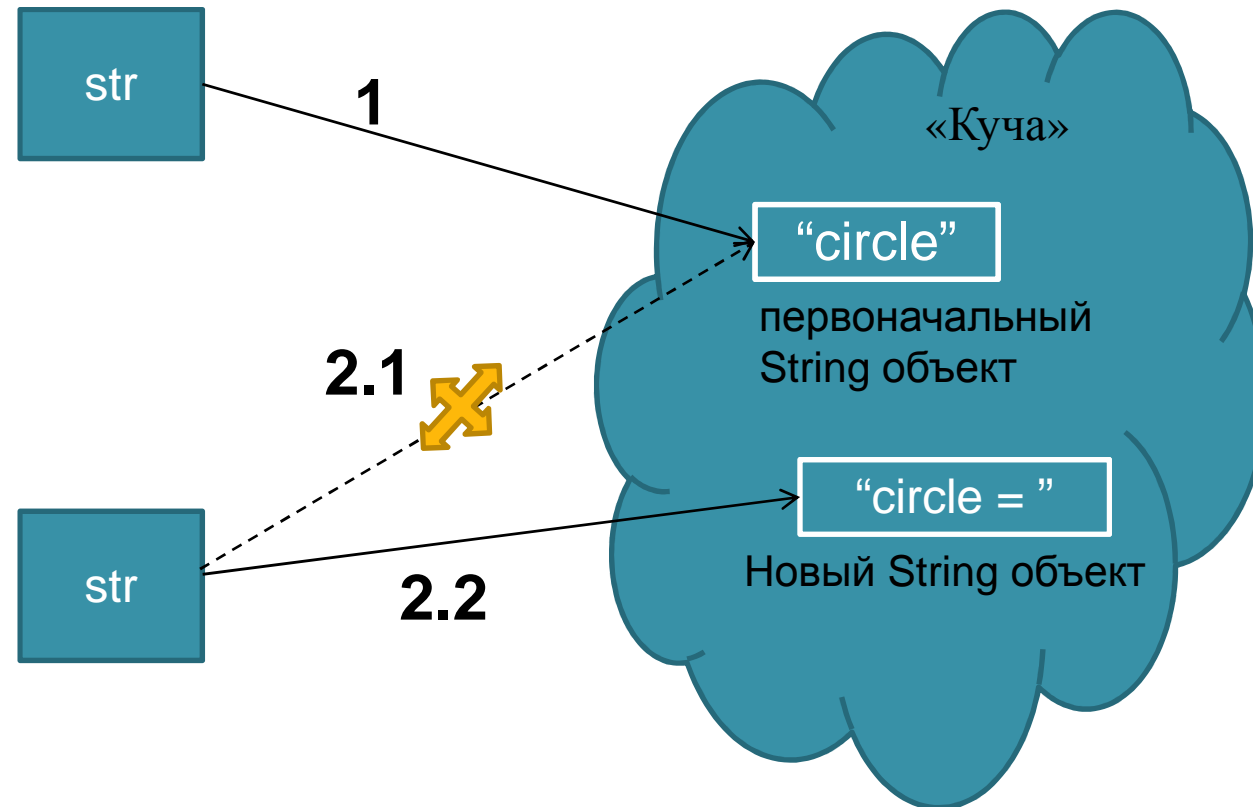
Learning java!

Learning java!

