SCJP – Capitulo 7

Paquete java.lang – La clase String, Math, Envoltorios y equals()

Agenda

- Usando la clase String
- Usando la clase Math
- Usando envoltorios (Wrapper)
- Usando el método equals()

La clase String

- Recordando : String es inmutable, NO SE PUEDE MODIFICAR…
- Cualquier operación sobre un objeto String, crea un nuevo String

```
String s = new String(“abc”);
s.concat(“xyz”); // nuevo String : “abcxyz”
```

La clase String

- Ejemplo:

1: String s1 = “spring ”;//asigna 1

2: String s2 = s1 + “summer ”; //asigna 2+3

3: s1.concat(“fall ”);//crea variable concatenada sin asignar 1

4: s2.concat(s1);//crea var +7

5: s1 += “winter ”;//crea var y asigna –nuevo valor para s1 +8

6: System.out.println(s1 + “ ” + s2);

¿Cuantos String son creados?

R: 8 (sin contar línea 6)

¿Cuál es la resultado?

R: spring winter spring summer

String y Memoria

- Para hacer más eficiente el uso de la memoria y evitar la redundancia de Literales String. JVN utiliza un área especia llamado “String constant pool”
  - La primera vez que se encuentra un literal String se agrega al “pool”
  - La segunda vez que se encuentra el mismo literal, se referencia al objeto antes encontrado… (por eso debe ser inmutable)

Métodos importantes

- public char charAt(int index)
- public String concat(String s)
- public boolean equalsIgnoreCase(String s)
- public int length()
- public String replace(char old, char new)
- public String substring(int begin)
- public String substring(int begin, int end)
- public String toLowerCase()
- public String toString()
- public String toUpperCase()
- public String trim()

La clase StringBuffer

- Es un String modificable, que maneja internamente un “Pool de String”…

```
StringBuffer sb = new StringBuffer(“abc”);
sb.append(“xyz”);// no necesario asignar
System.out.println( “sb : ” + sb ); // sb : abcxyz
```

Nota: recordar que cuando se concatena o “imprime” un objeto, se invoca implícitamente el metodo toString()

Métodos importantes

- public synchronized StringBuffer append(String s)
- public synchronized StringBuffer insert(int offset, String s)
- public synchronized StringBuffer reverse()
- public synchronized StringBuffer toString()

Todos los metodos actualizan el objeto StringBuffer y también lo retornan, esto permite realizar las cadenas de métodos

StringBuffer sb = new StringBuffer(“abc”); //crea sb y asigna abc
sb.append(“def”).reverse().insert(3, “- - -”); //se + def //luego lo reversa fedcba// e inserta en pos 4 - - - =“fed---cba”
System.out.println(sb); // fed- - -cba

Los métodos se ejecutan de izquierda a derecha
IMPORTANTE: StringBuffer no sobrescribe el método equals( )

La clase Math

- Provee acceso a operaciones matemáticas básicas
- Se encuentra en el paquete java.lang (el cual es importado automáticamente
- Todos sus métodos son *static*, se pueden utilizar simplemente con el nombre de la clase.
- No es necesario crear una instancia del objeto Math (en realidad no se puede, ya que su constructor es *private*)

Métodos importantes

- abs() : retorna el valor absoluto
  - public static int abs(int a)
  - public static long abs(long a)
  - public static float abs(float a)
  - public static double abs(double a)

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

Métodos importantes

- ceil() : retorna un entero superior o igual (como double)
  - public static double ceil(double a)

```
Math.ceil(9.0); // salida : 9.0 // siempre devuelve el entero pròximo superior en formato double
Math.ceil(8.8); // salida : 9.0
Math.ceil(8.02); // salida : 9.0

Math.ceil(-9.0); // salida : -9.0 // siempre devuelve el entero negativo pròximo superior (pero negativo por lo que es el"inferior")en formato double
Math.ceil(-9.4); // salida : -9.0
Math.ceil(-9.8); // salida : -9.0
```

Métodos importantes

- floor() : retorna un entero inferior o igual (como double)
  - public static double floor(double a)

```
Math.floor(9.0); // salida : 9.0
Math.floor(9.4); // salida : 9.0
Math.floor(9.8); // salida : 9.0

Math.floor(-9.0); // salida : -9.0
Math.floor(-8.8); // salida : -9.0
Math.floor(-8.1); // salida : -9.0
```

Métodos importantes

- max() : retorna el mayor de dos argumentos
  - 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)

```
Math.max(1024, -5000); // max : 1024
```

Métodos importantes

- min() : retorna el menor de dos argumentos
  - 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)

```
Math.min(0.5, 0.0); // min : 0.0
```

