



J2SE 5

Novedades

Introducción

- Publicada en agosto de 2004.
- Lo más destacado:
 - Nuevas funcionalidades del lenguaje.
- Compatible con versiones anteriores.

Métodos con lista variable de argumentos

- Variación de la sobrecarga: métodos que aceptan cualquier número de argumentos del mismo tipo:

```
public class Statistics {  
    public float average(int x1, int x2){}  
    public float average(int x1, int x2, int x3){}  
    public float average(int x1, int x2, int x3, int x4){}  
}
```

- Invocación:

```
float res1 = stats.average(4, 3, 4);  
float res2 = stats.average(24, 32, 27, 18);
```

Varargs: argumentos variables

- Estos métodos se pueden agrupar:

```
public class Statistics {  
    public float average(int... nums){  
        int sum = 0;  
  
        //nums es un int[]  
        for (int x : nums) {  
            sum += x;  
        }  
        return ((float) sum) / nums.length;  
    }  
}
```

Tipos enumerados

- Son conjuntos finitos de símbolos que representan los valores posibles de un atributo.
 - Ej: palos={"OROS", "COPAS", "ESPADAS", "BASTOS"}.
- Nota: no confundir con la clase `java.util.Enumeration`.

Tipos enumerados

- Antes de Java 5:

```
package cards.domain;
```

```
public class PlayingCard {  
    //pseudo enumerado  
    public static final int SUIT_SPADES = 0;  
    public static final int SUIT_HEARTS = 1;  
    public static final int SUIT_CLUBS = 2;  
    public static final int SUIT_DIAMONDS = 3;  
  
    private int suit;  
    private int rank;  
  
    public PlayingCard(int suit, int rank) {  
        this.suit = suit; this.rank = rank;  
    }  
}
```

Tipos enumerados

```
public String getSuitName() {  
    String name = "";  
    switch(suit) {  
        case SUIT_SPADES:  
            name = "Spades";  
            break;  
        case SUIT_HEARTS:  
            name = "Hearts";  
            break;  
        case SUIT_CLUBS:  
            name = "Clubs";  
            break;  
        case SUIT_DIAMONDS:  
            name = "Diamonds";  
            break;  
        default:  
            System.err.println("Invalid suit");  
    }  
    return name;  
}
```

Tipos enumerados

- Problemas:
 - No es “type-safe”.
 - Como `suit` es entero, se puede pasar cualquier otro valor sin significado.
 - Alguien podría hacer operaciones aritméticas con `suit`, lo cual no tiene sentido.
 - No hay espacio de nombres.
 - Las constantes se deben prefijar con `SUIT_` para evitar colisiones con otras constantes.
 - Comportamiento frágil.
 - Si se añaden constantes o se modifica el orden, hay que recompilar los clientes para que se actualicen con los nuevos valores.

Tipos enumerados

- Problemas:
 - Valores no informativos.
 - Son enteros. Si se imprimen, aparece un número. Es necesario escribir un método que ofrezca una representación textual legible.

Tipos enumerados

- A partir de Java 5:

```
package cards.domain;
```

```
public class Suit{
```

```
    public enum SuitValue {SPADES, HEARTS, CLUBS, DIAMONDS}
```

```
}
```

Tipos enumerados

```
package cards.domain;

public class PlayingCard {

    private Suit.SuitValue suit;
    private int rank;

    public PlayingCard(Suit.SuitValue suit,
int rank) {
        this.suit = suit;
        this.rank = rank;
    }
}
```

Tipos enumerados

```
public String getSuitName() {  
    String name = "";  
    switch(suit) {  
        case SPADES:  
            name = "Spades";  
            break;  
        case HEARTS:  
            name = "Hearts";  
            break;  
        case CLUBS:  
            name = "Clubs";  
            break;  
        case DIAMONDS:  
            name = "Diamonds";  
            break;  
        // No hace falta comprobar tipos  
        // erróneos: Suit es un conjunto  
        // finito  
    }  
    return name;  
}
```

Tipos enumerados avanzados

```
package cards.domain;

public class PlayingCards{

    public enum Suit {
        SPADES      ("Spades"),
        HEARTS       ("Hearts"),
        CLUBS        ("Clubs"),
        DIAMONDS     ("Diamonds");

        private final String name;

        private Suit(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }
    }
}
```

Autoboxing de tipos primitivos

- En J2SE 5, el boxing y el unboxing se hacen automáticamente (autoboxing).

```
int pInt = 500;  
Integer wInt = pInt; //boxing  
int p2 = wInt; //unboxing
```

- No se debe abusar de esta funcionalidad para evitar problemas de rendimiento.

