UNIVERSIDAD CATOLICA DE COLOMBIA
FACULTAD DE INGENIERIA DE SISTEMAS

CURSO: JAVA BASICO
PROFESOR: EMERSON CASTAÑEDA SANABRIA

TEMA: Clases de utilidad

OBJETIVOS:

- Obtener un manejo de las clases para trabajar cadenas de caracteres.
- Aprender la forma de uso y la utilidad de las clases Wrappers.
- Conocer algunas de las clases de utilidad proporcionadas por el lenguaje.

CONTENIDO:

1. Las clases String y StrinBuffer
2. Wrappers
3. La clase Math
4. Clases del paquete java.util

DESARROLLO:

1. Las clases String y StrinBuffer

El paquete java.lang contiene dos clases de cadenas: String y StringBuffer. La clase String se utiliza cuando se trabaja con cadenas que no pueden cambiar. Por otro lado, StringBuffer, se utiliza cuando se quiere manipular el contenido de una cadena.

La clase String

La clase String representa cadenas de caracteres. Todas las expresiones en java , como "abc", son implementadas como instancias de esta clase.

Los objetos Strings son constantes; es decir, son valores que no pueden cambiar después de ser creados. Por ejemplo:

```
String str = "abc";
```

Es equivalente a:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

La clase String incluye métodos para examinar caracteres individualmente, para comparar cadenas, para buscar cadenas, para extraer subcadenas y para crear una copia de una cadena con todos sus caracteres convertidos a mayúscula o a minúscula.

Java proporciona un soporte especial para el operador de concatenación de cadenas (+), y la conversión de otros objetos a objetos String (a través del método toString()).

Algunos Constructores
String() inicializa un Nuevo objeto String que representa una cadena vacía.
String(byte[] bytes)

Construye una nueva cadena decodificando el arreglo de bytes especificado, utilizando el conjunto de caracteres por defecto de la plataforma.
String(byte[] bytes, String charsetName) Construye una nueva cadena decodificando el arreglo de bytes especificado, utilizando el conjunto de caracteres especificado por charsetName.
String(char[] value) Contruye una nueva cadena que representa el arreglo de caracteres indicado..
String(String original) Inicializa una nueva cadena que contiene la misma secuencia de caracteres que el argumento especificado.
String(StringBuffer buffer) Inicializa una nueva cadena que contiene la misma secuencia de caracteres que el argumento buffer.

Algunos métodos	
char	**charAt**(int index) Retorna el carácter del índice especificado.
int	**compareTo**(Object o) Compara la cadena con otro objeto.
int	**compareTo**(String anotherString) Compara dos cadenas lexicograficamente.
int	**compareToIgnoreCase**(String str) Compara dos cadenas ignorando las diferencias lexicograficas.
String	**concat**(String str) Concadena la cadena especificada al final de la cadena.
byte[]	**getBytes**() Codifica la cadena en un arreglo de bytes utilizando el conjunto de caracteres por defecto de la plataforma.
int	**hashCode**() Retorna el código hash para la cadena.
int	**indexOf**(int ch) Retorna el índice de la primera ocurrencia del carácter especificado dentro de la cadena.
int	**lastIndexOf**(int ch) Retorna el índice de la ultima ocurrencia del carácter especificado dentro de la cadena.
int	**length**() retorna la longitud de de la cadena.
String	**replace**(char oldChar, char newChar) Retorna una nueva cadena remplazando las ocurrencias del oldChar por el newChar.
String	**substring**(int beginIndex) Retorna una subcadena de la cadena desde la posición especificada.

La clase StringBuffer

Stringbuffer implementa una secuencia de caracteres que puede cambiar. Es como un String con la diferencia que puede ser modificada en cualquier momento a treves de los métodos que incorpora.

String buffers are safe for use by multiple threads. The methods are synchronized where necessary so that all the operations on any particular instance behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads involved.

La clase StringBuffer es usada por el compilador para implementar la concatenación de cadenas con el operador +. Por ejemplo, el codigo:

```
x = "a" + 4 + "c"
```

Es compilado con su código equivalente:

```
x = new StringBuffer().append("a").append(4).append("c")
.toString()
```

Las principales operaciones de StringBuffer son la adición y la inserción. Estas están redefinidas para aceptar datos de cualquier tipo.

