IT-Academy

RegExp
Обертки
Object

Рассматриваемые вопросы

IT-Academy

- RegExp
- Классы оболочки
- Object

Что такое регулярное выражение

Регулярное выражение (**RegExp**) - это своего рода шаблон, который может быть применен к тексту. Java предоставляет пакет java.util.regex для сопоставления с регулярными выражениями.

Регулярное выражение или соответствует тексту (его части) или нет. Если регулярное выражение совпадает с частью текста, то мы можем найти его. Если регулярное выражение составное, то мы можем легко выяснить, какая часть регулярного выражения совпадает с какой частью текста.

RegExp. Пример

IT-Academy

Например, есть следующая строка:

```
Используются файлы file1.doc, file2.txt.
А еще было бы неплохо обратить внимание на файл file3.img.
Также просмотрите содержимое file4.doc.
```

Из строки нужно вырезать все имена файлов: file1.doc, file2.txt, file3.img, file4.doc. Для нахождения используется регулярное выражение:

```
[a-zA-Z]+\\.[a-z]{3}
```

Регулярное выражение [a-z]+ соответствует всем строчным буквам в тексте. [a-z] означает любой символ от a до z включительно, и + означает «один или более» символов.

RegExp. Пример

IT-Academy

Например, есть следующая строка:

```
Используются файлы file1.doc, file2.txt.
А еще было бы неплохо обратить внимание на файл file3.img.
Также просмотрите содержимое file4.doc.
```

Из строки нужно вырезать все имена файлов: file1.doc, file2.txt, file3.img, file4.doc. Для нахождения используется регулярное выражение:

```
[a-zA-Z]+\\.[a-z]{3}
```

Регулярное выражение [a-z]+ соответствует всем строчным буквам в тексте. [a-z] означает любой символ от a до z включительно, и + означает «один или более» символов.

Pattern

IT-Academy

Pattern класс - объект класса составляет представление регулярного выражения. Класс Pattern не предусматривает никаких публичных конструкторов. Чтобы создать шаблон, необходимо сначала вызвать один из публичных статических методов, которые затем возвращают объект класса Pattern. Эти методы принимают регулярное выражение в качестве аргумента.

```
Pattern pattern = Pattern.compile("[a-z]+");
```

Matcher

IT-Academy

Matcher класс - объект «Искатель» является двигателем, который интерпретирует шаблон и выполняет операции сопоставления с входной строкой. Как и Pattern класс, Matcher не имеет публичных конструкторов. Вы получаете объект Matcher вызовом метода matcher(), на объекте класса Pattern.

Методы класса Matcher:

- **matches**() возвращает true, если шаблон соответствует всей строке, иначе false.
- **lookingAt**() возвращает true, если шаблон соответствует началу строки, и false в противном случае.
- **find**() возвращает true, если шаблон совпадает с любой частью текста.

Основные метасимволы

IT-Academy

- **^** (крышка) начало проверяемой строки
- **$** (доллар) конец проверяемой строки
- **.** (точка) представляет собой сокращенную форму записи для символьного класса, совпадающего с любым символом
- **|** означает «или». Подвыражения, объединенные этим способом, называются альтернативами (alternatives)
- **?** (знак вопроса) означает, что предшествующий ему символ является необязательным
- **+** обозначает «один или несколько экземпляров непосредственно предшествующего элемента
- ***** любое количество экземпляров элемента (в том числе и нулевое)

Основные метасимволы

IT-Academy

- **\\d** цифровой символ
- **\\D** не цифровой символ
- **\\s** пробельный символ
- **\\S** не пробельный символ
- **\\w** буквенный(латиница) или цифровой символ или знак подчёркивания
- **\\W** любой символ, кроме буквенного или цифрового символа или знака подчёркивания
- **[abc]** Диапазон символов или цифр

Примеры регулярных выражений

IT-Academy

a? - a один раз или ни разу

a* - a ноль или более раз

a+ - a один или более раз

a{n} - a n раз

a{n,}- a n или более раз

a{n,m}- a от n до m

Жадный режим квантификатора

Особенностью квантификаторов является возможность использования их в разных режимах: **жадном**, **сверхжадном** и **ленивом.**

По умолчанию квантификатор работает в жадном режиме. Это означает, что он ищет максимально длинное совпадение в строке.

