Números e cadeas

Nesta sección verase como usar as clases `Number` e `String` que proporciona a API de Java. Ademais, explícase como formatar a saída de datos dun programa.

Sumario

Números

Java ten unha clase especial que facilita o traballo con números (por exemplo o cambio de base de decimal a octal; a conversión entre tipos, por exemplo, de String a int, etc.), é a clase Number que está no paquete java.lang. Esta clase e as súas subclases proporcionan ferramentas para traballar con tipos numéricos. Non hai que confundila cos tipos de datos primitivos que proporciona Java e que o único que permiten é declarar unha variable dun tipo de dato determinado (int, float, etc.)

Veremos ademais a clase PrintStream que proporcionan métodos para imprimir números cun determinado formato.

Por último, traballaremos coa clase Math no paquete java.lang que contén funcións matemáticas que serven de complemento dos operadores xa existentes na linguaxe. Esta clase ten métodos para as funcións trigonométricas, exponenciais, etc.

A clase Number

Cando traballamos con números normalmente usamos os tipos de datos primitivos. Por exemplo:

```
int i = 500;
float gpa = 3.65;
byte mascara = 0xff;
```

Non entanto, existen motivos para usar obxectos en lugar de tipos de datos primitivos, como se verá máis adiante. Java proporciona clases que **"envolven"** (wrap) cada un dos tipos de datos primitivos, permitindo traballar indistintamente cos obxectos de tipo Number ou cos tipos primitivos. Isto é transparente para o programador xa que se usamos un tipo primitivo en lugar dun obxecto da clase Number o compilador empaqueta automaticamente o tipo primitivo na clase que o envolve. Analogamente, se usamos un obxecto de tipo Number onde se espera un tipo primitivo o

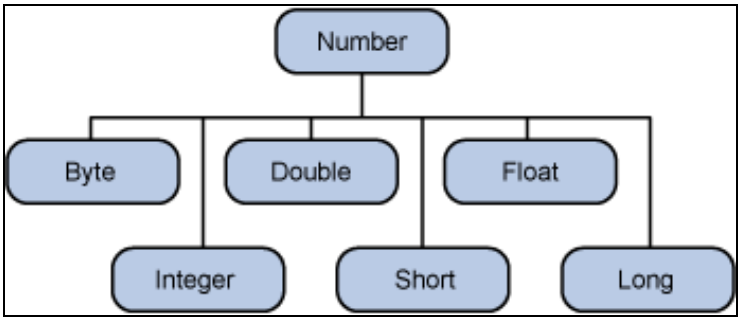
compilador desempaqueta o obxecto automaticamente. Vexamos un exemplo:

```
Integer x, y; // Integer é unha subclase da clase Number para traballar con enteiros
x = 12;
y = 15;
System.out.println(x+y);
```

Cando se asignan valores a x e y o compilador empaqueta os enteiros 12 e 15 porque x e y son obxectos da clase Integer. Na sentenza println() desempaquétanse x e y para que podan sumarse como enteiros, xa que o operador + usa operandos de tipo int (tipo primitivo) e non obxectos de tipo Integer. Con todo, hai que ter en conta que un obxecto de tipo Boolean non se pode usar como un tipo primitivo boolean.

Para cada tipo de dato primitivo hai unha clase que o envolve. Por exemplo, a clase que envolve a int e Integer; a clase que envolve a float é Float, etc. A clase que envolve o tipo de dato primitivo sempre se chame igual que o tipo primitivo pero con a primeira letra en maiúsculas, exceptuando Integer para int e Character para char que se verá máis adiante.

Todas as clases numéricas son subclases da clase Number, que é abstracta. A **xerarquía** é a seguinte:



Hai outras catro subclases de Number que non veremos. Estas son BigDecimal e BigInteger, usadas para cálculos de alta precisión; e AtomicInteger e AtomicLong usadas para aplicacións muti-fío (multi-threaded).

