APRENDERAPROGRAMAR.COM

REPASO AL PAQUETE DE CLASES JAVA.LANG. INTRODUCCIÓN. CONCEPTO DE OBJETO INMUTABLE.(CU00911C)

Sección: Cursos
Categoría: Lenguaje de programación Java nivel avanzado I

Fecha revisión: 2039

Resumen: Entrega nº11 curso "Lenguaje de programación Java Nivel Avanzado I".

Autor: Manuel Sierra y José Luis Cuenca

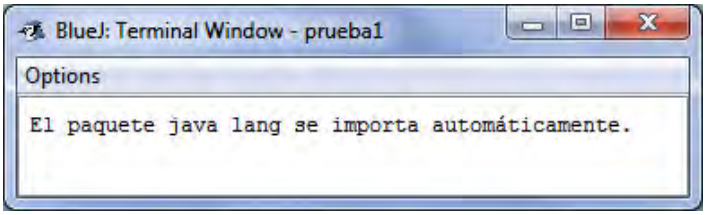
PAQUETE JAVA.LANG

Antes de nada queremos recordar que el paquete java.lang se importa automáticamente. Esto quiere decir que no es necesario poner en la clase donde se vaya a hacer uso de este paquete la cláusula o instrucción correspondiente "import java.lang". Dentro de este paquete están gran parte de las clases más utilizadas dentro de las aplicaciones o programas creados con tecnología Java.

Para ver lo anteriormente comentado hemos creado un ejemplo de código muy básico, como el siguiente:

```
/* Ejemplo Aprenderaprogramar.com */
public class Programa {
    public static void main(String arg[]) {
        String cadena = new String("El paquete java lang se importa automáticamente.");
        System.out.println(cadena);
    }
}
```

Donde vemos que en el código del Programa no ha sido necesario hacer uso de la clausula import java.lang para poder utilizar la clase String. Obtenemos la siguiente salida:



Ahora vamos a proceder a repasar muy rápidamente un poco tanto las interfaces, como las clases principales del paquete, así como las posibles excepciones y errores que se pueden producir. En los siguientes epígrafes haremos un repaso rápido y teórico para luego hacer ejercicios y ejemplos.

INTERFACES

Las interfaces que consideramos principales del paquete java.lang son las siguientes:

- **Cloneable**
- **Comparable**
- **Iterable**

Estas 3 interfaces nos definen los métodos necesarios a definir en nuestra clase para que puedan implementar dicha interfaz. Por un lado la interfaz Cloneable nos obliga a implementar el método denominado *clone()* que aparece en la clase Object y que sirve para crear una copia o clon de un objeto.

Comparable <K> nos obliga a implementar el método cuya cabecera es la siguiente:

int compareTo(claseK objeton)

Y que lo que debe hacer es comparar 2 objetos de la misma clase claseK y se debe devolver -1, 0 o 1 si el objeto que llama al método es respectivamente menor, igual o mayor que el objeto especificado en el método por objeton.

Por último la interfaz Iterable <T> nos obligaría a definir el método siguiente:

Iterator<T> iterator()

Donde deberemos de devolver un iterador sobre el conjunto de elementos de tipo T. Un iterador es un objeto de una clase que implementa la interfaz Iterator y que obliga a implementar 3 métodos:

public boolean hasNext();

public Object next();

public void remove();

CLASES

En cuanto a lo que respecta a clases, todas son casi imprescindibles de conocer, ya que son básicas y por eso están en el paquete java.lang pero vamos a destacar unas cuantas por considerarlas principales.

PRINCIPALES CLASES EN EL PAQUETE JAVA.LANG	
Boolean	Math
Double	String
Float	StringBuffer y StringBuilder
Integer	System

BOOLEAN

La clase Boolean ya la conocemos. Es lo que se llama un wrap o envoltorio, que es una clase que permite manejar los datos equivalentes de tipo primitivo. En este caso la clase Boolean es un wrap del tipo primitivo boolean. Los métodos de esta clase permiten el manejo de los valores primitivos true o false, su modificación o su comparación ya que implementa la interfaz Comparable.

DOUBLE

Es la clase wrap correspondiente al tipo primitivo double, por lo que lo métodos son muy parecidos a los de la clase Boolean, pero manejando los tipos primitivos para double. Permitiendo obtener, modificar, comparar, etc valores de tipo double.

FLOAT

Al igual que las anteriores también es un wrap pero para el tipo básico o primitivo float.

INTEGER

Esta es quizás la clase de todos los wrappers más utilizada con diferencia y por tanto maneja tipos primitivos de tipo int. Tiene una gran cantidad de métodos sobre todo para poder convertir el entero a otros tipos como long, float, double, etc.

MATH

La clase Math tiene una gran cantidad de métodos para poder hacer operaciones matemáticas, como las funciones sin (double a) que calcula el seno del valor a, tan (double a) que calcula la tangente de a, etc.

STRING

La clase String, quizás hasta más utilizada que la Integer, es una clase que permite la definición y manejo de cadenas de caracteres. Pero un inconveniente posible es que se define como constante y tras su creación no puede ser cambiada (se dice que es inmutable, como explicaremos más adelante).

Por ejemplo si quisiéramos tener una cadena con valores “abc” bastaría con definir una variable de la Clase String de la siguiente manera:

String str = "abc";

Esto automáticamente es correcto aunque quizás fuera más formal lo siguiente:

char data[] = {'a', 'b', 'c'};

String str = new String(data);

La amplia funcionalidad y manejo de String se puede ver en la gran cantidad de métodos que disponemos en esta clase para su manejo, donde podemos hacer comparaciones entre cadenas, comparaciones ignorando mayúsculas (muy útil cuando por ejemplo queremos comparar si la contraseña de un usuario es la correcta), concatenación de cadenas, consulta de carácter en una posición determinada de la cadena, reemplazar caracteres, convertir a mayúsculas, o minúsculas, etc.

