C++ vs. Java

Dr. J.B. Hayet

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS

Octubre 2009

Java...

- Desarrollado a partir de la nada por la empresa Sun en los 90s.
- Por su concepción, está a más alto nivel que C++ o C, eliminando muchas de las caracteristicas de esos lenguajes de bajo nivel (manejo directo de la memoria...).
- Aplicaciones compiladas en bytecode, con la idea de que sirva de "código objeto" para una máquina virtual, que es la que si cambia en función de la plataforma, "Write once, run everywhere".
- Como todo es finalmente interpretado por la JVM, tiene generalmente la reputación de ser menos eficiente.

Java...

- En 2006-2007, Sun liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre.
- Ultima *release* estable: Java Standard Edition 6.

Java...

Motivación:

1. Programación orientada a objetos (POO).
2. Portabilidad y ejecución de un mismo programa en múltiples máquinas (independientemente de la plataforma) .
3. Fuerte soporte para la red.
4. Ejecución de código en sistemas remotos de forma segura.
5. Facilidad de uso.

Pasar de C++ a Java...

Antes de todo:

- si ya entendieron bien los conceptos OO con C++ y los elementos sintácticos de C++, les resultará muy fácil pasar a Java, ya que son los mismos,
- en la mayoría de los aspectos, simplifica aún la programación.

Programas Java

- Un programa en JAVA es formado de clases, solamente.
- Cada clase compilada resulta en un archivo de bytecode .CLASS. Al menos una clase tiene que tener un método MAIN() con el cual el programa arranca.
- Para correr el programa, se invoca como ejecutable el interpretador java seguido por el nombre de la clase que contiene el main (sin el .class).

Programas Java

El interpretador corre hasta que se salga el MAIN(), que tiene que tener el prototipo siguiente:

```
public static void main(String argv[])
```

- en realidad se parará cuando todos los hilos creados en el MAIN() se harán acabado.
- el argv se parece al de C/C++ excepto que el primer elemento no es el nombre del programa...

Programas Java

```
public class Tata {
    public static void main(String argv[]) {
        for(int i=0; i < argv.length; i++)
            System.out.print(argv[i] + " ");
        System.out.print("\n");
        System.exit(0);
    }
}
```

Programas Java

Una diferencia notable:

- MAIN() tiene que tener VOID de valor de regreso.
- Para salir con un valor de status, hay que usar SYSTEM.EXIT(N), pero eso es violento: sale de la JVM, incluso si varios hilos van corriendo, y la manera de que este valor *n* es procesado depende del OS.

Programas Java

- La idea del lenguaje es en particular se soportar cargamiendo de modulos hasta por la red: hay interés particularmente importante a evitar todo conflicto de nombre.
- No variables globales: por la misma razón, toda variable está definida dentro de una clase.

Los nombres enteramente calificados, de métodos por ejemplo son de tipo:

`yomero.misaplicaciones.sudoku.resolve`

(notar el punto en lugar de ::)

Paquetes Java

El equivalente (grueso) de los namespace de C++ son los packages.

Definir código para un *package* (namespace), al principio de un archivo de código

```
package yomero.misaplicaciones;
```

Las clases que escribes tendrán como nombre completo:

```
yomero.misaplicaciones.toto
```

Paquetes Java: import

Usar un paquete por su ultima palabra (using namespace Y includes)

```
import yomero.misaplicaciones;
...
misaplicaciones.sudoku a = .. ;
```

Usar todo el contenido de un package (using namespace Y includes)

```
import yomero.misaplicaciones.*;
```

Usar nada más una clase

```
import yomero.misaplicaciones.sudoku;
...
sudoku a = ...;
```

Paquetes de la API Java

java.lang	Lo esencial (cargado por default)
java.util	Estructuras de datos, algoritmos
java.io	Archivos, IO
java.math	Aritmética multi-precisión
java.net	Paquetes de redes ...
java.security	Encripción. . .
java.sql	Bases de datos
java.awt	Paquetes para construir una GUI simple
java.awt.image	Procesamiento de imágenes
javax.swing	Paquetes para construir una GUI máas elaborada
java.applet	Clases para crear applets

Programas Java

Código compilado

- Todas las clases compiladas están en su propio archivo (.class).
- El código compilado está en un .class de mismo nombre.
- El nombre tiene que corresponder al nombre de la clase y además, si está parte de un paquete, el camino donde se encuentra el .class tiene que corresponder al nombre completo del paquete.

`yomero.misaplicaciones.sudoku`

tiene que estar ubicada (relativamente al CLASSPATH) en "yomero/misaplicaciones/sudoku.class"

Programas Java

Código fuente

- No hay .h o .cpp, nada más el .java.
- Para clases públicas (visibles fuera del paquete), el .java tiene que llamarse igual y no puede haber más clases públicas adentro. Para las otras, se genera .class diferentes.

Programas Java

¿Como funciona sin los includes ?

El interpretador busca

- en directorios de la instalación de JAVA,
- en los directorios especificados en linea de comando (-classpath),
- en los directorios apareciendo en la variable de entorno CLASSPATH.