Os **motivos** para usar obxectos de tipo Number en lugar de tipos primitivos son, principalmente, tres:

1. A clase Number proporciona métodos para converter valores entre tipos primitivos, a cadeas ou conversións entre diferentes sistemas numéricos (decimal, octal, hexadecimal e binario).
2. Permitir traballar cos tipos de datos primitivos coma se fosen obxectos, por exemplo, cando se manipulan conxuntos de números pode ser máis axeitado usar métodos con argumentos de tipo Number.
3. A clase Number proporciona valores constantes, como MIN_VALUE e MAX_VALUE, que devolven os valores máis baixos e máis altos que se poden usar neste tipo de datos.

Construtores

Os construtores das clases que envolven os tipos primitivos poden ter como argumento:

- Un tipo primitivo
- Un String que representa ao número

Por exemplo:

```
Integer i1 = new Integer(42);
Integer i2 = new Integer("42");
```

Ou tamén:

```
Float f1 = new Float(3.14f);
Float f2 = new Float("3.14f");
```

Isto é así para todos os construtores das subclases de Number exceptuando a clase Character que só pode ter como argumento un char:

```
Character c1 = new Character('c');
```

Métodos útiles

Nestas clases existen tres tipos de métodos que son especialmente útiles:

- Os métodos **xxxValue()**. Non teñen argumentos e devolven un tipo primitivo a partir dun obxecto de tipo Number. Por exemplo:

```
 Integer i2 = new Integer(42); //  crea un obxecto de tipo Integer (wrapper object)
 byte b = i2.byteValue();      //  convirte o valor de i2 ao tipo primitivo byte
 short s = i2.shortValue();    //  convirte o valor de i2 ao tipo primitivo short
 double d = i2.doubleValue();  //  convirte o valor de i2 ao tipo primitivo double
//Outro exemplo:
 Float f2 = new Float(3.14f);
 short s = f2.shortValue();
 System.out.println(s);        // o resultado é 3  (truncado, non redondeado)
```

- Os métodos **parseXxx()**. Teñen un String como argumento e devolven un tipo primitivo (xa vistos para "parsear" argmentos en liña de comandos). Permiten facer conversión entre números en base 10 a ouras bases. Por exemplo:

```
double d4 = Double.parseDouble("3.14"); // convirte un String a un double
System.out.println("d4 = " + d4);       // o resultado é "d4 = 3.14"
//Outro exemplo
long L2 = Long.parseLong("101010", 2); // String en binario (número 42) a long
System.out.println("L2 = " + L2);      // o resultado é "L2 = 42"
```

- Os métodos **valueOf()**. Teñen un String como argumento e devolven un obxecto. Por exemplo:

```
Double d5 = Double.valueOf("3.14");
System.out.println(d5 instanceof Double ); // O resultado é true
```

As clases PrintStream e DecimalFormat

Xa coñecemos os métodos `print` e `println` que serven para imprimir cadeas pola saída estándar (pantalla) mediante `System.out`. Os números poden converterse a cadeas, polo tanto, podemos usar `print` e `println` tamén para imprimir números. Con todo, Java proporciona métodos adicionais que se atopan na clase PrintStream que permiten a impresión de números con formato dando ao programador maior control sobre o formato de saída cando se traballa con números.

Por outra parte, a clase `DecimalFormat` permite formatar números decimais. Non a veremos pero poden consultarse os seus detalles na ligazón anterior.

Os métodos printf e format

A clase `PrintStream` do paquete `java.io` inclúe os métodos `printf` e `format` que se poden usar no canto de `print` e `println`. En Java, System.out é unha instancia da clase PrintStream, polo que se pode usar `format` ou `printf` en calquera sitio onde anteriormente se usara `print` ou `println`.