Constructores
StringBuffer() Construye un StringBuffer sin caracteres y con una capacidad inicial de 16.
StringBuffer(int length) Construye un StringBuffer sin caracteres y con la capacidad inicial especificada en el argumento.
StringBuffer(String str) Construye un StringBuffer que representa la misma secuencia del caracteres del argumento.

Algunos Métodos	
StringBuffer	**append**(boolean b) Adiciona la representación en cadena de un argumento booleano a un StringBuffer.
StringBuffer	**append**(char c) Adiciona la representación de un argumento char a un StringBuffer.
StringBuffer	**append**(char[] str) Adiciona la representación de un argumento tipo arreglo de caracteres a un StringBuffer.
StringBuffer	**append**(double d) Adiciona la representación en cadena de un argumento double a un StringBuffer.
StringBuffer	**append**(Object obj) Adiciona la representación en cadena de un argumento Objeto a un StringBuffer.

int	**capacity**() Retorna la actual capacidad de un StringBuffer.
StringBuffer	**delete**(int start, int end) Remueve la subcadenas especificada en el StringBuffer.
StringBuffer	**insert**(int offset, boolean b) Inserta la representacion de un argumento booleano en el stringBuffer.
StringBuffer	**insert**(int offset, char c) Inserta la representacion de un argumento character en el stringBuffer.
int	**length**() Retorna la longitude del StringBuffer.

2. Wrappers

Java trabaja con dos tipos de entidades: datos primitivo y referencias a objetos. Java no utiliza los objetos para sustituir a la mayoría de los tipos primitivos (los números, boléanos y caracteres), sobre todo para mayor efectividad. La manipulación de los tipos primitivos independientes es más eficaz. Pero existe la posibilidad de agrupar los tipos primitivos. Por ejemplo podemos crear una clase cuya finalidad sea englobar un entero. De esta forma podremos trabajar con nuestros tipos primitivos de siempre como si fueran objetos.

El paquete java.lang contiene las siguientes clases envolventes: Integer, Long, Byte, Short, Float, Double, Character, Boolean y Void.

La clase Integer

La clase Integer envuelve un valor de un dato primitiva tipo int en un objeto. Un objeto de tipo Integer contiene un único atributo que es de tipo int.

Adicionalmente, la clase proporciona métodos para convertir un entero a una cadena y una cadena a un entero.

Constructores
Integer(int value) Construye un nuevo objeto Integer que representa el valor especificado en el parámetro entero value.
Integer(String s) Construye un nuevo objeto Integer que representa el valor especificado en el parámetro tipo cadena s.

Algunos Métodos	
byte	**byteValue**() Retorna el valor del Integer como un byte.
int	**compareTo**(Integer anotherInteger) Conpara dos objetos Integer numericamente.
double	**doubleValue**()

	Retorna el valor del Integer como un double.
int	**intValue**() Retorna el valor del Integer como un entero.
long	**longValue**() Retorna el valor del Integer como un long.
Static int	**parseInt**(String s) Convierte el argumento s a un entero.
String	**toString**() Retorna la representación en cadena del objeto Integer.

3. La clase Math

La clase Math agrupa una serie de funciones matemáticas que se pueden clasificar de la siguiente forma:

Funciones límite, máximo, mínimo, valor absoluto.

Funciones cuadráticas, raíces, potencias, logaritmos y exponenciales. Todas ellas trabajan con valores double.

Funciones trigonométricas (sen, cos, tan, asen, acos, atan). Todas estas funciones trabajan con ángulos medidos en radianes en vez de grados.

Función generadora de números aleatorios. Contienen el método random() que se utiliza para efectuar aplicaciones aleatorias.

4. clases del paquete java.util

La clase Vector

La clase java.util.Vector provee la funcionalidad equivalente a un arreglo, con las siguientes diferencias principales:

Crece dinámicamente, o sea, puede contener un número ilimitado de elementos, a diferencia del arreglo, en el que se especifica su tamaño al crearse.

Sólo almacena Objetos: no permite almacenar tipos básicos (double, int, char,…) Si se desea almacenar un tipo básico se deben usar las clases envolventes respectivas.

Cuando se extraen los elementos, estos salen como referencias de tipo Object. La clase java.lang.Object es la raíz del árbol de herencia de Java, por lo cual es la superclase padre de todas las clases. Si se necesita acceder el elemento extraído del Vector con otro tipo de referencia, debe usarse el operador de casting para cambiar el tipo de dato de la referencia.