## Обработка строк

### Неизменяемость строк

1. String str = "circle";
2. str = str + " = ";





## Обработка строк

### Работа со ссылками на строки типа String

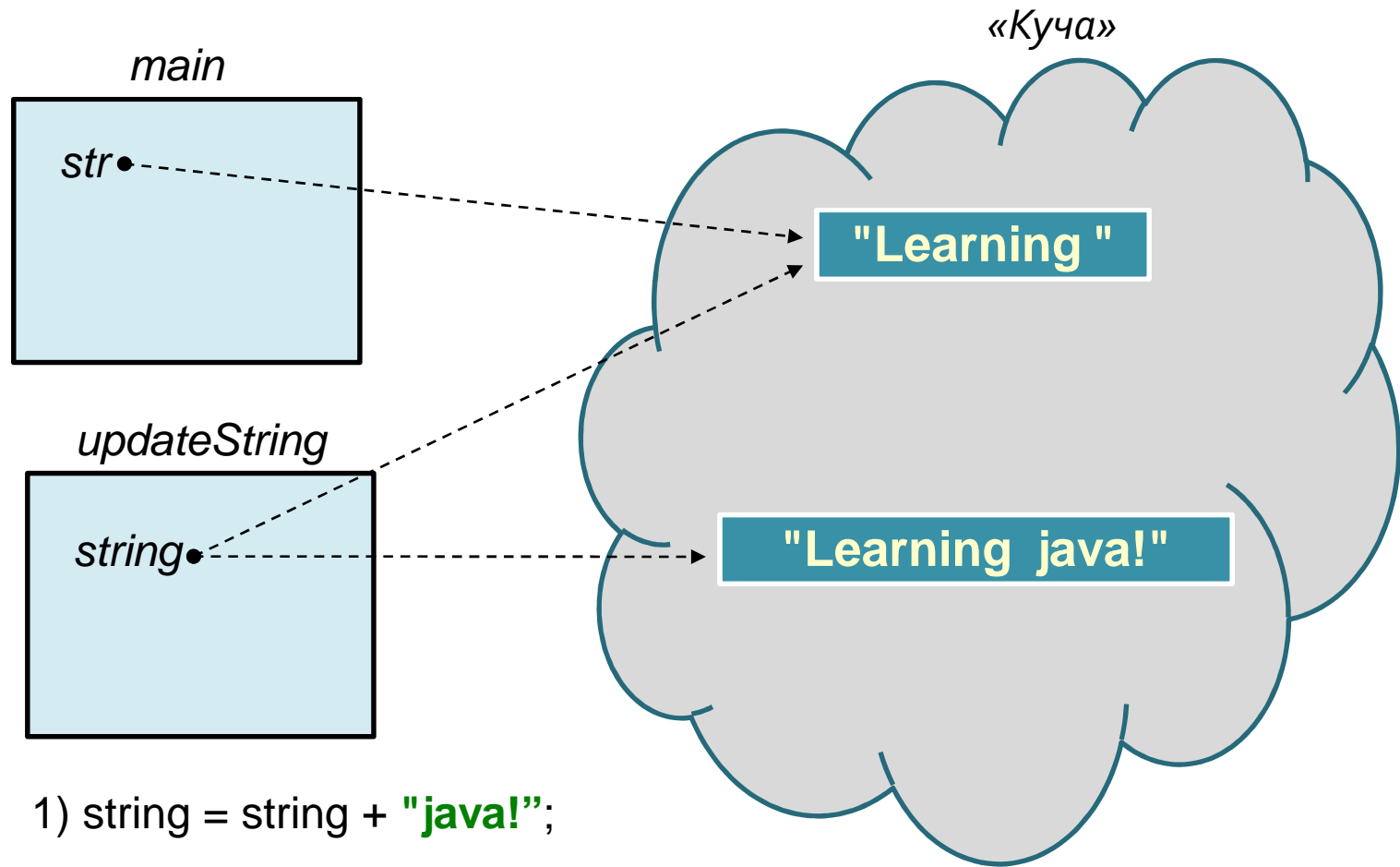
Пример 2:

```
public class ChangeString {  
    public static void main(String [] args) {  
        String str = "Learning";  
        updateString(str);  
        System.out.println(str);  
    }  
    static void updateString(String string) {  
        string = string + "java!";  
    }  
}
```

Вывод в консоли:

Learning

## Обработка строк



## Обработка строк

### Сравнение строк

#### Пример 3:

```
class EqualsNotEqualTo {  
    public static void main(String args[]) {  
        String s1 = "Привет";  
        String s2 = new String(s1);  
  
        System.out.println(s1 + " equals " + s2 + " -> " +  
                           s1.equals(s2));  
  
        System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));  
    }  
}
```

