ПРИМИТИВНЫЕ ТИПЫ JAVA

СПБАУ 03.09.2014

ПРИМИТИВНЫЕ ТИПЫ

void	«пустой» тип
boolean (1 бит)	логический тип
char (2 байта)	символьный тип
byte (1 байт)	целочисленные типы
short (2 байта)	
int (4 байта)	
long (8 байт)	
float (4 байта)	типы с плавающей точкой
double (8 байт)	

- Простые типы
- Зарезервированные ключевые слова языка
- Передаются по значению

```
int i = 4;
int j = i;
j++;
// j -> 5, i -> 4
```

ССЫЛОЧНЫЕ ТИПЫ

- Все остальные, кроме перечисленных на предыдущем слайде
- Являются объектами
- Передаются по ссылке

```
Dog thisDog = new Dog ("Sparky");
Dog thatDog = thisDog ;
thatDog.setName ("Wolfie");
/*
* thatDog.getName() -> "Wolfie"
* thisDog.getName() -> "Wolfie"
*/
```

ССЫЛОЧНЫЕ ТИПЫ

Все классы наследуются от java.lang.Object

Методы в классе java.lang.Object:

- String toString() – все объекты можно приводить к строке (как следствие – печатать на экран)

- boolean equals(Object obj) – используется для сравнения объектов
- int hashCode() – хеш-код объекта

…

boolean

ПРИМИТИВНЫЕ ТИПЫ JAVA

ЛОГИЧЕСКИЕ ЗНАЧЕНИЯ

- Литералы: false, true
- Любое сравнение имеет тип boolean:
  - < > ==
  - <= >= !=
- Нет преобразования между boolean и другими примитивными типами. Как следствие нельзя писать

```
int x = 3;
if (x) {
        ...
}
```

НЕЛЬЗЯ

ЛОГИЧЕСКИЕ ОПЕРАЦИИ

- and && & &=
- or || | |=
- xor ^ ^=
- not !

- && и || - вычисление по сокращенной схеме
- & и | - вычисление по полной схеме

ПРИМЕРЫ

```
boolean a = true;
boolean b = false;
boolean c = (a ^ b) == (a != b);
boolean d = c ^= !b || a;

int m = 0;
int n = 10;
if (m != 0 & n / m >= 1) {
      System.out.println(" condition is true ");
} else {
      System.out.println(" condition is false ");
}
```

java.lang.Boolean

Класс-обертка для boolean

- boolean parseBoolean(String)
- String toString(boolean)

- boolean b = Boolean.parseBoolean(“false”);
- String str = Boolean.toString(b);

char

ПРИМИТИВНЫЕ ТИПЫ JAVA

СИМВОЛЬНЫЕ ЗНАЧЕНИЯ

- char — 16 бит, беззнаковый (0 .. $2^{16} - 1$)
- Представляет номер символа в кодировке Unicode
- Литералы:
  - символ в одинарных кавычках: 'a'
  - шестнадцатеричный код символа: '\u78bc'
  - спецпоследовательности: '\t', '\n', '\r', '\"', '\'', '\\'
- Свободно конвертируется в числовые типы и обратно

java.lang.Character

Класс-обертка для char

- boolean isLowerCase(char)
- boolean isUpperCase(char)
- boolean isDigit(char)
- boolean isWhiteSpace(char)
- boolean isLetter(char)
- char toLowerCase(char)
- char toUpperCase(char)
- int getNumericValue(char)

- char c = Character.toLowerCase('a')

Целочисленные типы

ПРИМИТИВНЫЕ ТИПЫ JAVA

ДИАПАЗОНЫ ЗНАЧЕНИЙ

Тип	Бит	Диапазон
byte	8	-128 .. +127
short	16	-2^{15} .. $+2^{15}-1$
int	32	-2^{31} .. $+2^{31}-1$
long	64	-2^{63} .. $+2^{63}-1$

Размер фиксирован и одинаков для всех платформ

Все типы знаковые, беззнаковых вариантов нет (не было до появления Java8 ☺)

Десятичное число: 123
Восьмеричное число: 0123
Шестнадцатеричное число: 0x123
Двоичное число: 0b101 (с Java 7)
С подчеркиванием: 123_456_789 (с Java 7)
С суффиксом L для long

ОПЕРАЦИИ

сложение	+	+=
вычитание	-	-=
умножение	*	*=
деление	/	/=
остаток	%	%=
инкремент	++	
декремент	--	

Деление целочисленное
(для целых типов)
a == (a / b) * b + (a % b)

Деление на ноль — исключительная ситуация, бросается ArithmeticException

Переполнение не является исключительной ситуацией, лишние старшие биты просто выкидываются

