ADSI

Cadenas y Números Java - POO

Instructor: Gustavo Adolfo Rodríguez Q.

garodriguez335@misena.edu.co

ADSI

CADENA Y NUMEROS CON JAVA

En esta lección vamos a aprender sobre la clase *Number* y sus subclases. Particularmente aprender sobre los casos en los cuales utilizaremos instancias de la clase *Number* y no tipos numéricos primitivos. Adicionalmente se describen algunas otras clases que son necesarias cuando se realiza trabajos con números. Estas clases nos pueden permitir dar formato a un número o realizar funciones matemáticas avanzadas que complementan a las básicas incluidas en el lenguaje.

Adicionalmente, aprenderemos sobre la utilización de la clase *String,* la cual es una clase que representa cadenas de caracteres en Java y se utiliza ampliamente en cualquier tipo de programa.

1. NUMEROS

Cuando trabajamos con números la mayor parte del tiempo lo hacemos con datos numéricos primitivos como se muestra a continuación:

```
int entero = 2;
int a =6;
double doble = 12.5;
```

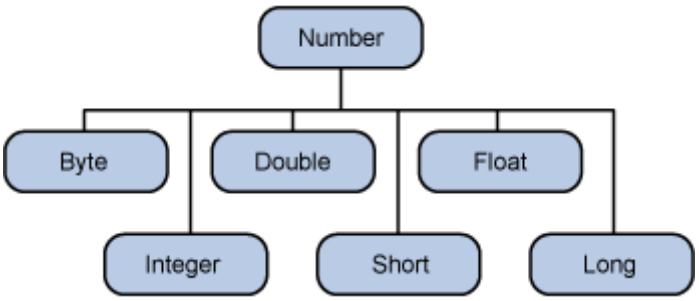
Sin embargo, existen casos donde es necesario utilizar las clases *envolventes (wrappers* en Inglés*)* para cada uno de los tipos numéricos primitivos. Las clases wrappers envuelven un dato primitivo y lo convierten en un objeto.

Algunas operaciones pueden realizarse directamente sobre los objetos envolventes por ejemplo, es permitido realizar las operaciones que se muestran a continuación:

```
Integer x, y;
x = 12;
y = 15;
System.out.println(x+y);
```

Y el resultado de la ejecución de este código es que en la consola nos aparecerá el número 27.

Todas las clases numéricas son subclases de la clase *Number* como puede verse en el siguiente diagrama.



La siguiente tabla muestra los métodos disponibles para utilizar en la clase *Number* y sus descendientes.

Method	Description

`byte byteValue()`
`short shortValue()`
`int intValue()`
`long longValue()`
`float floatValue()`
`double doubleValue()`
	Converts the value of this `Number` object to the primitive data type returned.

`int compareTo(Byte anotherByte)`
`int compareTo(Double anotherDouble)`
`int compareTo(Float anotherFloat)`
`int compareTo(Integer anotherInteger)`
`int compareTo(Long anotherLong)`
`int compareTo(Short anotherShort)`
	Compares this `Number` object to the argument.

`boolean equals(Object obj)`	Determines whether this number object is equal to the argument. The methods return `true` if the argument is not `null` and is an object of the same type and with the same numeric value. There are some extra requirements for `Double` and `Float` objects that are described in the Java API documentation.

Cada subclase de la clase *Number* brinda métodos adecuados para la transformación de cadenas de caracteres (String) en datos numéricos primitivos. Estos métodos son estáticos y pueden utilizarse sin la necesidad de crear una instancia de la clase en cuestión para hacer la transformación. A continuación se muestran ejemplos de la transformación de String en números.

```
int b = Integer.parseInt("5");
int c = Integer.parseInt("23500");
long m = Long.parseLong("9384756178");
float f = Float.parseFloat("45.2349098");
System.out.println(b);
System.out.println(c);
System.out.println(m);
System.out.println(f);
```

El lenguaje Java incorpora soporte para realizar operaciones matemáticas básicas tales como sumas, restas, multiplicaciones, divisiones y divisiones modulares o residuales. Si necesitamos realizar operaciones más avanzadas, Java brinda la clase ***Math*** dentro del paquete java.lang. Todos los métodos de la clase *Math* son estáticos, por tanto, pueden ser invocados directamente en la clase como se muestra a continuación:

```
//el valor absoluto de un numero
Math.abs(f);
//la funcion trigonometrica seno de un angulo
Math.sin(c);
//la funcion trigonometrica coseno de un angulo
Math.cos(b);
```

La siguiente tabla muestra los métodos matemáticos básicos de la clase *Math*

Method	Description

`double abs(double d)`
`float abs(float f)`
`int abs(int i)`
`long abs(long lng)`
	Returns the absolute value of the argument.

`double ceil(double d)`	Returns the smallest integer that is greater than or equal to the argument. Returned as a double.
`double floor(double d)`	Returns the largest integer that is less than or equal to the argument. Returned as a double.
`double rint(double d)`	Returns the integer that is closest in value to the argument. Returned as a double.
`long round(double d)` `int round(float f)`	Returns the closest long or int, as indicated by the method's return type, to the argument.

`double min(double arg1, double arg2)`
`float min(float arg1, float arg2)`
`int min(int arg1, int arg2)`
`long min(long arg1, long arg2)`
	Returns the smaller of the two arguments.

`double max(double arg1, double arg2)`
`float max(float arg1, float arg2)`
`int max(int arg1, int arg2)`
`long max(long arg1, long arg2)`
	Returns the larger of the two arguments.