Autoboxing de tipos primitivos

- ¡Cuidado!

```
Integer iObjeto1 = 1000;  
Integer iObjeto2 = 1000;
```

```
int i1= 1000;  
int i2= 1000;
```

```
(i2 == i1) // resultado = ?
```

```
(i2 == iObjeto1) // resultado = ?
```

```
(iObjeto2 == iObjeto1) // resultado = ?
```

```
Integer i= null;  
int i2= i;
```

¿Qué ocurre aquí?

Static import

- Para acceder a los miembros estáticos de una clase, hay que:
 - O bien proporcionar su nombre plenamente cualificado.
 - O bien importar la clase.
- En Java 5 existe la posibilidad de importar todos los miembros estáticos de una clase o incluso los valores de un enumerado.

Static import

- **Antes:**

```
public class Maths {  
  
    public static void main(String[] args) {  
        double r = java.lang.Math.cos(2*java.lang.Math.PI);  
        System.out.println(r);  
    }  
}
```

- **Ahora:**

```
import static java.lang.Math.*;  
  
public class Maths {  
    public static void main(String[] args) {  
        double r = cos(2*PI);  
        System.out.println(r);  
    }  
}
```

Genéricos

- En el API Collections tradicional, las colecciones eran colecciones de Object. Esto obligaba al programador a hacer un casting, lo cual no es “type-safe”:

```
ArrayList list = new ArrayList();  
list.add(0, new Integer(42));  
int total = ((Integer)list.get(0)).intValue();  
  
//Lo siguiente compila, pero falla al ejecutar  
String total = (String)list.get(0);
```

Genéricos

- En Java 5, las colecciones pueden ser definidas de un tipo concreto, lo cual aporta dos ventajas:
 - Evita la necesidad de hacer casting.
 - Permite la comprobación de errores en tiempo de compilación.

```
ArrayList<Integer> list =  
new ArrayList<Integer>();  
list.add(0, new Integer(42));
```

```
//Genéricos & autoboxing  
int total = list.get(0);
```

Warnings del compilador

- El compilador de Java 5 lanza warnings sobre el código 1.4 sin uso de genéricos

```
public class GenericsWarning {  
    public static void main(String[] args) {  
        List list = new ArrayList();  
        list.add(0, new Integer(42));  
        int total = ((Integer)list.get(0)).intValue();  
    }  
}
```

javac GenericWarnings.java

Note: GenericWarnings.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

Warnings del compilador

```
javac -Xlint:unchecked GenericWarnings.java
```

```
GenericWarnings.java:5: warning: [unchecked]  
unchecked call to add(int,E) as a member of the  
raw type java.util.List
```

```
list.add(0, new Integer(42));  
                ^
```

Nuevo bucle for

- Con iteradores (Java 1.4):

```
public void printAll(Collection c) {  
    Iterator it = c.iterator();  
    while (it.hasNext()) {  
        String element =(String)it.next();  
        System.out.println(element);  
    }  
}
```

- Con iteradores y genéricos:

```
public void printAll(Collection<String> c) {  
    for (Iterator<String> i=c.iterator(); i.hasNext(); ){  
        String element = i.next();  
        System.out.println(element);  
    }  
}
```

Nuevo bucle for

- Con el nuevo bucle for:

```
public void printAll(Collection<String> c) {  
    for (String s : c) {  
        System.out.println(s);  
    }  
}
```

- Debe ser leído como:

— “Para cada elemento 's' de la colección 'c',
hacer...”

Salida formateada

- Java 5 introduce en `System.out` un método `printf()` que funciona como en C/C++. Esta funcionalidad también está en el método `format()` de la clase `String`.

```
System.out.printf("name count\n");  
String s = String.format("%s %d\n", user, total);
```

Código	Descripción
%s	Formatea el argumento como una cadena, usualmente llamando al <code>toString()</code> de dicho objeto.
%d %o %x	Formatea un entero como decimal, octal o hexadecimal.
%f %g	Formatea un número en coma flotante. El formato %g usa notación científica.
%n	Inserta un salto de línea
%%	Inserta el carácter %

Entrada formateada

- La clase `java.util.Scanner` proporciona entrada del usuario formateada.
 - Obtiene datos en un tipo adecuado y se bloquea esperando la entrada del usuario.

```
public class ScannerTest {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        String param1 = s.next();  
        System.out.println("Param 1: " + param1);  
        int param2 = s.nextInt();  
        System.out.println("Param 2: " + param2);  
        s.close();  
    }  
}
```