КЛАССЫ-ОБЕРТКИ

Классы:

- java.lang.Byte
- java.lang.Short
- java.lang.Integer
- java.lang.Long

Методы в классах:

- MIN_VALUE
- MAX_VALUE
- toString(*typename*)
- parse*Typename*(String)

Вещественные типы

ПРИМИТИВНЫЕ ТИПЫ JAVA

ЛИТЕРАЛЫ

- Обычная запись: -1.234
- Экспоненциальная запись: -123.4e-2 ($-123.4 \cdot 10^{-2}$)
- Шестнадцатеричная запись: 0xFFFFpFF ($FFFF \cdot 2^{FF}$)
- С суффиксом типа:
  - 38f
  - 3e19d
  - 123.4e-2f
  - 444.444d

ОСОБЫЕ СЛУЧАИ

- Деление положительного числа на 0 дает +∞
- Деление отрицательного числа на 0 дает −∞
- Деление 0 на 0 дает NaN

- Переполнение дает +∞ или −∞, в зависимости от направления

- Любая арифметическая операция с NaN дает NaN
- NaN != NaN

Преобразование типов

ПРИМИТИВНЫЕ ТИПЫ JAVA

НЕЯВНОЕ ПРЕОБРАЗОВАНИЕ ТИПОВ

Неявное преобразование возможно в случаях:

- Преобразование целочисленных типов в более емкие (byte → short → int → long)
- Преобразование char в int и long
- Преобразование целочисленные типов в типы с плавающей точкой (возможна потеря точности)

ЯВНОЕ ПРЕОБРАЗОВАНИЕ ТИПОВ

- Операторы приведения типа (typename):
  - (int), (char)…
- При приведении более емкого целого типа к менее емкому старшие биты просто отбрасываются
- При приведении типа с плавающей точкой к целому типу дробная часть отбрасывается (никакого округления)
- Слишком большое дробное число при приведении к целому превращается в MAX_VALUE или MIN_VALUE
- Слишком большой double при приведении к float превращается в Float.POSITIVE_INFINITY или Float.NEGATIVE_INFINITY

АВТОМАТИЧЕСКОЕ РАСШИРЕНИЕ

- При вычислении выражения (a @ b) аргументы a и b преобразовываются в числа, имеющие одинаковый тип:
  - если одно из чисел double, то в double;
  - иначе, если одно из чисел float, то в float;
  - иначе, если одно из чисел long, то в long;
  - иначе оба числа преобразуются в int.
- Следствие: Арифметическое выражение над byte, short или char имеет тип int, поэтому для присвоения результата обратно в byte, short или char понадобится явное приведение типа

НЕЯВНОЕ ПРИВЕДЕНИЕ ТИПА С ПОТЕРЕЙ ДАННЫХ

- Сокращенная запись var @= expr раскрывается в var = (*typename*) (var @ (expr))

@ - здесь любая операция (+, -, *…)

- Неявно срабатывает приведение типа, в том числе с потерей данных

Autoboxing (автоупаковка)

ПРИМИТИВНЫЕ ТИПЫ JAVA

Boxing & Unboxing

- Boxing – автоматическое преобразование примитивного типа в соответствующий объект класса-обертки:

```
Integer a = 5;
```

- UnBoxing – обратное преобразование

```
Integer a = 5;
int b = a;
```

ВАЖНЫЕ ЗАМЕЧАНИЯ

```
public static void test1() {
	int a = 1;
	int b = 1;

	System.out.println("---------------");
	System.out.println(a > b);
	System.out.println(a < b);
	System.out.println(a == b);
}
//	false false true
```

ВАЖНЫЕ ЗАМЕЧАНИЯ

```
public static void test1() {
	Integer a = 1;
	Integer b = 1;

	System.out.println("---------------");
	System.out.println(a > b);
	System.out.println(a < b);
	System.out.println(a == b);
}
//	false false true
```

ВАЖНЫЕ ЗАМЕЧАНИЯ

```
public static void test1() {
        Integer a = 1000;
        Integer b = 1000;

        System.out.println("---------------");
        System.out.println(a > b);
        System.out.println(a < b);
        System.out.println(a == b);
}
//      false false false
```

ВАЖНЫЕ ЗАМЕЧАНИЯ

```
public static void test1() {
        Integer a = new Integer(1);
        Integer b = new Integer(1);

        System.out.println("---------------");
        System.out.println(a > b);
        System.out.println(a < b);
        System.out.println(a == b);
}
//      false false false
```

ПРОВЕРКА НА РАВЕНСТВО

В Java есть два способа сравнить объекты на равенство, == и метод equals.

- == используется для примитивных типов.
- Для объектов == это исключительно сравнение ссылок!