Métodos importantes

- random() : devuelve numeros aleatorios mayores o iguales a 0.0 y menores a 1.0
  - public static double random( )
- round() : redondea números realizando la siguiente operación:

  Math.floor(a + 0.5);
  - public static int round(float a)
  - public static long round(double a)

Métodos importantes

- Funciones trigonometriítas: Todas reciben un ángulo en radianes
  - public static double sin(double a)
  - public static double cos(double a)
  - public static double tan(double a)
- toDegrees() : convierte decimales a radianes
  - public static double toDegrees(double a)
- toRadians() : convierte radianes a decimales:
  - public static double toRadians(double a)

```
Math.sin(Math.toRadians(90.0)) // salida : 1.0
```

Métodos importantes

- sqrt(): Retorna la raíz cuadrada
  - public static double sqrt(double a)

```
Math.sqrt( 9.0 ) // salida : 3.0
Math.sqrt(-9.0) // salida : NaN
```

Cuando se ingresa un numero negativo, retorna un NaN (not a number).

Observaciones

- NaN : no es igual a nada (ni siquiera a el mismo)
- La única forma de saber si un mumero es NaN es por medio de la utlidad de la clase Double.isNaN();
- Math.E y Math.PI : constantes útiles.

Envoltorios

- Proveen un mecanismo de “envolver” un valor primitivo en un objeto, con esto se pueden utilizar en actividades reservadas sólo para objetos (agregar a colecciones, ser retornados en métodos de tipo Object, etc)
- Proveen un surtido grupo de utilidades: conversiones desde y hacia String, convertir primitivos y String hacia y desde diferentes bases (octal, hexa, binario)

Envoltorios

Primitivo	Clase	Argumento del constructor
boolean	Boolean	boolean o String
byte	Byte	byte o String
char	Character	char
double	Double	double o String
float	Float	float, double o String
int	Integer	int o String
long	Long	long o String
short	Short	short o String

Envoltorios : valueOf()

- Método *static* que permite convertir un String a un Envoltorio (puede lanzar excepción NumberFormatException)

Float f1 = Float.valueOf(“3.14f”);

O

Integer i1 = Integer.valueOf(“101011”, 2);

Método valueOf(String, int) sólo esta disponible en los envoltorios de primitivos enteros

Envoltorios : xxxValue()

- Método que permite extraer el valor del envoltorio numerico hacia un primitivo numerico. (son 36 en total, 6 Wrapper * 6 primitivos)

```
Float f1 = Float.valueOf("3.14f");

byte b = f1.byteValue();
short s = f1.shortValue();
int i = f1.intValue();
long l = f1.longValue();
float f = f1.floatValue();
double d = f1.doubleValue();
```

Convierte el objeto envoltorio (Wrapper) al tipo primitivo (realiza internamente el casting)

Envoltorios : parseXxx()

- Método *static* que permite convertir un String a un tipo de dato primitivo (puede lanzar excepción NumberFormatException)

```
double d = Double.parseDouble(“6.5”);
int i = Integer.parseInt(“77”);
```

toString() y toXxxString()

- 1. Método (heredado de la clase Object) que permite representar el valor primitivo en un String
- 2. Método *static* que permite convertir un primitivo a un String (sólo envoltorio numéricos)
- 3. Método *static* que permite convertir un número en base 10 hacia otra base (sólo para Integer y Long)
- 4. Método *static* que permite convertir un número en base 10 hacia otra base predefinida (Binary, Hexa, Octal)

```
1. String s1 = new Boolean(true).toString(); // “true”
2. String s2 = Double.toString(77); // “77”
3. String s3 = Integer.toString(77, 2); // “1001101”
4. String s4 = Integer.toBinaryString(77) ; // “1001101”
```

Método : equals( )

- Para compara primitivos se utiliza operador de igualdad : ==
- Para comparar si dos variables de referencia apuntan al mismo objeto, se utiliza el operador de igualdad: == (deben ser del mismo tipo)
- Para comparar si dos objetos tienen el mismo contenido se utiliza el método equals( ).

Método : equals( )

```
String str1 = “ABC”;
String str2 = “A” + “B” + “C”;
Integer int1 = new Integer(5);
Double dbl1 = new Double(5.0);

(str1 == str2) // false
str1.equals(str2) // true
(dbl1 == int1) // Erro de compilación
                                    //son distinto obj
dbl1.equals(int1) // false
```

String s1 = “abc”;

String s2 = “abc”;

Las siguientes sentencias se evaluan como verdaderas

(s1 == s2)

s1.equals(s2)

¿Por qué? …