A sintaxe é a mesma para ambos os dous métodos, `printf` e `format`:

```
public PrintStream format(String formato, Object... args)
```

Onde formato é unha cadea e args é unha lista de variables a imprimir. Por exemplo:

```
System.out.format("Variable float: %f, variable enteira: %d, cadea: %s", varFloat, varInt, varString);
```

O primeiro parámetro, chamado formato, é o que se coñece como unha **cadea de formato** que contén texto plano e **especificadores de formato**. Os especificadores de formato son caracteres especiais que formatan as variables declaradas en Object... args. (Á notación Object... args chámaselle **varargs**, que significa que o números de argumentos é variable.)

Os especificadores de formato comezan co carácter de porcentaxe (%). O **conversor** é un carácter que indica o tipo do argumento a ser formatado. Entre o carácter de porcentaxe (%) e o conversor pode haber opcións e especificadores adicionais. Existen moitos conversores e están documentados en `java.util.Formatter`

Por exemplo:

```
int i = 461012;
System.out.format("O valor de i e: %d%n", i);
```

O especificador de formato %d indica que a variable que se vai imprimir é un enteiro. O especificador %n é un carácter de nova liña independente da plataforma. A saída é:

```
O valor de i e: 461012
```

Os métodos `printf` e `format` están sobrecargados. Cada un deles ten unha versión coa seguinte sintaxe:

```
public PrintStream format(Locale l, String formato, Object... args)
```

Para imprimir números no sistema español ou francés (onde se usa unha coma en lugar dun punto para representar a parte decimal dos números en coma flotante) úsase, por exemplo, o seguinte:

```
import java.util.*;
public class Formato2 {
 public static void main (String arg[]) {
   float i = 3.1416F;
   System.out.format(Locale.FRANCE, "O valor de i e:%f%n", i);
 }
}
```

A clase Math

Xa vimos os operadores aritméticos esenciais que se poden usar en Java: +, -, *, /, e %. A clase `Math` que se atopa no paquete `java.lang` porporciona métodos e constantes para realizar operacións matemáticas máis complexas. Estes métodos son static, polo que poden chamarse directamente co nome da clase. Por exemplo:

```
Math.cos(angulo);
```

Asemade, a clase Math é una clase final, polo que non se pode derivar dela.

Constantes

Temos dúas constantes na clase Math:

- Math.E, que é o número E, base dos logaritmos naturais
- Math.PI, que é o número PI

Métodos básicos

A clase Math inclúe máis de 40 métodos. Os máis importantes explícanse a continuación.

abs()

Este método devolve o valor abosluto do argumento. Por exemplo:

```
x = Math.abs(99);           // saída  99
x = Math.abs(-99)           // saída 99
```

O método está sobrecargado para funcionar con argumentos de tipo int, long, float, ou double. A súa **sinatura** é a seguinte:

```
public  static    int abs(int a)
public  static    long abs(long a)
public  static    float abs(float a)
public  static    double abs(double a)
```

ceil()

Este método devolve un double que é o valor enteiro maior ou igual ao valor do argumento . As seguintes chamadas a Math.ceil() devolven o valor de tipo double 9.0:

```
Math.ceil(9.0)          // devolve 9.0
Math.ceil(8.8)          // devolve 9.0
Math.ceil(8.02)         // devolve 9.0!
```

Para os números negativos é igual. Só hai que recordar que ?9 é maior que ?10. As seguintes chamadas ao método Math.ceil() devolven o valor double -9.0:

```
Math.ceil(-9.0)         // devolve ?9.0
Math.ceil(-9.4)         // devolve ?9.0
Math.ceil(-9.8)         // devolve ?9.0
```

O método ceil() non está sobrecargado e a súa **sinatura** é a seguinte:

```
public static double ceil(double a)
```

floor()

É a antítese do método anterior. Polo tanto devolve un double que é o enteiro menor ou igual ao valor do argumento. As seguintes chamadas a Math.floor() devolven o valor double 9.0:

```
Math.floor(9.0)            // devolve 9.0
Math.floor(9.4)            // devolve 9.0
Math.floor(9.8)            // devolve 9.0
```

