PROGRAMACIÓN EN JAVA

TEMA ANEXO 1: Clases de JAVA

Bibliografía

- “JAVA 2 Lenguaje y aplicaciones”

Fco. Javier Ceballos. Ed. Ra-Ma, 2006

Contenido

- **Math**
- **Character**
- **Float**
- **Double**
- **Integer**
- **Long**
- **Boolean**
- **String**
- **StringBuffer**

Clase Math

- La biblioteca de clases de JAVA incluye una clase llamada *Math* (No se pueden crear instancias de esta clase) en su paquete *java.lang*, la cual define un conjunto de operaciones matemáticas de uso común que pueden ser utilizadas por cualquier programa

- La clase *math* contiene métodos para ejecutar operaciones numéricas elementales tales como raíz cuadrada, exponencial, logaritmo, y funciones trigonométricas. Por ejemplo:

```
double raíz_cuadrada , n = 345.0;
raíz_cuadrada = Math.sqrt(n);
System.out.println("La raiz cuadrada de "+ n +" es "+ raíz_cuadrada);
```

- La tabla siguiente resume los miembros de la clase *Math*. Todos los miembros de esta clase son *static* para que puedan ser invocados sin necesidad de definir un objeto de la clase

Valor de retorno	Atributo/método	Descripción
	static double E	Valor del número e
double	Math.PI	Valor del número π
tipo	Math.abs(tipo *a*)	Valor absoluto de *a*
double	Math.ceil(double *a*)	Devuelve el entero inmediato inferior
double	Math.floor(double *a*)	Devuelve el entero inmediato superior

Clase Math (cont.)

Valor de retorno	Atributo/método	Descripción
tipo	Math.min(tipo *a*, tipo *b*)	Devuelve el valor menor de *a* y *b*
tipo	Math.max(tipo *a*, tipo *b*)	Devuelve el valor mayor de *a* y *b*
double	Math.random()	Devuelve un valor aleatorio mayor o igual que 0.0 y menor que 1.0
double	Math.rint(double *a*)	Devuelve el valor double sin decimales más cercano a *a* (redondeo de *a*)
long	Math.round(double *a*)	Devuelve el valor long más cercano a *a*
int	Math.round(float *a*)	Devuelve el valor int más cercano a *a*
double	Math.sqrt(double *a*)	Devuelve la raíz cuadrada de *a* (*a* no puede ser negativo)
double	Math.exp(double *a*)	Valor de e^a
double	Math.log(double *a*)	Devuelve el logaritmo natural de *a*
double	Math.pow(double *a*, double *b*)	Devuelve el valor de a^b
double	Math.IEEEremainder(double *f1*, double *f2*)	Devuelve el resto de una división entre números reales: *c=f1/f2*, siendo *c* el valor entero más cercano al valor real de *f1/f2*; por lo tanto, el resto puede ser positivo o negativo
double	Math.acos(double *a*)	Arco, de 0.0 a ∏, cuyo coseno es *a*

Clase Math (cont.)

Valor de retorno	Atributo/método	Descripción
double	Math.asin(double *a*)	Arco, de -∏/2 a ∏/2, cuyo seno es *a*
double	Math.atan(double *a*)	Arco, de -∏/2 a ∏/2, cuya tangente es *a*
double	Math.atan2(double *a*, double *b*)	Convierte las coordenadas rectangulares (*b*,*a*) a polares: (*r*,*Θ*)
double	Math.sin(double *a*)	Seno de *a* radianes
double	Math.cos(double *a*)	Coseno de *a* radianes
double	Math.tan(double *a*)	Tangente de *a* radianes
double	Math.toDegrees(double *rads*)	Convertir un ángulo en radianes a grados
double	Math.toRadians(double *grados*)	Convertir un ángulo en grados a radianes