Использование new всегда приводит к созданию независимой копии строки

#### Вывод в консоли:

```
Привет equals Привет -> true  
Привет == Привет -> false
```

## Обработка строк

### Работа с индексами в строке

#### Пример 4:

```
String str = "Software And Hardware!";  
char aChar0 = str.charAt(0);  
char aChar9 = str.charAt(9);  
char aCharEnd = str.charAt( str.length() - 1);  
System.out.println(aChar0);  
System.out.println(aChar9);  
System.out.println(aCharEnd);
```

Вывод в консоли:

S  
A  
!

- Если *указанный индекс* не входит в границы строки, то будет ошибка типа **StringIndexOutOfBoundsException**

## Обработка строк

### Получение подстроки

#### Пример 5:

```
String str = "Software And Hardware!";
```

```
String substr1 = str.substring(13);
```

От указанного индекса до  
конца строки

```
String substr2 = str.substring(0, 8);
```

```
String substr3 = str.substring(13, 17);
```

Начиная с первого  
индекса и до второго  
индекса

```
System.out.println(substr1);
```

```
System.out.println(substr2);
```

```
System.out.println(substr3);
```

Вывод в консоли:

*Hardware!*

*Software*

*Hard*

## Обработка строк

### Поиск символа в строке

#### Пример 6:

```
String str = "Software And Hardware!"
```

```
int i1 = str.indexOf('a');
```

От начала и до первого  
встреченного

```
int i2 = str.lastIndexOf('a');
```

С конца и до первого  
встреченного

```
int i3 = str.indexOf('x');
```

```
System.out.println(i1);
```

```
System.out.println(i2);
```

```
System.out.println(i3);
```

Вывод в консоли:

5

18

-1


- Если указанного символа, то возвращается -1, а также можно искать от указанного индекса.

## Обработка строк

### Поиск подстроки в строке

Пример 6:

```
String str = "String in java is a sequence of characters java";  
int i1 = str.indexOf("java");  
int i2 = str.lastIndexOf("java");  
int i3 = str.indexOf("java", 20);  
System.out.println(i1);  
System.out.println(i2);  
System.out.println(i3);
```



Вывод в консоли:

10

44

44

## Обработка строк

### Пул общих строк

- ❑ Для уменьшения использования памяти виртуальной машины и оптимизации работы с часто используемыми строками в Java существует специальный механизм хранения строковых литералов в отдельной области памяти – пул общих строк (*string common pool*).
- ❑ Если существует два или более одинаковых строковых литерала, то для них в этой области выделяется место только для одного, но ссылки на этот объект могут быть присвоены любому количеству строковых переменных.



## Обработка строк

### Пример 7:

String s1 = "Hello";

String s2 = "Hello";

String s3 = s1;

String s4 = new String("Hello");

String s5 = new String("Hello");