La siguiente tabla muestra los métodos exponenciales y logarítmicos de la clase *Math*

Method	Description
`double exp(double d)`	Returns the base of the natural logarithms, e, to the power of the argument.
`double log(double d)`	Returns the natural logarithm of the argument.
`double pow(double base, double exponent)`	Returns the value of the first argument raised to the power of the second argument.
`double sqrt(double d)`	Returns the square root of the argument.

La siguiente tabla muestra los métodos trigonométricos de la clase *Math*

Method	Description
`double sin(double d)`	Returns the sine of the specified double value.
`double cos(double d)`	Returns the cosine of the specified double value.
`double tan(double d)`	Returns the tangent of the specified double value.
`double asin(double d)`	Returns the arcsine of the specified double value.
`double acos(double d)`	Returns the arccosine of the specified double value.
`double atan(double d)`	Returns the arctangent of the specified double value.
`double atan2(double y, double x)`	Converts rectangular coordinates `(x, y)` to polar coordinate `(r, theta)` and returns `theta`.
`double toDegrees(double d)` `double toRadians(double d)`	Converts the argument to degrees or radians.

2. CADENAS

Los objetos de la clase String son una secuencia de caracteres en el lenguaje Java. La clase String es ampliamente utilizada dentro de los programas Java ya que brindan la posibilidad de manipular con facilidad estas secuencias de caracteres.

La forma más directa de crear un objeto de la clase String es la siguiente:

```
String saludo = "Hola a todos";
```

Donde entre comillas dobles escribimos el contenido que va a tener nuestra nueva cadena.

Para obtener la longitud de una cadena de caracteres se debe utilizar el método *length()* de la clase String

2.1.Concatenación de Cadenas

La concatenación de cadenas es el proceso mediante el cual se unen dos cadenas de caracteres para formar una sola. La clase String provee el método *concat(String string2)* para realizar esta operación.

La forma más fácil de concatenar dos cadenas es mediante la utilización del operador + como se muestra en los siguientes ejemplos.

En las dos primeras líneas de códgo se realiza la declaración y construcción de dos objetos de la clase String. Las siguientes dos líneas en el código concatenan estas cadenas obteniéndose el mismo resultado con las dos operaciones.

```
String saludo = "Hola a todos";
String destino = " los aprendices SENA";

String saludoTotal = saludo + destino;
String saludoTotal2 = saludo.concat(destino);
```

2.2.Convertir números en String

La conversión de números en String se realiza mediante la utilización del operador + o mediante el método estático *valueOf()* de la clase String. A continuación se muestran un ejemplo donde se obtiene el mismo resultado utilizando las dos formas.

```
int numero = 5;
String s1 = "" + numero;
String s2 = String.valueOf(numero);
```

Para la conversión de cualquier otro objeto en un objeto de la clase String se debe utilizar el método *toString()* brindado. El ejemplo siguiente muestra como se transforman los objetos de la clase *Number* en objetos de la clase String.

```
Integer entero1 = new Integer(4);
String cadena1 = entero1.toString();

Float flotante1 = new Float(4.5);
String cadena2 = flotante1.toString();
```

2.3.Manipulación de caracteres en un String.

Para obtener un carácter específico dentro de una cadena se debe utilizar el método *charAt(int index)* recordando que los índices en una cadena van desde 0 hasta lenght() – 1.

Si deseamos obtener una subcadena de una cadena más larga debemos utilizar el método *substring(int inicio, int fin)*

La siguiente tabla muestra los métodos disponibles para la manipulación de objetos de la clase String.

Method	Description
`String[] split(String regex)` `String[] split(String regex, int limit)`	Searches for a match as specified by the string argument (which contains a regular expression) and splits this string into an array of strings accordingly. The optional integer argument specifies the maximum size of the returned array. Regular expressions are covered in the lesson titled "Regular Expressions."
`CharSequence subSequence(int beginIndex, int endIndex)`	Returns a new character sequence constructed from `beginIndex` index up until `endIndex` - 1.
`String trim()`	Returns a copy of this string with leading and trailing white space removed.
`String toLowerCase()` `String toUpperCase()`	Returns a copy of this string converted to lowercase or uppercase. If no conversions are necessary, these methods return the original string.

La siguiente tabla muestra los métodos disponibles para la búsqueda de caracteres dentro de un objeto de la clase String.

Method	Description
`int indexOf(int ch)` `int lastIndexOf(int ch)`	Returns the index of the first (last) occurrence of the specified character.
`int indexOf(int ch, int fromIndex)` `int lastIndexOf(int ch, int fromIndex)`	Returns the index of the first (last) occurrence of the specified character, searching forward (backward) from the specified index.
`int indexOf(String str)` `int lastIndexOf(String str)`	Returns the index of the first (last) occurrence of the specified substring.
`int indexOf(String str, int fromIndex)` `int lastIndexOf(String str, int fromIndex)`	Returns the index of the first (last) occurrence of the specified substring, searching forward (backward) from the specified index.
`boolean contains(CharSequence s)`	Returns true if the string contains the specified character sequence.

La siguiente tabla muestra los métodos disponibles para realizar reemplazo de caracteres dentro de un objeto de la clase String.

Method	Description
`String replace(char oldChar, char newChar)`	Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
`String replace(CharSequence target, CharSequence replacement)`	Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.
`String replaceAll(String regex, String replacement)`	Replaces each substring of this string that matches the given regular expression with the given replacement.
`String replaceFirst(String regex, String replacement)`	Replaces the first substring of this string that matches the given regular expression with the given replacement.