- **NOTA:** tipo puede ser: double, float, int o long

Clase Math – Ejemplo

- Programa que da como resultado las soluciones reales x1 y x2 de una ecuación de 2do grado, de la forma:

$$ax^2 + bx + c = 0$$

- Las soluciones de una ecuación de 2do grado vienen dadas por la fórmula:

$$X_i = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Las soluciones son reales sólo si b^2-4ac es mayor o igual que 0

```
public class Cecuacion {
   public static void main(String[] args) {
      double a, b, c, d, x1, x2;

      System.out.print("Coeficiente a: ");
      a = Leer.datoDouble();
      System.out.print("Coeficiente b: ");
      b = Leer.datoDouble();
      System.out.print("Coeficiente c: ");
      c = Leer.datoDouble();

      d = b * b - 4 * a * c;

      if (d < 0) {
         System.out.println("\nLas raices son complejas.");
         return;
      }

      System.out.println("\nLas raices reales son:");
      d = Math.sqrt(d);
      x1 = (-b + d) / (2 * a);
      x2 = (-b - d) / (2 * a);
      System.out.println("x1 = " + x1 + ", x2 = " + x2);
   }
}
```

Clase Math – Ejemplo (cont.)

- Programa sencillo que utiliza algunos de los atributos/métodos vistos en las tablas anteriores

```
public class Mates
{
   public static void main(String[] args)
   {
      int x;
      double rand, y, z;
      float max;
      rand = Math.random();
      x = Math.abs(-123);
      y = Math.round(123.567);
      z = Math.pow(2,4);
      max = Math.max((float)1e10,(float)3e9);
      System.out.println("rand= "+rand);
      System.out.println("x= "+x);
      System.out.println("y= "+y);
      System.out.println("z= "+z);
      System.out.println("max= "+max);
   }
}
```

Clase Character

- Contiene muchas funciones de comprobación y traslación
- De esta clase sí se pueden crear instancias

Comprobaciones booleanas

Character.isLowerCase(*c*)	Determina si el carácter está en minúscula
Character.isUpperCase(*c*)	Determina si el carácter está en mayúscula
Character.isDigit(*c*)	Determina si el carácter es un dígito
Character.isSpace(*c*)	Determina si *c* es un espacio en blanco

Translaciones de caracteres

char c2 = Character.toLowerCase(*c*)	Convierte a minúscula
char c2 = Character.toUpperCase(*c*)	Convierte a mayúsculas

Translaciones de carácter/dígito

int i = Character.digit(*c*, *base*)	Devuelve el valor numérico de *c*, según la base especificada
char c = Character.forDigit(*i*, *base*)	Devuelve el carácter del digito *i*, según la base especificada

Clase Character (cont.)

Métodos de la clase Character

```
C = new Character('J');     //C será un objeto de la clase Chacracter
char c = C.charValue();     //c contendrá el valor de C (el carácter J)
String s = C.toString();    //s contendrá el valor de C (el carácter J) en forma de String
```

Clase Float

- Esta clase contiene los métodos para trabajar con objetos de tipo float

Valores de Float

```
Float.POSITIVE_INFINITY
Float.NEGATIVE_INFINITY
Float.NaN
Float.MAX_VALUE
Float.MIN_VALUE
```

Conversiones de clase/cadena

```
String s = Float.toString(f);
f = Float.valueOf(“3.14”);
```

Comprobaciones

```
boolean b= Float.isNaN(f);
boolean b = Float.isInfinite(f);
```

Clase Float (cont.)

- La función isNaN() comprueba si *f* es un No-Número (dato no numérico). Un ejemplo de un No-Número es raíz cuadrada de -2

Conversiones de objetos

```
Float F = new Float(Float.PI);
String s = F.toString();
int i = F.intValue();
long i = F.longValue();
float F = F.floatValue();
double d = F.doubleValue();
```

Otros métodos

```
int i = F.hashCode();                  //Retorna el valor hashCode
boolean b = F.equals(Object obj);      //Verifica si los objetos son iguales
int i = Float.floatToIntBits(f);       //Retorna la representación de bits
float f = Float.intBitsToFloat(i);     //Retorna el valor float del valor bit
```

Clase Double

Valores de Double

Double.POSITIVE_INFINITY
Double.NEGATIVE_INFINITY
Double.NaN
Double.MAX_VALUE
Double.MIN_VALUE

Métodos de Double

```
D.isNaN()                       //Verifica si es un número válido
Double.isNaN(d)
D.isInfinite()                  //Retorna true si es un número infinito
Double.isInfinite(d)
boolean D.equals(Object obj)    //Comprueba si los objetos son iguales
String D.toString()             //Convierte a cadena el valor double
int D.intValue()                //Convierte a entero el valor double
long D.longValue()              //Convierte a long el valor double
```