"A.+a" // жадный режим

Сверхжадный режим квантификатора

IT-Academy

В сверхжадном режиме работа матчера аналогична механизму жадного режима. После захвата всей строки матчер добавляет остаток шаблона и сравнивает с захваченной строкой. В нашем примере при выполнении метода main с шаблоном "A.++a" совпадений не будет найдено.

"A.++a" // сверхжадный режим

Ленивый режим квантификатора

В результате работы метода main при использовании шаблона "A.+?a" мы получим следующий результат:

Алла

Алекса

"A.+?a" // ленивый режим

Сравнение регулярного выражения с текстом

IT-Academy

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexDemo2 {
    public static void main(String[] args) {
        Pattern pattern = Pattern.compile("a*b");
        Matcher matcher = pattern.matcher("aaab");
        boolean b = matcher.matches();
        System.out.println(b);
    }
}
```

Простой валидатор ссылки

IT-Academy

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexDemo3 {
    public static void main(String[] args) {
        System.out.println(test("google.com"));
        System.out.println(test("reference1.ua"));
        System.out.println(test("reference1.org"));
    }

    public static boolean test(String testString) {
        Pattern pattern = Pattern.compile(".+\\.(com|ua|ru)");
        Matcher matcher = pattern.matcher(testString);
        return matcher.matches();
    }
}
```

Метод Pattern.split()

IT-Academy

Класс Pattern содержит метод **split**(), который разбивает строку на подстроки, используя указанный в шаблоне разделитель:

```
import java.util.Arrays;
import java.util.regex.Pattern;

public class RegexDemo5 {
    public static void main(String[] args) {
        Pattern pattern = Pattern.compile("\\d+\\s?");
        String[] words = pattern.split("java5tiger 77 java6mustang");
        System.out.print(Arrays.toString(words));
    }
}
```

Метод String.split()

IT-Academy

```
import java.util.Arrays;


public class RegexDemo6 {
    public static void main(String[] args) {
        String str = "java5tiger 77 java6mustang";
        String[] words = str.split("\\d+\\s?");
        System.out.print(Arrays.toString(words));
    }
}
```

Простой валидатор ссылки

IT-Academy

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexDemo3 {
    public static void main(String[] args) {
        System.out.println(test("google.com"));
        System.out.println(test("reference1.ua"));
        System.out.println(test("reference1.org"));
    }