Igual que antes, hai que ter en conta que -9 é menor que -8:

```
Math.floor(-9.0)           // devolve ?9.0
Math.floor(-8.8)           // devolve ?9.0
Math.floor(-8.1)           // devolve ?9.0
```

A **sinatura** de floor() é a seguinte:

```
public static double floor(double a)
```

max()

O método max() devolve o maior dos seus dous argumentos. Por exemplo:

```
x = Math.max(1024, -5000);            //    devolve 1024.
```

Este método está sobrecargado para traballar con argumentos de tipo int, long, float, ou double. Se os dous parámetros son iguais, max() o valor igual aos dous argumentos.

A **sinatura** de max() é a seguinte:

```
public  static    int max(int a, int b)
public  static    long max(long a, long b)
public  static    float max(float a, float b)
public  static    double max(double a, double b)
```

min()

O método min() é a antítese de max(). Devolve, polo tanto, o valor menor dos dous argumentos. Por exemplo:

```
 x = Math.min(0.5, 0.0);           //   devolve        0.0
```

Este método está sobrecargado para poder traballar con argumentos de tipo int, long, float, ou double. A súa **sinatura** é a seguinte:

```
public  static    int min(int a, int b)
public  static    long min(long a, long b)
public  static    float min(float a, float b)
public  static    double min(double a, double b)
```

round()

O método round() devolve o enteiro máis cercano ao argumento. O algoritmo que utiliza round() é sumar 0.5 ao argumento e truncar ao enteiro equivalente. Este método está sobrecargado para traballar con argumentos float ou double. Por exemplo:

```
Math.round(-10.5);         // devolve ?10
```

A **sinatura** de round() é a seguinte:

```
public static int round(float a)
public static long round(double a)
```

Métodos logarítmicos e exponenciais

sqrt()

Devolve a raíz cadrada dun double. Por exemplo:

```
 Math.sqrt(9.0) // devolve 3.0
```

A **sinatura** do método é a seguinte:

```
  public static double sqrt(double a)
```

pow()

Devolve o valor do primeiro argumento elevado ao segundo argumento. A **sinatura** do método é a seguinte:

```
double pow(double base, double exponente)
```

log()

Devolve o logaritmo natural do argumento. A súa **sinatura** é a seguinte:

```
double log(double d)
```

Métodos trigonométricos

sin()

Devolve o seno dun ángulo. O argumento é un double representando un ángulo en radiáns. Os grados poden converterse a radiáns usando **Math.toRadians()**. Analogamente existe o método **Math.toDegrees()** Por exemplo:

```
  Math.sin(Math.toRadians(90.0)) //           devolve 1.0
```

A **sinatura** de sin() é a seguinte: public static double sin(double a)

cos()

Devolve o coseno dun ángulo. O argumento é un double representando un ángulo en radiáns. Por exemplo:

```
  Math.cos(Math.toRadians(0.0)) // devolve 1.0
```

A **sinatura** de cos() é a seguinte:

```
  public static double cos(double a)
```

tan()

Devolve a tanxente dun ángulo. O argumento é un double representando un ángulo en radiáns. Por exemplo:

```
  Math.tan(Math.toRadians(45.0))          // devolve 1.0
```

A **sinatura** de tan() é a seguinte:

```
public static double tan(double a)
```

Números aleatorios

random()

O método random() devolve un double aleatorio maior ou igual a 0.0 e menor que 1.0. Este método non ten parámetros:

```
public class RandomTest {
 public static void main(String [] args) {
   for (int x=0; x < 15; x++)
     System.out.print( (int)(Math.random()*10) + " " );
 }
}
```

A **sinatura** do método random() é a seguinte:

```
public static double random( )
```

Caracteres

Cando queremos usar un carácter nun programa normalmente empregamos o tipo primitivo char. Por exemplo:

```
char ch = 'a';
char uniChar = '\u039A'; // Código Unicode para el carácter griego omega
char[] charArray ={ 'a', 'b', 'c', 'd', 'e' }; // un array de caracteres
```

Java proporciona unha clase que envolve (wrap) o tipo char nun obxecto de tipo Character, tal e como acontecía coa clase Number. Un obxecto de tipo Character ten un único atributo de tipo char. A clase Character proporciona unha serie de métodos útiles para manipular caracteres.