En Unix con tcsh

```
setenv CLASSPATH .:~/toto/java:/usr/local/misClases/
```

Programas Java

Control de acceso

- los archivos de los paquetes requeridos tienen que estar accesibles (derechos, accesibles por Internet...),
- todas las clases/interfaces definidas en un paquete son visibles dentro de todo el código del paquete, con cualquiera palabra llave de acceso,
- las clases declaradas como public son accesibles a otros paquetes, las otras no son accesibles fuera del paquete en que van definidas,
- dentro de las clases, variables o métodos tienen control de acceso: public por default, private o (private) protected (muy similar a C++).

Programas Java

Comentarios

- comentarios “a la C” (no se imbrican)

```
/* No me digaaaas */
```

- comentarios “a la C++”

```
// No me digaaaas
```

- comentarios “a la doxygen/javadoc”

```
/**
 * No me digaaaas
 * @see    Chismes
 */
```

Programas Java: bye, preprocesor

No hay ningún preprocesador, consecuencias:

- No macros.
- No includes.
- En lugar de #if 0 para comentarios usar /* */ (que pueden tener // adentro).
- Para constantes numéricas, hay que hacer a la C++, i.e. con constantes estáticas, a definir dentro de clases

```
public final class Math {
    ...
    public static final double PI = 3.14159....;
    ...
}
```

Programas Java

- Convención de Java: mayúsculas para constantes,
- como en C++, el uso de las constantes (que tienen un nombre calificado muy específico) reducen el riesgo de colisiones,
- naturalmente, no hay macros pero el compilador se ocupa automáticamente de hacer inline a las funciones para las cuales se puede,
- no se necesitan los includes: (1) hay un mapeo implícito entre el nombre calificado de las clases y los archivos y (2) no se necesita tener todas las declaraciones antes (el orden de las definiciones no importa, no hay distinción entre declaración y definición).

Programas Java

Ya que no hay pre-procesador no hay manera directa de hacer compilación condicional ("#if 0 ... #endif")...
Pero:

- en general no sirve mucho la compilación condicional (código independiente de la plataforma...)
- en general, el código que el compilador ve que no va a usarse no se compila,
- entonces, pasando por una constante

```
private static final boolean DEBUG = false;
```

el código en un bloque "if (DEBUG) {" no va a ser compilado.

Tipos en Java...

Las grandes diferencias con C

- hay un tipo "byte" y un tipo boolean (que no es entero),
- los tamaños no dependen de la máquina o del OS, son definidos en bits,
- un tipo tiene nada más una signedness,
- todos las variables de algún tipo básico están inicializadas con un valor por default.

Tipos en Java...

Tipo	Contenido	Default	Tamaño	Valor min/max
boolean	true or false	false	1 bit	N.A./N.A.
char	Car. Unicode	0x0000	16 bits	0x0000/0xFFFF
byte	signed integer	0	8 bits	-128/127
short	signed integer	0	16 bits	-32768/32767
int	signed integer	0	32 bits	-2147483648/2147483647
long	signed integer	0	64 bits	-9223372036854775808/ 9223372036854775807
float	IEEE 754 floating-point	0.0	32 bits	+-3.40282347E+38/ +-1.40239846E-45
double	IEEE 754 floating-point	0.0	64 bits	+-1.79769E+308 +-4.94066E-324

Tipos en Java...

El tipo booleano no es tipo "entero", no se puede hacer cast desde booleano o hacia booleano. La única forma de hacer:

```
b = (i != 0);
// integer-to-boolean: non-0 -> true; 0 -> false;
i = (b)?1:0;
// boolean-to-integer: true -> 1; false -> 0;
```

Tipos en Java...

- No hay "UNSIGNED": todos los tipos otros que char son signados.
- No se puede escribir LONG INT o SHORT INT.
- Una constante numérica se puede especificar como "LONG" igual que en C, agregando la letra "L" o "l".
- Una división por cero o un modulo cero provoca una excepción ARITHMETICEXCEPTION.

Tipos en Java...

- Flotantes indicados por el sufijo f o F (d o D para doubles).
- Valores especiales: $+\infty$, $-\infty$, 0^-, 0^+, *NaN* accesibles por las clases java.lang.Float and java.lang.Double como constantes: POSITIVE_INFINITY, NEGATIVE_INFINITY, NaN.
- NaN no se puede usar para comparación directa: Float.isNaN() or Double.isNaN().
- 0^- y 0^+ se comparan igual, se distinguen por el resultado de la división por 0 ($\pm\infty$).
- Aritmética con punto flotante nunca causa excepciones.

Elemento sintácticos ausentes en Java...

- No hay STRUCT o UNION, solo CLASS.
- No hay ENUM (usar valores constantes STATIC FINAL).
- No hay bitfields (uso de unos tantos bits para elementos de clases/struct...).
- No hay TYPEDEF.
- No hay equivalente de lo de funciones con lista de argumentos variable.