    public static boolean test(String testString) {
        Pattern pattern = Pattern.compile(".+\\.(com|ua|ru)");
        Matcher matcher = pattern.matcher(testString);
        return matcher.matches();
    }
}
```

Класс Object

IT-Academy

В Java определен один специальный класс, называемый **Object**. Все остальные классы являются подклассами, производными от этого класса, даже если в объявлении это явно не указано. В классе **Object** определен ряд методов, которые доступны всем классам языка Java.

Методы класс Object

IT-Academy

- **protected Object clone()** - создает новый объект, не отличающийся от клонируемого.
- **public boolean equals(Object obj)** - определяет, равен ли один объект другому.
- **protected void finalize()** - вызывается перед удалением неиспользуемого объекта.
- **public final Class<?> getClass()** - получает класс объекта во время выполнения.
- **public int hashCode()** - возвращает хеш-код, связанный с вызывающим объектом.
- **public final void notify()** - возобновляет исполнение потока, ожидающего вызывающего объекта.

Методы класс Object

IT-Academy

- **public final void notify()** - возобновляет исполнение потока, ожидающего вызывающего объекта.
- **public final void notifyAll()** - возобновляет исполнение всех потоков, ожидающих вызывающего объекта.
- **public String toString()** - возвращает символьную строку, описывающую объект.
- **public final void wait()** - ожидает другого потока исполнения.
- **public final void wait(long timeout)** - ожидает другого потока исполнения.
- **public final void wait(long timeout, int nanos)** - ожидает другого потока исполнения.

Методы класс Object

IT-Academy

- **public final void notify()** - возобновляет исполнение потока, ожидающего вызывающего объекта.
- **public final void notifyAll()** - возобновляет исполнение всех потоков, ожидающих вызывающего объекта.
- **public String toString()** - возвращает символьную строку, описывающую объект.
- **public final void wait()** - ожидает другого потока исполнения.
- **public final void wait(long timeout)** - ожидает другого потока исполнения.
- **public final void wait(long timeout, int nanos)** - ожидает другого потока исполнения.

Метод equals()

IT-Academy

В Java сравнение объектов производится с помощью метода **equals**() класса **Object**. Этот метод сравнивает содержимое объектов и выводит значение типа boolean. Значение true - если содержимое эквивалентно, и false — если нет.

Операция == не рекомендуется для сравнения объектов в Java. Дело в том, что при сравнение объектов, операция == вернет true лишь в одном случае — когда ссылки указывают на один и тот же объект. В данном случае не учитывается содержимое переменных класса.

При создании пользовательского класса, принято переопределять метод **equals**() таким образом, что бы учитывались переменные объекта.

Метод toString()

IT-Academy

Часто необходимо вывести на экран содержимое объекта. Для этого в классе **Object** определен метод **toString**(), возвращающий символьную строку описывающую объект. При создании нового класса принято переопределение **toString**() таким образом, чтобы возвращающая строка содержала в себе имя класса, имена и значения всех переменных.

Для вызова метода toString() необходимо просто передать нужный объект в System.out.println:

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

Чисто теоретически можно явно вызывать метод toString() - System.out.println(person.toString()), но так не принято.

Если у класса Person не переопределен метод toString(), то при запуске класса PersonDemo4 вызовется метод toString(), определенный в классе Object. И на консоль выведется нечто такое:

oop.Person@5e2a3193

Метод hashCode()

IT-Academy

Что такое хеш-код?

Если очень просто, то хеш-код — это число. Если более точно, то это битовая строка фиксированной длины, полученная из массива произвольной длины.

Представление объекта в виде числа.

В java hash code представлен в виде числа примитивного типа int, который равен 4-м байтам, и может помещать числа от -2 147 483 648 до 2 147 483 647. На данном этапе важно понимать, что хеш-код это число, у которого есть свой предел, который для java ограничен примитивным целочисленным типом int.

Метод toString()

IT-Academy

Часто необходимо вывести на экран содержимое объекта. Для этого в классе **Object** определен метод **toString**(), возвращающий символьную строку описывающую объект. При создании нового класса принято переопределение **toString**() таким образом, чтобы возвращающая строка содержала в себе имя класса, имена и значения всех переменных.

Для вызова метода toString() необходимо просто передать нужный объект в System.out.println:

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

Чисто теоретически можно явно вызывать метод toString() - System.out.println(person.toString()), но так не принято.

Если у класса Person не переопределен метод toString(), то при запуске класса PersonDemo4 вызовется метод toString(), определенный в классе Object. И на консоль выведется нечто такое:

oop.Person@5e2a3193

Модификатор native

IT-Academy

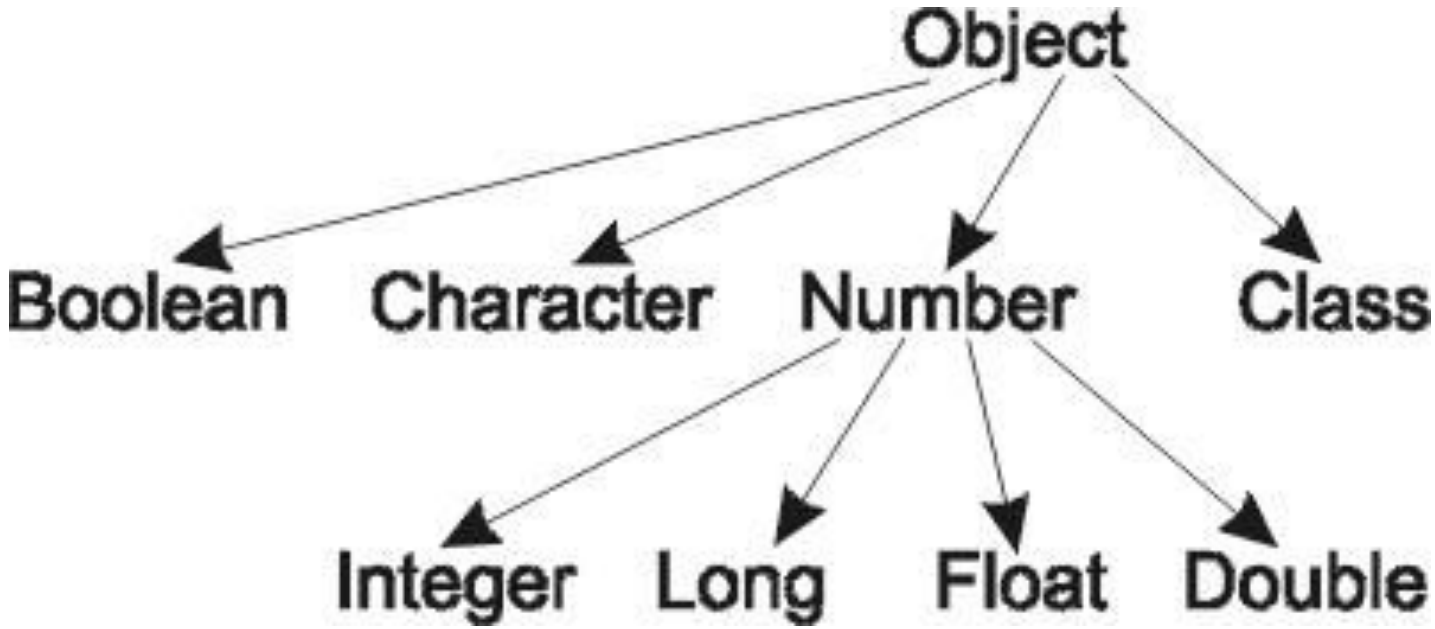
Приложение на языке Java может вызывать методы, написанные на языке C++.

Такие методы объявляются в языке Java с ключевым словом **native**, которое сообщает компилятору, что метод реализован в другом месте.