Constructores
Vector() Construye un vector vacio con capacidad inicial de 10.
Vector(int initialCapacity) Construye un vector vacio con capacidad inicial especificada.
Vector(int initialCapacity, int capacityIncrement) Construye un vector vació con la capacidad inicial y el factor de incremento especificados.

Method Summary	
void	**add**(int index, Object element) Inserta el objeto en la posición especificada del vector.
boolean	**add**(Object o) Adiciona el objeto al final del vector.
void	**addElement**(Object obj) Adiciona el objeto al final del vector, incrementando la capacidad del vector en 1.
int	**capacity**() Retorna la actual capacidad del vector.
void	**clear**() Elimina todos los elementos almacenados en el vector.
void	**copyInto**(Object[] anArray) Copia todos los elementos del vector en el arreglo de objetos especificado.
Object	**elementAt**(int index) Retorna el elemento almacenado en la posición especificada.
Enumeration	**elements**() retorna una enumeracion que contiene todos los elementos del vector.
Object	**firstElement**() Retorna el primer elemento del vector (el de indice 0)
Object	**get**(int index) Retorna el elemento almacenado en la posición especificada.
int	**indexOf**(Object elem) Busca la primer ocurrencia de un argumento especificado dentro del vector, retorna -1 en caso de no encontrarlo.

La clase Hashtable

La clase Hashtable extiende Dictionary (abstract) e implementa Cloneable y Serializable. Es una tabla que relaciona una clave con un valor. Cualquier objeto distinto de null puede ser tanto clave como valor. La clase a la que pertenecen las claves debe implementar los métodos hashcode() y equals().

Cada objeto de Hashtable tiene dos variables: capacity y load factor (entre 0.0 y 1.0). Cuando el número de elementos excede el producto de estas variables, la Hashtable crece llamando al método rehash(). Un load factor más grande utiliza mejor la memoria, pero será menos eficiente. Es conveniente partir de una Hashtable suficientemente grande para no tener que estar ampliando continuamente.

Constructores
Hashtable() Construye un objeto hashtable vació, con capacidad inicial de 11 y un factor de carga de 0.75.
Hashtable(int initialCapacity) Construye un objeto hashtable vació, con la capacidad inicial especificada y un factor de carga de 0.75.

Hashtable(int initialCapacity, float loadFactor) Construye un objeto hashtable vació, con la capacidad inicial y el factor de carga de especificados.

Method Summary	
void	**clear**() Limpia el objeto hashtable.
boolean	**contains**(Object value) verifica si alguna llave de la tabla tiene asociado el objeto especificado.
boolean	**containsKey**(Object key) verifica si la llave esta almacenada en el objeto hashtable.
boolean	**containsValue**(Object value) Retorna verdadero si la tabla tiene almacenado uno o mas claves con este valor.
Enumeration	**elements**() Retorna una enumeración con los valores almacenados en la tabla.
Object	**get**(Object key) Retorna el valor almacenado con la llave especificada.
boolean	**isEmpty**() Verifica si la tabla esta vacia.
Enumeration	**keys**() Retorna una enumeración con las llaves almacenadas en la tabla.
Object	**put**(Object key, Object value) Coloca un una clave con su respectivo valor en la tabla.
Object	**remove**(Object key) retira una lave especificada de la tabla.
int	**size**() retorna el numero de llaves almacenadas en la tabla.

La clase Stack

La clase Stack representa una pila de objetos con comportamiento LIFO (last-in-first-out). Esta clase extiende de la clase Vector e implementa cinco operaciones para que el Vector sea tratado como una pila. Provee las operaciones usuales de push y pop para colocar y retirar elementos de la pila., como también un método peek para obtener el ultimo elemento de la pila sin retirarlo de la misma.

Adicionalmente cuanta con un método para verificar si la pila esta vacía y un método para buscar un elemento en la pila que retorna la posición del objeto.

Cuando un objeto Snack se crea, inicialmente se encuentra vació.

Constructor

Stack() Crea un objeto Stack vació.

Method Summary

boolean	**empty**() Verifica si la pila esta vacía.
Object	**peek**() Retorna el elemento superior de la pila sin retirarlo.
Object	**pop**() Remueve el objeto que esta en la parte superior de la pila y lo retorna.
Object	**push**(Object item) Coloca el objeto en el tope de la pila.
int	**search**(Object o) Retorna la posición en donde esta el objeto en la pila.