Clase Double (cont.)

```
float D.floatValue()
double D.doubleValue()
int i = D.hashCode()
Double V.valueOf(String s)
long l = Double.doubleToLongBits(d)
double d = Double.longBitsToDouble(l)
```

Clase Integer

Valores de Integer

Integer.MAX_VALUE
Integer.MIN_VALUE

Métodos de Integer

String Integer.toString(int i, int base)
String Integer.toString(int i)
int I.parseInt(String s, int base)
int I.parseInt(String s)
Integer Integer.valueOf(String s, int base)
Integer Integer.valueOf(String s)
int I.intValue()
long I.longValue()
float I.floatValue()

Clase Integer (cont.)

double I.doubleValue()
String I.toString()
int I.hashCode()
boolean I.equals(Object obj)

- En los métodos toString(), parseInt() y valueOf() que no se especifica la base sobre la que se va a trabajar, se asume que se trabaja con la base 10

Clase Long

Valores de Long

Long.MAX_VALUE
Long.MIN_VALUE

Métodos de Long

String Long.toString(long l, int base)
String Long.toString(long l)
long L.parseLong(String s, int base)
long L.parseLong(String s)
Long Long.valueOf(String s, int base)
Long Long.valueOf(String s)
int L.intValue()
long L.longValue()
float L.floatValue()

Clase Long (cont.)

double L.doubleValue()
String L.toString()
int L.hashCode()
boolean L.equals(Object obj)

- En los métodos toString(), parseLong() y valueOf() que no se especifica la base sobre la que se va a trabajar, se asume que se trabaja con la base 10

Clase Boolean

Valores de Boolean

Boolean.TRUE
Boolean.FALSE

Métodos de Boolean

boolean B.booleanValue()
String B.toString()
boolean B.equals(Object obj)

Clase String

- Una cadena de caracteres en JAVA es un objeto de la clase *String*. Para entenderlo podemos pensar en un objeto *String* como en una cadena de caracteres almacenada en una matriz unidimensional de elementos de tipo *char*. Por ejemplo:

```
char [] cadena = new char [40];
```

- La clase *String*, que pertenece al paquete *java.lang*, proporciona métodos para examinar caracteres individuales de una cadena de caracteres, comparar cadenas, buscar y extraer sub-cadenas, copiar cadenas y convertir a mayúsculas o a minúsculas. Es necesario saber que un objeto *String* representa una cadena de caracteres no modificable. Por lo tanto, una operación como convertir a mayúsculas no modificará el objeto original sino que devolverá un nuevo objeto con la cadena resultante de esa operación

- JAVA proporciona el operador + para concatenar objetos *String*, así como soporte para convertir otros objetos a objetos *String*. Por ejemplo, en la siguiente línea de código, JAVA debe convertir las expresiones que aparecen entre paréntesis en objetos *String*, antes de realizar la concatenación

```
System.out.println("Dimension de la matriz: " + cadena.length);
```

Clase String (cont.)

Constructores

String()
String(String str)
String(char val[])
String(char val[], int offset, int count)
String(byte val[], int hibyte)
String(byte val[], int hibyte, int offset, int count)

Funciones básicas

int length() /*Devuelve la longitud o número de caracteres UNICODE del objeto *String* que recibe el mensaje *length*/

char charAt(int indice) /*Devuelve el carácter que está en la posición especificada en el objeto *String* que recibe el mensaje *charAt*. El parámetro indice debe estar entre 0 y *length()-1*/

Clase String (cont.)

Funciones de comparación de Strings	
boolean equals(Object obj)	/*Permite verificar si dos referencias se refieren a un mismo objeto*/
boolean equalsIsIgnoreCase(Object obj)	
int compareTo(String str2)	/*Compara lexicográficamente el *String* especificado, con el objeto *String* que recibe el mensaje *compareTo* (*equals* realiza la misma operación)*/

Funciones de comparación de sub-cadenas	
boolean regionMatches(int toffset, String other, int ooffset, int len)	
boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)	
boolean startsWith(String prefix)	/*Devuelve true si el *prefijo* especificado coincide con el principio del objeto *String* que recibe el mensaje *startsWith*/
boolean startsWith(String prefix, int offset)	
boolean endsWith(String suffix)	/*Devuelve true si el *sufijo* especificado coincide con el final del objeto *String* que recibe el mensaje *endsWith*/

Clase String (cont.)