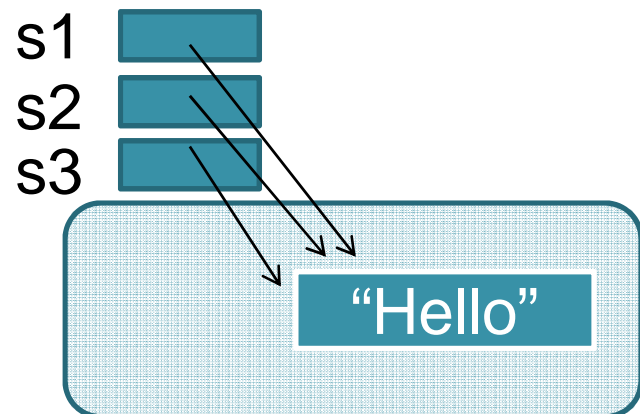
*// строковый литерал*

*// строковый литерал*

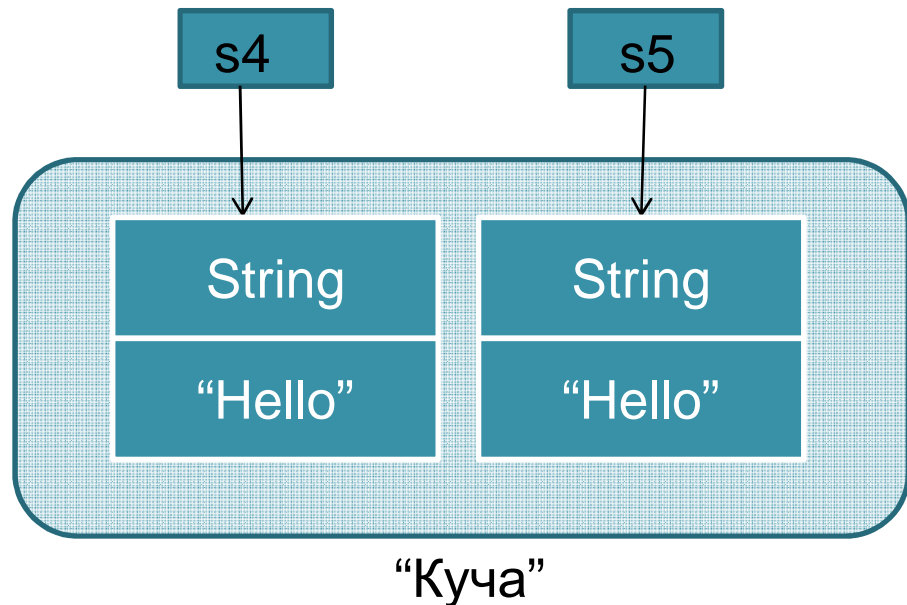
*// та же ссылка*

*// новый объект*

*// новый объект*



Пул строковых литералов



“Куча”

## Обработка строк

### Классы `StringBuilder` и `StringBuffer`

- ❑ Потеря эффективности работы с данными типа **String** при частой их модификации:

```
String str = "S0";  
for (int i = 1; i <= 6; i++) {  
    str += "m" + i;  
}  
System.out.println(str);
```

Выход: использование класса  
**`StringBuilder`** или **`StringBuffer`**

#### Объекты в «Куче»:

S0

S0m1

S0m1m2

S0m1m2m3

S0m1m2m3m4

S0m1m2m3m4m5

S0m1m2m3m4m5m6

## Обработка строк

- ❑ Класс **StringBuffer/StringBuilder** представляет расширяемые и доступные для изменений последовательности символов.

Отличие: Класс **StringBuffer** является потоко-безопасным, т.к. все методы синхронизированы.

*Конструкторы* класса **StringBuffer**:

**StringBuffer();** → пустой буфер размерностью 16

**StringBuffer(int size);** → буфер размерностью *size*

**StringBuffer(String obj);** → буфер размерностью 16 + длина *obj*

<https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html>

<https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html>

## Обработка строк

### Методы класса StringBuffer

<i>Прототип метода</i>	<i>Назначение метода</i>
setLength(int n)	Установить размер буфера для объекта
capacity()	Определить размер буфера объекта
append(...)	Добавить символы, значения базовых типов, массивы и строки
insert(...)	Вставить символ, объект или строку в указанную позицию
deleteCharAt(int index)	Удалить символ по указанному индексу
delete(int start, int end)	Удалить подстроку с указанными позициями
reverse()	Перевертывание строки символов

## Обработка строк

### Пример 8:

```
StringBuffer sb = new StringBuffer(10);  
sb.append("Hello...");  
char c = '!';  
sb.append(c);  
sb.insert(8, " Java");  
  
sb.delete(5, 8);  
  
System.out.println(sb);
```

*// добавить символ*  
*// вставить строку в*  
*// позицию 8*  
*// удалить подстроку с 5*  
*// до 8 позиции*

## Обработка строк

1) `StringBuilder sb = new StringBuilder(10);`

0	1	2	3	4	5	6	7	8	9

2) `sb.append("Hello...");`

0	1	2	3	4	5	6	7	8	9
H	e	l	l	o	.	.	.		