- Для остального надо использовать метод equals.
  Кроме того, метод hashCode служит (теоретически) для той же цели. Хорошим тоном считается переопределять его, если вы переопределили equals.
- Золотое правило сравнения:
  Если после инициализации неких объектов a и b выражение a.equals(b) вернёт true, то a.hashCode() должен быть равен b.hashCode().

КАК НЕ НУЖНО ПИСАТЬ

НЕЛЬЗЯ

```
long sum = 0;
for (Integer j = 0; j < 1000; j++) {
     for (Integer i = 0; i < 1000; i++) {
          sum += j*1000+i;
     }
}
```

Работает в 10! раз дольше, чем с int

Передача аргументов в методы

ПРИМИТИВНЫЕ ТИПЫ JAVA

ПЕРЕДАЧА АРГУМЕНТОВ

- Передача по ссылке подразумевает передачу ссылки на объект. В этом случае реализация метода потенциально может модифицировать переданный объект (например, вызвав метод, изменяющий состояние объекта).
- В случае передачи по значению параметр копируется. Изменение параметра не будет заметно на вызывающей стороне.
- В Java объекты всегда передаются по ссылке, а примитивы - по значению

ПРИМЕР

```
class Links1 {
	public static void foo(int x) {
		x = 3;
	}

	public static void main(String[] args) {
		int x = 1;
		foo(x);
		System.out.println(x);
	}
}
// 1
```

ПРИМЕР

```
class Links2 {
	public static void foo(Integer x) {
		x = 3;
	}

	public static void main(String[] args) {
		Integer x = 1;
		foo(x);
		System.out.println(x);
	}
}
// 1
```

ПРИМЕР

```
class Links3 {
	public static void foo(String x) {
		x = “3”;
	}

	public static void main(String[] args) {
		String x = “1”;
		foo(x);
		System.out.println(x);
	}
}
// 1
```

ПРИМЕР

```
class Links4 {
	public static void foo(String x) {
		x = new String(“3”);
	}

	public static void main(String[] args) {
		String x = “1”;
		foo(x);
		System.out.println(x);
	}
}
// 1
```

ПРИМЕР

```
class Links5 {
	public static void foo(Point p) {
		p.x = 3;
	}

	public static void main(String[] args) {
		Point p = new Point(0, 0);
		foo(p);
		System.out.println(p.x);
	}
}
// 3
```

Java

НЕ примитивные типы

СПБАУ 26.02.2014

String

НЕ ПРИМИТИВНЫЕ ТИПЫ JAVA

СТРОКИ

- Последовательность символов произвольной длины
- Класс java.lang.String
- Не то же, что массив символов
- Никаких нулевых символов в конце, длина хранится отдельно
- Строки не изменяемы!

- String hello = " Hello";
- String specialChars = "\r\n\t\"\\ \u0101\u2134\u03ff ";
- char[] charArray = {'a', 'b', 'c'};
- String string = new String (charArray);

ДОСТУП К СОДЕРЖИМОМУ

- int length()
- char charAt(int index)
- char[] toCharArray()

- String substring(int beginIndex)
- String substring(int beginIndex, int endIndex)

СРАВНЕНИЕ СТРОК

- Оператор == сравнивает ссылки, а не содержимое строки

- boolean equals(Object anObject)
- boolean equalsIgnoreCase(String anotherString)

- int compareTo(String anotherString)
- int compareToIgnoreCase(String anotherString)

ОПЕРАЦИИ

- boolean startsWith(String prefix)
- boolean endsWith(String suffix)
- int indexOf(String str)
- int lastIndexOf(String str)

- String trim()
- String replace(char oldChar, char newChar)
- String toLowerCase()
- String toUpperCase()

КОНКАТЕНАЦИЯ СТРОК

- String concat(String str)
- Оператор + (работает, как StringBuilder)

```
String helloWorld = " Hello " + " World!";
```

- java.lang.StringBuilder

```
StringBuilder buf = new StringBuilder();
buf.append ("Hello");
buf.append ("World");
buf.append ('!');
String result = buf.toString()
```

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

- Регулярные выражения поддерживаются в стандартной библиотеке Java

- boolean matches(String regex)
- String[] split(String regex)
- String replaceAll(String regex, String replacement)
- String replaceFirst(String regex, String replacement)

ПРИМЕРЫ

```
String str = "a, b, c,d, e";
String[] items = str.split (", *");
// items -> {"a", "b", "c", "d", "e"}

String str = "(aa)(bb)(cccc)";
String regex = "\\(([^)]*)\\)";
String result = str.replaceAll (regex , "$1");
// result -> "aabbcccc"
```