# Лекция 10. Пакет java.lang

# Классы-обертки (Wrapper Classes)

Во многих случаях бывает предпочтительней работать именно с объектами, а не примитивными типами. Так, например, при использовании коллекций, просто необходимо значения примитивных типов каким-то образом представлять в виде объектов. Для этих целей и предназначены так называемые классы-обертки. Для каждого примитивного типа Java существует свой класс обертка.

Объект типа класса-обертки является неизменяемым (то есть, для изменения значения необходимо создавать новый объект), а сам класс является final, что делает невозможным наследование от него.

Все классы-обертки реализуют интерфейс ***java.io.Serializable***, поэтому объекты данных типов могут быть разложены в набор байт и вновь собраны из них. Все классы-обертки содержат статическое поле TYPE типа Class, соответствующий оборачиваемому примитивному типу. Так же классы-обертки содержат статические методы для обеспечения удобного манипулирования соответствующими примитивными типами, например преобразование к строковому виду.

|  |  |
| --- | --- |
| **Класс-обертка** | **Примитивный тип** |
| Byte | byte |
| Short | short |
| Integer | int |
| Long | long |
| Float | float |
| Double | double |
| Character | char |
| Boolean | boolean |

*(!) При этом классы обертки числовых типов (Byte, Short, Integer, Long, Float, Double) наследуются от общего родителя – класса Number. В нем содержится код, общий (часть реализована посредством абстрактных методов) для всех классов-оберток числовых типов – получение значения примитивного значения в виде byte, short, int, long, float и double.*

Все классы-обертки реализуют интерфейс Comparable. Number реализует интерфейс java.io.Serializable. Все классы-обертки числовых типов имеют свой переопреленный метод ***equals(Object)***, сравнивающий примитивные значения объектов.

*(!) Стоит заметить, что результат выполнения строчки кода (new Integer(1)).equals(new Byte(1)) будет* ***false****, хотя сами примитивные значения равны. Такой результат получается потому, что во всех таких классах, метод equals() определен таким образом, что сначала производится проверка, совпадают ли типы (классы) значений, и если нет – сразу возвращается false.*

# Integer

Наиболее часто используемые статические методы:

* ***public static int parseInt(String s)*** преобразует в int значение переданную строку, представляющую десятичную запись целого числа;
* ***public static int parseInt(String s, int radix)*** преобразует в int значение строку, представляющую запись целого числа в системе счисления radix.

*(!) Оба этих метода могут возбуждать исключение NumberFormatException, если строка, переданная на вход, содержит нецифровые символы.*

Не следует путать эти методы, с другой парой похожих методов: public static Integer valueOf(String s) и public static Integer valueOf(String s,int radix). Эти методы выполняют аналогичную работу, только результат представляют в виде объекта.

Существует так же два конструктора для создания экземпляров в класса: Integer(String s) принимает в качестве параметра строку, представляющую значение; и Integer(int i) принимает значение оборачиваемого примитивного типа

* ***pubic static String toString(int i)*** используется для преобразования значения типа int в строку.

Ниже перечислены методы, преобразующие int в строковое восьмеричное, двоичное и шестнадцатеричное представление:

* ***pubic static String toOctalString(int i)*** – восьмеричное;
* ***pubic static String toBinaryString(int i)*** – двоичное;
* ***pubic static String toHexString(int i)*** – шестнадцатиричное.

Имеется так же две статические константы:

* ***Integer***.MIN\_VALUE – минимальное int значение;
* ***Integer***.MAX\_VALUE – максимальное int значение.

**Остальные классы-обертки предложены для самостоятельного ознакомления.**

Оставшиеся классы-обертки числовых типов – это Float и Double. Помимо описанного для целочисленных примитивных типов, дополнительно содержат определения следующих констант:

* ***NEGATIVE***\_INFINITY - отрицательная бесконечность;
* ***POSITIVE***\_INFINITY - положительная бесконечность;
* ***NaN*** – не числовое значение (неопределенность, комплексное число и т.д.)

Кроме того, несколько иначе трактуется значение MIN\_VALUE: вместо наименьшего значения, оно представляет минимальное положительное (строго > 0) значение, которое может быть представлено соответствующим примитивным типом и, соответственно, классом-оберткой.