3) `sb.append('!');`

0	1	2	3	4	5	6	7	8	9
H	e	l	l	o	.	.	.	!	

4) `sb.insert(8," Java");`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
H	e	l	l	o	.	.	.		J	a	v	a	!	

5) `sb.delete(5,8);`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
H	e	l	l	o		J	a	v	a	!				

## Обработка строк

### Работа со ссылками на строки типа StringBuffer

Пример 9:

```
public class ChangeStringB {  
    public static void main(String [] args) {  
        StringBuffer str = new StringBuffer("Learning ");  
        updateString(str);  
        System.out.println(str);  
    }  
    static void updateString(StringBuffer string) {  
        string.append("java!");  
    }  
}
```

Вывод в консоли:

Learning java!

### ПЕРЕГРУЗКА МЕТОДОВ

- ❑ Java поддерживает *перегрузку методов* на базе методов с различной сигнатурой.
- ❑ Перегруженные методы имеют одно и тоже имя, а различны по количеству, типу и порядку следования аргументов, переданных в метод.
- ❑ Компилятор не учитывает тип возвращаемого значения при различении перегруженных методов.

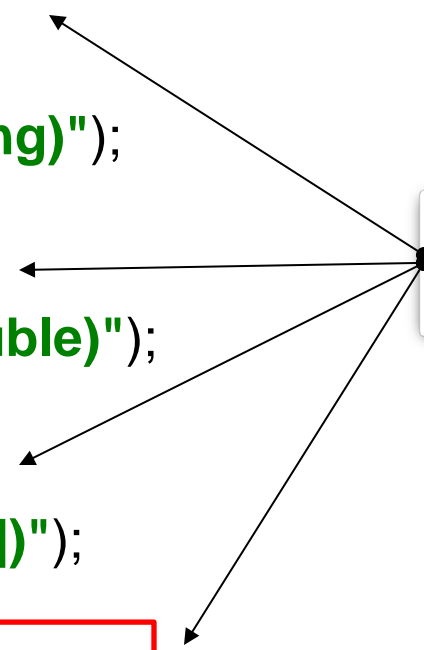


## Классы

Пример 9:

```
public void test(String s) {  
    s = "abcd";  
    System.out.println("test(String)");  
}  
public void test(double dd) {  
    System.out.println("test(double)");  
}  
public void test(int[] arr) {  
    System.out.println("test(int[])");  
}  
public double test(int i, double f) {  
    System.out.println("test(int, double)");  
    return i * f;  
}
```

Различные  
сигнатуры



## Классы

Пример 10:

```
public void test(String s) {  
    // ...  
}  
public void test(double d) {  
    // ...  
}  
public int test(double dd) {  
    // ...  
}  
public int test(int i, double f) {  
    // ...  
    return i;  
}
```



Одинаковые  
сигнатуры методов

### РАЗРЕШЕНИЕ ПЕРЕГРУЗКИ

- ❑ Определение метода, который будет вызываться по выражению вызова метода, включает в себя несколько этапов:
  - **Первый этап** - выяснить имя метода, который будет вызываться, и какой класс/интерфейс требуется проверяться для этого имени метода;
  - **Второй этап** - поиск типа и методов-кандидатов для применения ( т.е. те, которые могут быть корректно вызваны по заданным аргументам);
  - **Третий этап** – выбор из кандидатов метода, который фактически выполнится во время исполнения.

### Выбор метода из кандидатов

- ❑ **Первый шаг** – поиск совпадения типов аргумента и параметра;
- ❑ **Второй шаг** – использование приведения типов:
  - для примитивных типов – расширяющее;
  - для объектных – к ближайшему супертипу;
- ❑ **Третий шаг** – использование упаковки/распаковки;
- ❑ **Четвертый шаг** – использование методов переменной аргументности с упаковкой/распаковкой.