Para crear un obxecto de tipo Character podemos instancialo usando o constructor, como se fose outro obxecto calquera:

```
Character ch = new Character('a');
```

Se usamos unha variable de tipo char en lugar dun obxecto da clase Character, e viceversa, o compilador, nalgunhas situacións fará a "conversión" por nós, automaticamente. Ao primeiro chámaselle **autoboxing** e ao segundo **unboxing** (que se podería traducir por autoempaquetado e desempaquetado). Vexámolo no seguinte exemplo:

```
Character ch = 'a'; // o tipo primitivo 'a' autoempaquétase no obxecto ch de tipo Character
```

A seguinte táboa lista algúns dos métodos útiles desta clase. Non é exhaustiva. Podes consultar a lista completa (máis de 50) en java.lang.Character.

Método	Descrición
`boolean isLetter(char ch)` `boolean isDigit(char ch)`	Determina se o parámetro é unha letra ou un díxito.
`boolean isWhiteSpace(char ch)`	Determina se o parámetro e un espazo en branco.
`boolean isUpperCase(char ch)` `boolean isLowerCase(char ch)`	Determina se o carácter é maiúscula ou minúscula.
`char toUpperCase(char ch)` `char toLowerCase(char ch)`	Devolve o carácer en maiúsuclas ou minúculas respectivamente.
`toString(char ch)`	Devolve un obxecto de tipo String a partir do carácter, é dicir, unha cadea formada por un só carácter.

Secuencias de escape

Un carácter precedido por un *backslash* (\) é unha secuencia de escape é ten un significado especial para o compilador. Por exemplo, o carácter (\n) indica unha nova liña cando imprimimos con `System.out.println()`. A seguinte táboa mostra as secuencias de escape en Java:

Secuencia de escape	Descrición
`\t`	Tabulación do texto.
`\b`	Backspace (tecla de retroceso)
`\n`	Nova liña.
`\r`	Retorno de carro (enter).
`\f`	Nova liña e tabulación.
`\"`	Imprime as comiñas
`\\"`	Imprime o carácter \

Por exemplo, se queremos imprimir a cadea:

```
Díxome "hola!".
```

Habería que escribir:

```
System.out.println("Díxome \"hola!\".");
```

Cadeas

As cadeas de caracteres están representadas pola clase String. Unha cadea de caracteres en Java é un obxecto.

Creación de cadeas

O xeito máis rápido de crear unha cadea en Java é o seguinte:

```
String saudo = "Ola!";
```

Como unha cadea é un obxecto tamén podemos usar o palabra reservada new e un construtor para crear unha cadea. Hai 11 construtores que permiten crear cadeas. Por exemplo:

```
char[] arrayOla = { 'o', 'l', 'a', '.'};
String cadeaOla = new String(arrayOla);
System.out.println(arrayOla);
```

As cadeas son **inmutables**, polo que unha vez creadas non se poden modificar. O que os métodos da clase String fan sobre un obxecto de tipo cadea é crear e devolver unha nova cadea que contén o resultado da operación.

Lonxitude dunha cadea

Os métodos que se usan para acceder aos atributos dunha clase chámanse métodos accesores. Un método accesor da clase String para devolver o tamaño dunha cadea é length() (ollo, non confundir co atributo length dos arrays).

```
String palindromo = "Radar";
int len = palindromo.length();
```