int indexOf(int car)	/*Devuelve el índice de la primera ocurrencia del carácter especificado por *car*. Devuelve -1 si *car* no existe*/
int indexOf(int car, int fromIndex)	
int indexOf(String str)	/*Devuelve el índice de la primera ocurrencia de la subcadena especificada por *str*. Devuelve -1 si *str* no existe*/
int indexOf(String str, int fromIndex)	
int lastIndexOf(int car)	/*Igual que indexOf, pero la búsqueda comienza por el final*/
int lastIndexOf(int car, int fromIndex)	
int lastIndexOf(String str)	/*Igual que indexOf, pero la búsqueda comienza por el final*/
int lastIndexOf(String str, int fromIndex)	
String substring(int beginIndex)	
String substring(int beginIndex, int endIndex)	
String concat(String str)	
String replace(char oldChar, char newChar)	/*Devuelve un nuevo *String* resultado de reemplazar todas las ocurrencias *oldChar* por *newChar*. Si *oldChar* no existiera, entonces se devuelve el objeto *String* original*/
String toLowerCase()	/*Convierte a minúsculas las letras mayúsculas. El resultado es un nuevo *String* en minúsculas*/
String toUpperCase()	/*Convierte a mayúsculas las letras minúsculas. El resultado es un nuevo *String* en mayúsculas*/
String trim()	/*Devuelve un objeto *String* resultado de eliminar los espacios en blanco que pueda haber al principio y al final*/

Clase String (cont.)

void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)	
String toString()	/*Devuelve el propio objeto *String* que recibe el mensaje *toString*. Es decir, que se hace una copia de la referencia (no se crea un objeto nuevo)*/
char[] toCharArray()	/*Devuelve una matriz de caracteres creada a partir del objeto *String* que recibe el mensaje *toCharArray**/

Funciones ValueOf	
static String valueOf(tipo dato)	/*Devuelve un nuevo *String* creado a partir del *dato* pasado como argumento. Puesto que el método es *static* no necesita ser invocado por un objeto *String*. El argumento puede ser de los tipos: *boolean*, *char*, *char[]*, *int*, *long*, *float*, *double* y *Object**/
String valueOf(char[] data, int offset, int count)	
String copyValueOf(char[] data)	
String copyValueOf(char[] data, int offset, int count)	

Clase StringBuffer

- Un objeto de la clase *String* no es modificable
- Los métodos que actúan sobre un objeto *String* con la intención de modificarlo, no lo hacen, sino que devuelven un objeto nuevo con las modificaciones solicitadas
- En cambio, un objeto *StringBuffer* es un objeto modificable tanto en contenido como en tamaño

Constructores

StringBuffer()
StringBuffer(int length)
StringBuffer(String str)

Métodos de StringBuffer

int length()
char charAt(int index)
void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
String toString()
void setLength(int newLength)
void setCharAt(int Index, char car)
int capacity()

Clase StringBuffer (cont.)

Funciones para cambiar el contenido	
StringBuffer append(tipo x)	/*Este método permite añadir la cadena de caracteres resultante de convertir el argumento *x* en un objeto *String*, al final del objeto *StringBuffer* que recibe el mensaje *append*. La longitud del objeto *StringBuffer* se incrementa en la longitud correspondiente al *String* añadido*/
StringBuffer insert(int indice, tipo x)	/*Este método permite insertar la cadena de caracteres resultante de convertir el argumento *x* en un objeto *String*, en el objeto *StringBuffer* que recibe el mensaje *insert*. Los caracteres serán añadidos a partir de la posición especificada por el argumento *indice*. La longitud del objeto *StringBuffer* se incrementa en la longitud correspondiente al *String* insertado*/
StringBuffer delete(int p1, int p2)	/*Este método elimina los caracteres que hay entre las posiciones *p1* y *p2-1* del objeto *StringBuffer* que recibe el mensaje *delete*. El valor *p2* debe ser mayor que *p1*. Si *p1* es igual que *p2*, no se efectuará ningún cambio y si es mayor JAVA lanzará una excepción*/
StringBuffer reverse()	/*Este método reemplaza la cadena almacenada en el objeto *StringBuffer* que recibe el mensaje *reverse*, por la misma cadena pero invertida*/

- **NOTA:** tipo puede ser: *boolean*, *char*, *char[]*, *int*, *long*, *float*, *double*, *String* y *Object*