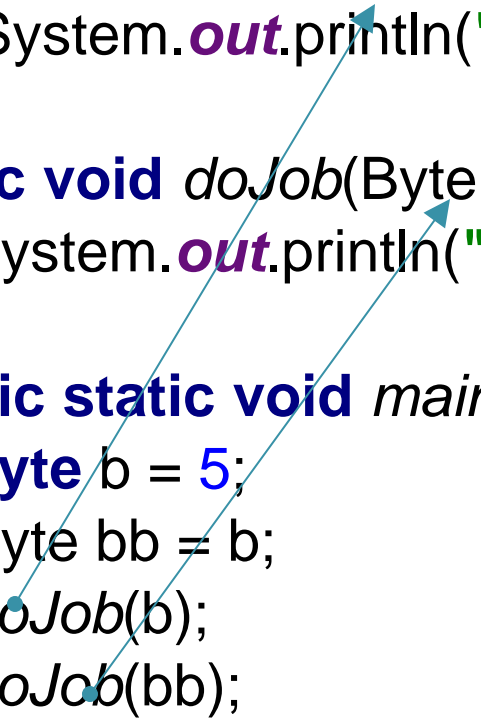


При разрешении перегрузки решающую роль играет тип ссылки аргумента, а не его фактический тип!

## Классы

Пример 11, разрешение на первом шаге:

```
public class Main {  
    static void doJob(byte b) {  
        System.out.println("byte");  
    }  
    static void doJob(Byte b) {  
        System.out.println("Byte");  
    }  
    public static void main(String[] args) {  
        byte b = 5;  
        Byte bb = b;  
        doJob(b);  
        doJob(bb);  
    }  
}
```



### Вывод консоли:

byte

Byte

## Классы

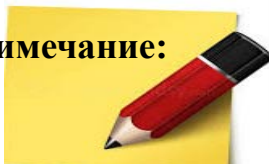
Пример 12, разрешение на втором шаге:

```
static void doJob(byte b) {   System.out.println("byte");   }  
static void doJob(int i) {     System.out.println("int");     }  
static void doJob(double b) {  System.out.println("double");  }  
public static void main(String[] args) {  
    short b = 10;  
    long x = b;  
    float d = b;  
    doJob(b);  
    doJob(x);  
    doJob(d);  
}
```

### Вывод консоли:

```
int  
double  
double
```

**Примечание:**



Расширяющие приведения для примитивных типов

## Классы

Пример 13, разрешение на втором шаге:

```
public class Main {  
    static void doJob(double i) {  
        System.out.println("double");  
    }  
    static void doJob(Byte b) {  
        System.out.println("Byte");  
    }  
    public static void main(String[] args) {  
        byte b = 5;  
        Byte bb = b;  
        doJob(b);  
        doJob(bb);  
    }  
}
```

**Вывод консоли:**

double  
Byte

## Классы

Пример 14, разрешение на третьем шаге:

```
public class Main {  
    static void doJob(int i) {  
        System.out.println("int");  
    }  
    static void doJob(Double b) {  
        System.out.println("Double");  
    }  
    public static void main(String[] args) {  
        byte b = 5;  
        Byte bb = b;  
        doJob(b);  
        doJob(bb);  
    }  
}
```

Вывод консоли:

int  
int

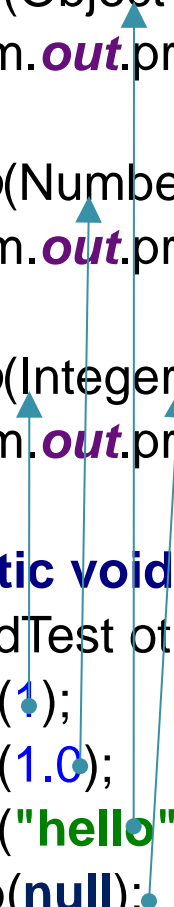
Использование автораспаковки, а затем расширяющего приведения, т.к. для типа **Byte** не было найдено соответствующего супертипа