Un palíndromo é unha palabra simétrica, é dicir, que se deletrea igual cara adiante ou cara atrás, ignorando as maiúsculas e a puntuación. O seguinte programa dálle a volta a un palíndromo. Invoca ao método charAt(i), que devolve o carácter na posición i da cadea, empezando a contar en 0.

```
public class StringDemo {
    public static void main(String[] args) {
        String palindromo = "Radar";
        int len = palindromo.length();
        char[] tempCharArray = new char[len];
        char[] charArray = new char[len];

        // Mete a cadea nun array de caracteres
        for (int i = 0; i < len; i++) {
            tempCharArray[i] = palindromo.charAt(i);
```

```
            }

            // Invirte o array
            for (int j = 0; j < len; j++) {
                charArray[j] = tempCharArray[len - 1 - j];
            }

            String reversePalindromo =  new String(charArray);
            System.out.println(reversePalindromo);
        }
    }
```

A saída do programa será:

```
radaR
```

Concatenación de cadeas

Podemos usar o método concat para concatenar dúas cadeas:

```
string1.concat(string2);
```

Isto devolve **unha nova cadea** formada pola concatenación de string1 e string2. O método concat tamén se pode usar do seguinte xeito:

```
"Meu nome é ".concat("Rumplestiltskin");
```

Tamén se pode usar o operador +:

```
"Ola," + " meu" + "!"
```

A saída será:

```
"Ola, meu!"
```

Pódense concatenar obxectos de distinto tipo. Para cada un deles chamarase implícitamente ao método toString() para convertilo nunha cadea.

En Java, un ficheiro de código fonte non pode ter sentenzas de máis dunha liña, polo que se usa o operador +, como vemos no seguinte exemplo:

```
        String quote = "Isto é unha cade de exemplo " +
                "para o curso de Java.";
```

Conversión entre números e cadeas

De cadeas a números

As subclases da clase Number teñen un método **valueOf** que converte unha cadea a un número. Por exemplo:

```
public class ValueOfDemo {
    public static void main(String[] args) {

        // require dous argumentos por liña de comandos
        if (args.length == 2) {
                    //converte cadeas a números
            float a = (Float.valueOf(args[0]) ).floatValue();
            float b = (Float.valueOf(args[1]) ).floatValue();

            //fai algunhas operacións
            System.out.println("a + b = " + (a + b) );
            System.out.println("a - b = " + (a - b) );
            System.out.println("a * b = " + (a * b) );
            System.out.println("a / b = " + (a / b) );
            System.out.println("a % b = " + (a % b) );
        } else {
           System.out.println("Este programa require dous argumentos por liña de comandos.");
        }
    }
}
```

A saída do programa anterior se lle pasamos os valores 4.5 and 87.2 será a seguinte:

```
a + b = 91.7
a - b = -82.7
a * b = 392.4
a / b = 0.0516055
a % b = 4.5
```

O método valueOf é similar ao método parseXxx. Para chamar por este método escribiríamos o seguinte:

```
float a = Float.parseFloat(args[0]);
float b = Float.parseFloat(args[1]);
```

De números a cadeas

A conversión de números a cadeas faise co método **toString()** que existe en todas as subclases de Number:

```
int i;
double d;
String s3 = Integer.toString(i);
String s4 = Double.toString(d);
```

O seguinte exemplo usa o método toString para converter un número a unha cadea. O programa usa algúns métodos para calcular o número de díxitos antes e despois do punto decimal:

```
public class ToStringDemo {

    public static void main(String[] args) {
        double d = 858.48;
        String s = Double.toString(d);

        int punto = s.indexOf('.');

        System.out.println(punto + " díxitos antes do punto decimal.");
        System.out.println( (s.length() - punto - 1) +
            " díxitos despois do punto decimal.");
    }
}
```

A saída do programa é:

```
3 díxitos antes do punto decimal.
2 díxitos despois do punto decimal.
```

Outro xeito de converter números en cadeas e usando o operador +, que está sobrecargado. Neste caso é o compilador o encargado de facer a conversión por nós:

```
int i;
String s1 = "" + i; //Concatena "i" cunha cadea baleira
                    // a conversión é automática
```