# Character

Реализует интерфейс Comparable и имеет единственный конструктор, принимающий char в качестве параметра. Кроме стандартных методов equals(), hashCode(), toString() из нестатических, содержит только два метода: ***public char charValue()*** (возвращает обернутое значение) и ***char public int compareTo(Character)*** (сравнивает обернутые значения char как числа, то есть возвращает значение).

Также представлен набор статических методов:

* ***public static boolean isLetter(char c)*** проверяет, является ли char буквой;
* ***public static boolean isDigitOrLetter(char c)*** проверяет, является ли char цифрой или буквой;
* ***public static boolean isIdentifierStart(char c)*** проверяет, является ли символ подходящим для того, что бы с него начиналось наименование переменной.

# Boolean

Представляет класс-обертку для примитивного типа boolean. Реализует интерфейс java.io.Serializable и во всем напоминает аналогичные классы-обертки. Для получения примитивного типа используется метод booleanValue().

# Math

Класс Math состоит из набора статических методов, производящих наиболее часто использующиеся математические вычисления и двух констант, имеющих особое значение в математике – это число PI и exp. Часто этот класс еще называют классом-утилитой (Utility class).

Так как все методы класса статические нет необходимости создавать экземпляр этого класса, поэтому он и не имеет открытого конструктора. Нельзя так же и унаследовать этот класс, поскольку он объявлен с атрибутом final. В таблице 3 приведен набор методов класса Math.

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| static double abs(double a) | возвращает абсолютное значение типа double |
| static float abs(float a) | возвращает абсолютное значение типа byte |
| static int abs(int a) | возвращает абсолютное значение типа int |
| static long abs(long a) | возвращает абсолютное значение типа long |
| static double acos(double a) | возвращает значение арккосинуса угла в диапазоне от 0 до pi |
| static double asin(double a) | возвращает значение арксинуса угла в диапазоне от pi/2 до pi/2 |
| static double atan(double a) | вернет значение арктангенса угла в диапазоне от pi/2 до pi/2 |
| static double ceil(double a) | возвращает наименьшее целое число, которое больше a |
| static double floor(double a) | возвращает целое число которое меньше a |
| static double cos(double a) | возвращает косинус угла |
| static double sin(double a) | возвращает косинус угла |
| static double tan(double a) | возвращает тангенс угла |
| static double exp(double a) | возвращает e в степени числа a |
| static double log(double a) | возвращает натуральный логарифм числа a |
| static double max(double a, double b) | возвращает наибольшее из двух чисел типа double |
| static float max(float a, float b) | возвращает наибольшее из двух чисел типа double |
| static long max(long a, long b) | возвращает наибольшее из двух чисел типа long |
| static int max(int a, int b) | возвращает наибольшее из двух чисел типа int |
| static double min(double a, double b) | возвращает наименьшее из двух чисел типа double |
| static float min(float a, float b) | возвращает наименьшее из двух чисел типа double |
| static long min(long a, long b) | возвращает наименьшее из двух чисел типа long |
| static int min(int a, int b) | возвращает наименьшее из двух чисел типа int |
| static double pow(double a, double b) | возвращает а в степени b |
| static double random() | возвращает случайное число в диапазоне от 0.0 до 1.0 |
| static double rint(double a) | возвращает int число, ближайшее к a |
| static long round(double a) | возвращает значение типа long ближайшее по значению к а |
| static long round(double a) | возвращает значение типа long ближайшее по значению к а |
| static double sqrt(double a) | возвращает положительный квадратный корень числа a |
| static double toDegrees(double angrad) | преобразует значение угла из радианов в градусы |
| static double toRadians(double angdeg) | преобразует значение угла из градусов в радианы |

# String

Класс представления строк. Предназначен для хранения не модифицируемых строк. После того как создан экземпляр этого класса, строка уже не может быть модифицирована.

Для создания объекта String можно использовать различные варианты конструкторов. Наиболее простой способ создать строку – это присвоение текста в кавычках переменной типа String: String abc = "abc". Однако можно использовать и конструктор, который получает в качестве параметра строковый литерал: String s = new String("immutable").