## Классы

Пример 15, разрешение на третьем шаге:

```
public class OverloadTest {  
    void dojob(Object o){  
        System.out.println("Object");  
    }  
    void dojob(Number i){  
        System.out.println("Number");  
    }  
    void dojob(Integer i){  
        System.out.println("Integer");  
    }  
    public static void main(String[] args) {  
        OverloadTest ot = new OverloadTest();  
        ot.dojob(1);  
        ot.dojob(1.0);  
        ot.dojob("hello");  
        ot.dojob(null);  
    }  
}
```



Примечание:



Упаковка примитивных типов в типы классов-оберток, а затем приведение к ближайшему супертипу

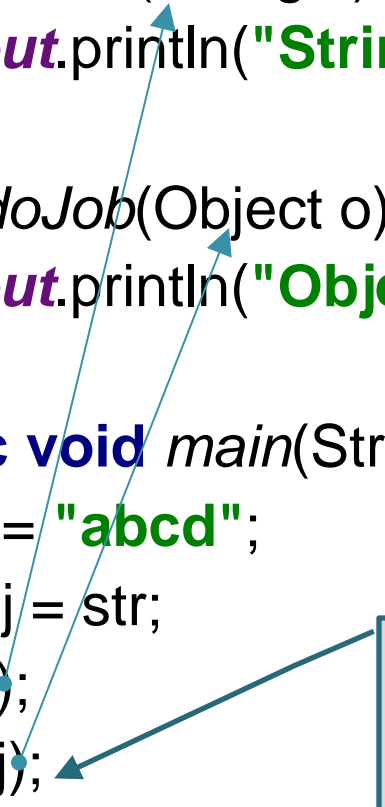
Вывод консоли:

Integer  
Number  
Object  
Integer

## Классы

Пример 16, разрешение на втором шаге:

```
public class Test {  
    static void doJob(String s) {  
        System.out.println("String");  
    }  
    static void doJob(Object o) {  
        System.out.println("Object");  
    }  
    public static void main(String[] args) {  
        String str = "abcd";  
        Object obj = str;  
        doJob(str);  
        doJob(obj);  
    }  
}
```



Вывод консоли:

String  
Object

Решающую роль  
играет тип ссылки  
(**Object**), а не тип  
объекта (**String**)

## Классы

Пример 17, разрешение на четвертом шаге:

```
public static void doJob(String s1) {  
    System.out.println("String");  
}  
  
public static void doJob(String s1, String s2) {  
    System.out.println("String, String");  
}  
  
public static void doJob(String s, String... strings) {  
    System.out.println("String, String...");  
}  
  
public static void main(String[] args) {  
    doJob("hi");  
    doJob("hi", "hi");  
    doJob("hi", "hi", "hi");  
}
```

### Вывод консоли:

```
String  
String, String  
String, String...
```

Сработало при  
отсутствии остальных  
вариантов

## Классы

Пример 18, разрешение на четвертом шаге:

```
public static void doJob(String... strings) {  
    System.out.println("String...");  
}  
public static void doJob(String s1, String s2) {  
    System.out.println("String, String");  
}  
public static void doJob(String s, String... strings) {  
    System.out.println("String, String...");  
}  
public static void main(String[] args) {  
    doJob("hi");  
    doJob("hi", "hi", "hi");  
}
```

Нет смысла в таком описании, т.к. второй параметр переменной аргументности может включать в себя и первый

Неопределенность, т.к. подходит и первый и третий методы для вызова

## Классы

Пример 19, разрешение на различных шагах:

```
public class MethodResolutionTest {  
    public static void doJob(Object obj1, Object obj2) {  
        System.out.println("Object, Object");  
    }  
    public static void doJob(String s, Object... objects) {  
        System.out.println("String, Object...");  
    }  
    public static void main(String[] args) {  
        doJob("hi", "hi");  
        doJob(new Object(), "hi");  
        doJob("hi", "hi", "hi");  
    }  
}
```

Параметр переменной  
арности до  
четверного шага  
воспринимается как  
одномерный массив

### Вывод консоли:

```
Object, Object  
Object, Object  
String, Object...
```