Manipulación de caracteres nun String

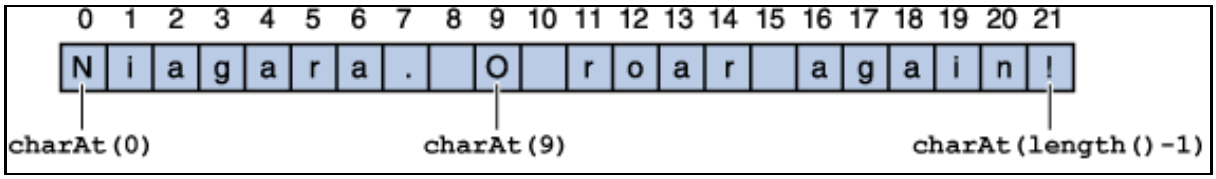
A clase String facilítanos o traballo cos caracteres da cadea a través de diversos métodos.

Obtención de caracteres e substrings mediante o índice

O métodos **charAt()** devolve o carácter da cadea na posición correspondente. Hai que ter en conta que **o primeiro carácter empeza no índice 0**, mentres que o último é length()-1. Por exemplo, o seguinte código devolve o carácter no índice 9:

```
String outroPalindromo = "A torre da derrota";
char unCaracter= outroPalindromo.charAt(9);
```

Na seguinte figura o carácter no índice 9 sería O:



Pódese usar o método **substring** se o que se quere é obter máis dun carácter consecutivo dunha cadea. Este método ten dúas versións, tal e como se amosa na seguinte táboa:

Método	Descrición
`String substring(int inicio, int fin)`	Devolve unha nova cadea que é unha subcadea da orixinal. O primeiro enteiro especifica o índice do primeiro carácter. O segundo argumento é o índice do último carácter + 1.
`String substring(int inicio)`	Devolve unha nova cadea que é unha subcadea da orixinal. O argumento especifica o índice do primeiro carácter. A subcadea que se devolve vai dende ese índice até o final.

O seguinte código obtén a subcadea que vai dende o carácter 2 até o 7, sen incluír, é dicir, a palabra "torre":

```
String s1 = "A torre da derrota";
String s2 = s1.substring(2, 7);
```

Curiosamente, o método substring non segue a convención de Java para nomear aos métodos, xa que debería chamarse subString!

Outros métodos de manipulación de cadeas

Amósanse a continuación outros métodos importantes da clase String:

Método	Descrición
`String trim()`	Devolve unha cadea sen espazos ao comezo e ao final
`String toLowerCase()` `String toUpperCase()`	Devolven unha cadea toda en minúsculas ou en maiúsculas respectivamente.

Busca de caracteres e subcadeas

A seguinte táboa ten métodos para realizar a busca de caracteres e sucadeas nun String:

Método	Descrición
`int indexOf(int ch)` `int lastIndexOf(int ch)`	Devolve o índice da primeira (última) ocurrencia do carácter especificado.
`int indexOf(String str)` `int lastIndexOf(String str)`	Devolve o índice da primeira (última) ocurrencia do substring especificado.
`boolean contains(CharSequence s)`	Devolve true se a cadea contén a secuencia de caracteres especificada.

Substitución de caracteres e subcadeas

A clase String ten varios métodos para a substituión de caracteres e cadeas. O máis usual é o método replace

Método	Descrición
`String replace(char charAntigo, char charNovo)`	Devolve unha nova cadea resultado de substituír todas as ocurrencias de `charAntigo` por `charNovo`.
`String replaceAll(String cadeaAntiga, String cadeaNova)`	Devolve unha nova cadea resultado de substituír todas as ocurrencias de da expresión regular `cadeaAntiga` pola cadea `cadeaNova`.

Comparación de cadeas

A seguinte táboa amosa algúns dos métodos máis importantes da clase String para a comparación de cadeas e subcadeas (para unha lista exhaustiva, consulta a API de Java):

Método	Descrición

`boolean endsWith(String sufixo)`
 	
`boolean startsWith(String prefixo)`
	Devolve true se a cadea finaliza con ou empeza co substring especificado como argumento.

