élogos

ARRAYS, LA CLASE OBJECT Y LOS TIPOS GENÉRICOS

élogos

ÍNDICE

ARRAYS, LA CLASE OBJECT Y LOS TIPOS GENÉRICOS

Arrays, la clase Object y los tipos genéricos

1. Arrays

Declaración

Los Arrays son una lista de elementos de cualquier tipo o clase que ocupan posiciones contiguas en la memoria y además se pueden referenciar a través de un índice. Todos los elementos han de ser del mismo tipo.

En la declaración de un Array nunca se indica el número de elementos de que consta:

String Palabras[ ];

int numeros[ ];

String[ ] Palabras;

int[ ] numeros;

Creación de un Array

Hay dos formas de crear un Array:

1. Tipo_Array[ ] NombreArray = new Tipo_Arra[tamaño];

Donde tamaño es el número de elementos del Array.

Aquí en la creación del Array es obligatorio indicar el número de elementos de que consta el Array:

int numeros=new int [10];

2. La segunda forma de crear un Array es creándolo y dándole valores iniciales:

Tipo_Array[ ] NombreArray = {valor1,valor2, ...};

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

Acceso a un array

El acceso para asignarle un valor a un objeto se hace de la siguiente manera:

NombreArray [posición] = valor;

La posición del Array comienza desde la posición 0.

Ejemplo:

```
int numeros[]=new int[3];

numeros [0]=1;

numeros [1]=3

 numeros [2]=6
```

Obtener la longitud del Array

Todos los objetos Array tienen una variable de instancia llamada length.

Ejemplo:

```
int numeros[] = new int[10];

for(int i=0;i<numeros.length;i++){

        numeros[i]=i*2;

}
```

Recorrido con un for each

Desde la versión Java 5 es posible recorrer un array utilizando una variante de la instrucción for conocida como for each. El for each permite recorrer en modo lectura un array sin necesidad de hacer uso de índices.

La sintaxis del for each es la siguiente:

```
for(Tipo variable: array){

        //instrucciones

}
```

Donde *variable* es una variable del tipo de dato del array, que irá tomando cada uno de los valores del mismo en cada iteración.

Por ejemplo, si tenemos un array de objetos String referenciado por la variable “datos” y queremos mostrar todo su contenido en pantalla, escribiríamos el siguiente código:

```
for(String s: datos){

    System.out.println(s);

}
```

2. La clase Object y las conversiones de tipo

La clase Object

La clase Object es la clase raíz de la cual derivan todas las clases. Esta derivación es implícita, por lo que si no se indica nada, todas las clases Java heredarán Object.

Comparación de objetos

Al comparar objetos sólo podemos saber si son o no iguales.

Solo serán iguales si son el mismo objeto, ya que lo que realmente contiene una variable objeto es una dirección de memoria. En esta dirección de memoria guarda el contenido del objeto, de tal modo, que aunque 2 objetos distintos tengan el mismo contenido, al compararlos con el operador == serán diferentes ya que lo que realmente estamos comparando son 2 direcciones de memoria distintas.

Por este motivo, para comprobar la igualdad de dos objetos String utilizaremos el método equals(), en vez del operador ==.

Determinando la clase de un objeto

Para saber de qué clase es un objeto determinado, haremos uso de funciones predefinidas que pertenecen a la clase madre de todas, la clase Object.

Dado el objeto obj haríamos:

```
String nombre = obj.getClass().getName();
```

También podríamos hacer uso del operador instanceof para saber si un objeto pertenece a una clase de la siguiente forma:

```
Objeto instanceof NombreClase;
```

Conversión de tipos de tipos primitivos a objetos

La conversión de tipos primitivos a objetos no se puede hacer directamente pues son dos estructuras distintas, pero Java define clases asociadas a los tipos primitivos básicas y así se soluciona el problema:

clase Integer para los int clase Float para los float

clase Boolean para los boolean

... etc.

Para crear de esta forma un objeto equivalente haríamos uso de la función new ya conocida:

```
ClaseBasica NombreObjeto = new ClaseBasica();
```

```
Integer num = new Integer(2);
```

3. Definición de tipos genéricos. Ventajas

Características de los tipos genéricos

Los tipos genéricos se utilizan principalmente para implementar tipos abstractos de datos como son las pilas, las colas y otros que permiten almacenar distintos tipos de elementos según sean instanciados en tiempo de compilación.

Los tipos genéricos han sido añadidos a Java a partir de la versión 1.5 del jdk

Problemática v/s Ventaja de los tipos genéricos

Revisa el siguiente cuadro comparativo con las problemáticas y ventajas de los tipos genéricos.

Problemática	Ventajas
• El programador debe recordar qué tipo de elemento ha almacenado en la lista y hacer el correspondiente Casting al tipo apropiado cuando lo extraiga de la lista. • No hay comprobación de tipos en tiempo de compilación. • Al no diferenciar los tipos almacenados en tiempo de compilación es necesario una comprobación de los mismos en tiempo de ejecución para poder detectar posibles errores de tipología.	• Comprobación de los tipos almacenados en las listas en tiempo de compilación. • Al no ser necesaria la comprobación de tipos en tiempo de ejecución, implica que haremos menos Castings y por lo tanto más legible el código y será menos propenso a errores. • Los tipos genéricos garantizan que las listas contienen sólo un conjunto de elementos del mismo tipo eliminando los errores derivados de la aparición de listas de diferentes tipos de objetos.

4. Comodines y restricciones de tipos.

Comodines

Si necesitamos que un método cualquiera aunque no sea genérico, admita cualquier objeto de un tipo genérico como parámetro, es posible usar el comodín de tipo o tipo desconocido.

Se representa con el carácter ?, en vez de un tipo concreto. Un ejemplo es el método reverse de la clase

Java.util.Collection

static void reverse(List<?> lista)

Comodín de tipo limitado

El concepto de comodín de tipo lo vamos a usar pero limitándolo a determinadas clases, aquellas que derivan o son superclases de una clase dada.

Ejemplo:

```
public static double sumaLaLista(List<? extends Number>listaDeNumeros){
        double total = 0.0;
        for (int i= 0; i < listaDeNumeros.size(); i++){
                total += listaDeNumeros.get(i).doubleValue();
        }
        return total;
}
```

5. Métodos genéricos

Los métodos genéricos son aquellos que tienen declaradas sus propias variables de tipo.

Un ejemplo son los métodos estáticos, que por su propia definición, no pueden acceder a las variables de tipo declaradas en su propia clase, pero sí que podrían tener sus propias variables de tipo.

En los método genéricos las variables se declaran como hasta ahora, entre los símbolos < >(menor-mayor) y separadas por comas.

Se sitúan entre los modificadores del método y el tipo de retorno. Esta sería su declaración:

Modificador_Metodo <A,B,..> Tipo_Retorno nombre_Metodo(arg1,arg2,..)

Los métodos genéricos se podrían utilizar para limitar los tipos pasados por parámetro del tipo genérico:

```
public static <N extends Number> double sumaPila(Pila<N> p){
double total= 0;
for (int i = 0; i < p.pila.size(); i++){
```

```
            total+= p.pila.get(i).doubleValue();

           }

return total;

}
```