CONCEPTO DE INMUTABILIDAD

Veamos el significado de inmutable, que no es precisamente sencillo de explicar. Consideremos que un objeto ocupa un espacio de memoria. Ahora diremos que hay dos tipos de objetos:

a) **Objetos mutables**: son aquellos cuyo espacio de memoria puede ser reemplazado por un contenido definido con posterioridad a la creación del objeto. En este caso, cuando usamos métodos modificadores es el objeto original el que sufre el cambio.

b) **Objetos inmutables**: son aquellos cuyo espacio de memoria permanece ocupado con el objeto tal y como se creó inicialmente. ¿Significa esto que no podemos usar métodos modificadores? No, no significa eso. Significa que cuando usamos métodos modificadores en vez de redefinirse el objeto original, la variable apuntadora pasa a apuntar a un nuevo objeto que se crea, permaneciendo el original en su espacio de memoria, podemos decir que en algunos casos “inaccesible”, pero estando.

En Java, la mayoría de los objetos "esenciales" son inmutables: Long, Integer, Boolean, String, , etc. Un objeto se considera inmutable si su estado no puede cambiarse luego de ser construido. Este comportamiento está ampliamente aceptado como una estrategia robusta para crear código simple y confiable. La inmutabilidad se considera efectiva en aplicaciones concurrentes (hablamos de concurrencia para referirnos a dos tareas que transcurren durante el mismo intervalo de tiempo) donde la mutabilidad (cambiar los objetos su estado), podría dar lugar a que los objetos se corrompieran o llegaran a tener un estado inconsistente.

Nosotros en principio no debemos preocuparnos por el costo de crear nuevos objetos frente a actualizar el mismo objeto. En general usar objetos inmutables es eficiente y solo en casos muy específicos tendríamos que preocuparnos por soluciones que ahorraran la multiplicación de objetos. Aunque no “nos preocupemos” por ella, sí es deseable que tengamos el concepto de inmutabilidad claro.

STRINGBUFFER

Es una versión mejorada o ampliada de la clase String, ya que permite su modificación después de su creación, es decir, los objetos de tipo StringBuffer son objetos mutables. Esto implica que no hay que crear un nuevo objeto cada vez que se redefine una cadena y esto mejora el rendimiento. El método más utilizado de esta clase es insert, que permite insertar cualquier objeto de cualquier tipo como String a la secuencia de StringBuffer.

STRINGBUILDER

Es una clase con características análogas a las de StringBuffer. La diferencia es que StringBuilder puede ofrecer resultados no consistentes en una ejecución multihilo (concurrente, con distintos “subprogramas” ejecutándose simultáneamente) ya que sus métodos no son sincronizados. Sin embargo, cuando no hay necesidades relacionadas con la concurrencia se considera la clase recomendada para operaciones como concatenaciones por tener un mejor rendimiento. Claro está que

para observar ese mejor rendimiento tendremos que referirnos a miles de operaciones con los String. Por el hecho de que un programa haga unas pocas operaciones no vamos a ser capaces de apreciar diferencias de rendimiento.

SYSTEM

Por último la clase System es una de las clases más usuales. Define tres campos err, in y out que respectivamente son la salida estándar de error, entrada estándar y salida estándar. Los métodos para el manejo de estas entradas y salidas estándares de datos son bastante utilizados, al igual que el método *currentTimeMillis()* que devuelve la fecha actual en milisegundos. También es muy usual el uso del método *getenv()* para acceder a variables de entorno o propiedades del sistema. Cuando se invoca este método, se nos devuelve información sobre el sistema con que estamos trabajando.

EXCEPCIONES

Aunque más adelante trataremos el uso y gestión de excepciones, queremos mencionar las más usuales de este paquete. Adelantar que una excepción se produce cuando ocurre un evento, un cálculo, o una ejecución de una instrucción falla produciéndose un problema por el que esa instrucción, cálculo o evento no se puede realizar.

Así las más típicas excepciones pueden ser las siguientes:

ArithmeticException: se produce cuando hay un problema aritmético, como por ejemplo una división por cero.

ArrayIndexOutOfBoundsException: producida cuando queremos acceder a un array con un índice fuera de su rango.

Exception: es la clase principal de todas las excepciones y de ellan cuelgan todas.

NullPointException: normalmente producida cuando intentamos acceder a un objeto que está apuntando a null.

RunTimeException: Es una excepción producida en tiempo de ejecución del programa, por tanto si queremos crear nuestras propias excepciones posiblemente debamos de crear nuestra clase heredando de esta.

En el próximo epígrafe del curso veremos ejemplos de código mostrando lo anteriormente comentado.

EJERCICIO

En esta entrega hemos podido ver lo siguiente: *StringBuilder puede ofrecer resultados no consistentes en una ejecución multihilo (concurrente, con distintos "subprogramas" ejecutándose simultáneamente) ya que sus métodos no son sincronizados.*

Busca información en internet y trata de poner un ejemplo de cómo podría generarse una inconsistencia en programación multihilo. No hace falta que escribas código, únicamente es necesario describir en unas pocas líneas (como si estuvieras explicándoselo a un amigo) la idea o situación.

Para comprobar si tu respuesta es correcta puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU00912C

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=58&Itemid=180