`int compareTo(String outraCadea)`	Compara dúas cadeas lexigograficamente. Devolve un enteiro inidacando se o string que chama polo método é maior que (resultado é > 0), igual (**resultado é = 0**), ou menor que (resultado é < 0) o argumento.
`int compareToIgnoreCase(String str)`	Igual que o anterior pero ignora a diferenza entre maiúsculas e minúsculas.
`boolean regionMatches(int inicioRexion, String str, int inicioRexionStr, int lonxitude)`	Comproba se unha parte especificada dun string é igual que outra parte especificada para o string do argumento. A parte especificada (rexión) ten unha `lonxitude` e para a cadea que chama polo método comeza no índice `inicioRexion` e no índice iniciosRexionStr para o string que se pasa como argumento (str).
`boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)`	Igual que o método anterior pero sen distinguir entre maiúsculas e minúsculas.

O seguinte programa amosa o uso do método regionMatches:

```
public class RegionMatchesDemo {
public static void main(String[] args) {
String cadea = "Queixo con marmelo";
String cadeaBuscada = "marmelo";
int lonCadea = cadea.length();
int lonCadeaBuscada = cadeaBuscada.length();
boolean atopado = false;
for (int i = 0; i <= (lonCadea - lonCadeaBuscada); i++) {
   if (cadea.regionMatches(i, cadeaBuscada, 0, lonCadeaBuscada)) {
      atopado = true;
      System.out.println(cadea.substring(i, i + lonCadeaBuscada));
      break;
   }
}
if (!atopado) System.out.println("Non se atopou.");
}
}
```

A saída do programa é:

```
marmelo
```

A clase StringBuilder

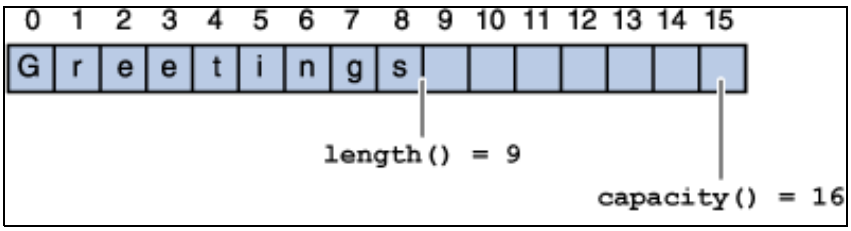
Os obxectos de tipo `StringBuilder` son como os obxectos de tipo `String` pero poden modificarse (recorda que os obxectos de tipo String son inmutables). Con `StringBuilder` en calquera momento a lonxitude e contido da cadea pode cambiarse a través da chamada aos métodos correspondentes.

Sun recomenda usar sempre a clase String a menos que o uso de StringBuilder supoña unha vantaxe en termos de simplificación de código ou mellor rendemento.

Ademais do método length() que tamén ten a clase String, StringBuilder ten un método capacity() que devolve o número de caracteres que foron asignados a unha cadea. Esta capacidade normalmente se asigna nos construtores. Hai catro construtres para StringBuilder que podes consultar na API e que permiten crear cadeas con distintas capacidades. Un exemplo de uso desta clase é o seguinte:

```
StringBuilder sb = new StringBuilder(); // crea unha cadea de capacidade 16
sb.append("Greetings");  // engade 9 character á cadea inicialmente baleira
```

Que creará unha cadea de lonxitude 9 e capacidade 16, tal e como se ve na seguinte figura:



Os principais métodos da clase StringBuilder que non están dispoñibles para a clase String son append() e insert(), que están sobrecargados para aceptar datos de calquera tipo. Os argumentos son, polo tanto, convertidos a cadeas e engadidos our insertados na cadea correspondente. O método append sempre engade ao final, mentres que insert pode engadir nun punto específico da cadea (consulta a API para ver os detalles destes métodos).