На первый взгляд, эти варианты создания строк отличаются только синтаксисом. На самом каждый строковый литерал имеет внутреннее представление, как экземпляр класса String. Классы в Java могут иметь целый набор таких строк и, когда класс компилируется, экземпляр представляющий литерал, добавляется в этот набор. Однако, если такой литерал уже имеется в программе, т.е. уже представлен в наборе строковых объектов, то новый экземпляр (фактически копирующий уже существующий) создан не будет. Вместо этого будет создана ссылка на уже имеющийся объект. Т.к. строки не являются модифицируемыми объектами, то это не нанесет никакого урона другим фрагментам программы. С другой стороны, если объект-строка создается с помощью явного вызова String-конструктора, то даже если эти строки будут совершенно идентичными, экземпляры класса String будут отличаться. В объекте String определен метод equals() который сравнивает две строки на предмет идентичности.

В Java для строк определен оператор конкатенации «+». В классе String так же определен метод ***public String concat(String s)***, который возвращает новый объект-строку, дополненный справа строкой s. Следует еще раз обратить внимание, что строки являются не модифицируемыми объектами. И если используется оператор конкатенации или какой-либо вспомогательный метод класса String то изменения строки не произойдет, а будет создан новый экземпляр класса String. Так же следует обратить на использование в методах параметров типа String. Не смотря на то, что String является объектом и передается в метод по ссылке, String не модифицируемый объект и все изменения в методе не повлекут изменений исходного объекта.

Преобразование строки в последовательность байтов (восьмибитовые символы) производится с методами ***getBytes(),*** которыйвозвращает последовательность байт, в принятой по умолчанию кодировке, ***getBytes(String encoding),*** которыйвозвращает последовательность байт, в кодировке encoding; Для выполнения обратной операции (преобразования байтов в строку) необходимо создать новый объект-строку с помощью конструкторов ***String(byte[] bytes и*** ***String(byte[] bytes,String enc).***

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| int compareTo(String anotherString) | лексиграфическое сравнение строк |
| int compareToIgnoreCase(String str) | лексиграфическое сравнение строк без учета регистра символов |
| boolean contains(CharSequence s) | проверяет, входит ли указанная последовательность символов в строку |
| boolean endsWith(String suffix) | проверяет завершается ли строка указанным суффиксом |
| boolean startsWith(String suffix) | проверяет, начинается ли строка с указанного префикса |
| boolean startsWith(String prefix, int toffset) | проверяет, начинается ли строка в указанной позиции с указанного префикса |
| byte[] getBytes() | возвращает байтовое представление строки |
| int indexOf(String str) | поиск первого вхождения подстроки в строку |
| int indexOf(String str, int fromIndex) | поиск первого вхождения подстроки в строку с указанной позиции |
| int lastIndexOf(String str) | поиск последнего вхождения строки |
| int lastIndexOf(String str, int fromIndex) | поиск последнего вхождения строки с указанной позиции |
| String replace(char oldChar, char newChar) | замена в строке одного символа на другой |
| String substring(int beginIndex, int endIndex) | возвратить подстроку как строку |
| String toLowerCase() | преобразовать строку в нижний регистр |
| String toUpperCase() | преобразовать строку в верхний регистр |
| String trim() | отсечь на концах строки пустые символы |
| static valueOf(Object e) | статические методы преобразования различных типов в строку |

Методы поиска возвращают индекс вхождения или -1 если искомое не найдено. Методы преобразования как replace не изменяют саму строку а возвращают соответствующий новый объект строки.

Если необходимо сделать множество преобразований над строкой, то на это время эффективнее воспользоваться динамической строкой, реализуемой классом StringBuffer.

Некоторые методы объекта StringBuffer.

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| StringBuffer append(Obejct a) | преобразовать a в строку и добавить в конец |
| StringBuffer insert(int offset, Object a) | преобразовать a в строку и вставить ее в указанную позицию |
| StringBuffer delete(int start, int end) | удалить символы с указанной по указанную позицию |
| int indexOf(String str) | поиск первого вхождения подстроки |
| int lastIndexOf(String str) | поиск последнего вхождения подстроки |
| StringBuffer reverse() | расположить символы в обратном порядке |
| StringBuffer setLength(int newLength) | установить новый размер строки |
| String toString() | вернуть строковое представление объекта |