```
public native int hashCode();
```

Классы оболочки

Каждому простому типу в Java соответствует класс-оболочка.



Классы оболочки

IT-Academy

Классы-оболочки выполняют две основные функции.

Первая состоит в поддержке контейнера методов и переменных, относящихся к определенному типу (скажем, методов преобразования строк и констант, задающих границы интервала допустимых значений).

Вторая функция классов-оболочек заключается в обеспечении возможности создания объектов для хранения значений простых типов и использования их в контексте классов, которые "умеют" обращаться только со ссылками на объекты типа Object.

```
Integer intObject = new Integer(3);
```

Конструкторы оболочек

В таблицы для каждого класса указан примитивный тип и варианты конструкторов. Один на вход принимает значение соответствующего примитивного значения, а второй - значение типа String. Исключения: класс Character, у которого только один конструктор с аргументом char и класс Float, объявляющий три конструктора - для значения float, String и еще double.

Примитивный тип	***Оболочка***	***Аргументы конструктора***
boolean	Boolean	boolean or String
byte	Byte	byte or String
char	Character	char
double	Double	double or String
float	Float	float, double, or String
int	Integer	int or String
long	Long	long or String
short	Short	short or String

Методы классов оболочек

IT-Academy

- **valueOf**() - второй способ создания объектов оболочек
- **parseXxx**() - метод позволяющие преобразовывать строку в соответствующее примитивное значение
- **toString**() – строковой предствление объекта
- **toHexString(), toOctalString(), toBinaryString() -** Integer и Long позволяют преобразовывать числа из десятичной системы исчисления к шестнадцатеричной, восьмеричной и двоичной

Статические константы классов оболочек

Каждый класс оболочка содержит статические константы, содержащие максимальное и минимальное значения для данного типа.
Например, в классе **Integer** есть константы **Integer.MIN_VALUE** – минимальное **int** значение и **Integer.MAX_VALUE** – максимальное int значение.
Классы-обертки числовых типов **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 (расшифровывается как Not a Number). При делении на ноль возникает ошибка - на ноль делить нельзя. Чтобы этого не происходило, и ввели переменные NEGATIVE_INFINITY и POSITIVE_INFINITY. Результат умножения бесконечности на ноль - это значение NaN.

Автоупаковка

IT-Academy

Автоупаковка (Autoboxing) - это механизм неявной инициализации объектов классов-оберток (Byte, Short, Character, Integer, Long, Float, Double) значениями соответствующих им исходных примитивных типов (соотв. byte, short, char, int, long, float, double), без явного использования конструктора класса.

Автоупаковка происходит при прямом присвоении примитива - классу-обертке (с помощью оператора "="), либо при передаче примитива в параметры метода (типа "класса-обертки").

Автораспаковка

Автораспаковка (Unboxing) - это механизм конвертации объекта класса оболочки в соответствующий ему примитивый типов (соотв. byte, short, char, int, long, float, double).

Primitive type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

Вопрос

IT-Academy

IT-Academy

Спасибо за внимание