Palabras llaves propias a Java

FINAL

Aplica para clases que no se puede derivar, métodos que no se puede sobrecargar, variables a las cuales no se pueden cambiar el valor (similar a const) o referencias a las cuales no se puede cambiar el objeto referenciado.

NATIVE

Aplica para métodos que en realidad son implementadas en algún otro lugar en C o C++. En este caso se pone nada más la "declaración" de la función.

Palabras llaves propias a Java

SYNCHRONIZED

Permite asegurar que no varios hilos aceden a un objeto/valor para ejecutar un bloque dado o un método de un objeto.

TRANSIENT

Utilizado para variables de una clase que no hacen parte del estado persistente. No estará serializado (e.g., grabado en un archivo).

VOLATILE

Garantiza que una variable no está puesta en cache en un registro por algún thread.

Variables Java

- A parte de los tipos primitivos (usados por valor), en Java sólo hay tipos-referencia, es decir objetos y arreglos que sólo se manipulan por referencia, de la misma manera que en C++, excepto que todo está implícito.
- No hay apuntador, no hay operadores *, $->$, & ...
- Los tipos primitivos son siempre pasados por valor, los objetos y arreglos siempre por referencia.

Variables Java

Los objetos-referencias siendo creados por el operador NEW, tenemos por una parte

```
Toto x1, x2;
x1 = new Toto();
x2 = x1;
x1.setUnaCosa(1);
int cosa = x2.getUnaCosa();
```

pero:

```
int x1 = 10;
int x2 = x1;
x1 = -1;
```

Variables Java

Ya que los objetos se manipulan por referencia, la asignación entre objetos no hace nada similar a la de C++

```
Toto x1 = new Toto("Tutu");
Toto x2 = new Toto("Tata");
x1 = x2;
```

El objeto inicialmente apuntado por x1 está perdido (pero liberado...). Tipicamente, para copiar, los objetos tienen un método CLONE()

```
List list = new ArrayList();
list.add("A");
List list2 = ((List) ((ArrayList) list).clone());
```

Las clases que tienen el método clone() implementan la interfaz Cloneable (de java.lang). Object implementa esta interfaz.

Variables Java

De la misma manera, los arreglos también son referencias,

```
Toto[] x1;
Toto[] x2;
x1 = new Toto[10];
... // Hay que inicializar cada elemento del arreglo
x2 = x1;
```

Variables Java

Copiar el contenido de un arreglo:

- Copiar los elementos uno a uno...
- Usar System.arraycopy()...

```
public static void arraycopy(Object src,
                             int srcPos,
                             Object dest,
                             int destPos,
                             int length)
```

Variables Java

- De la misma manera el operador == no compara los objetos sino las referencias.
- Para comparar objetos, se usa un método EQUALS() definido en OBJECT pero que es mejor re-implementar en tus clases derivadas.

Variables Java

No apuntadores significa en particular:

- de-referenciación automática manipulando referencias,
- no cast de referencias (arreglos o objetos) a enteros,
- no aritmética de apuntadores,
- no manera de manipular el tamaño de los tipos.

Todo eso para desconectar el programador de la manipulación de la memoria (seguridad, simplicidad, menos bugs. . .).

Variables Java

- El valor por default de las referencias es “null”. Indica explicitamente la ausencia de referenciado.
- No es como el NULL de C, es una palabra llave entera.
- No se puede hacer cast de este null a cualquier otro tipo, incluso enteros o booleanos, en particular el test tiene que hacerse explícito.

Variables Java

- Los objetos sólo se pueden crear en el montículo (operador new). Se manipula la referencia al objeto creado de esta manera.
- Como en C++, el operador new permite invocar un constructor.

```
Toto t = new Toto();
Complex c = new Complex(-2.0, 3.5);
```

Variables Java

En el caso de String, hay una manera más de inicializar el objeto:

```
String s = "Otra manera de hacer";
```

O sea, a la C...

Existe también un método newInstance()

```
Object t = tutu.newInstance();
```

Variables Java

El acceso es lo mismo que en C++ con referencias,

```
Toto t = new Toto();
t.a = 2.0;
t.b = -1.4;
double val = t.getVal();
```

Variables Java

- La gran diferencia con C es que no hay liberación de memoria que hacer: el GC (garbage collector) se encarga de liberar la memoria alocada, de una manera similar a lo que hemos visto en el cuenteo de referencias.
- Simplifica la vida, tener fugas de memoria queda posible pero es más difícil.
- Eso vale en particular para arreglos: están creados por new, manipulados por referencias y liberados automáticamente.

Variables Java

Crear arreglos:

```
byte buffer[] = new byte[1024];
Toto ts[] = new Toto[10];
```

El arreglo no crea los Totos ! es algo que se tiene que hacer a parte. Para los tipos básicos, los valores sí están inicializadas a 0 (enteros o flotantes). Para los tipos referencias (objetos o otros arreglos) están incializados a null.

Variables Java

Inicialización "a la C":

```
int misDatos[] = {2,3,1,2,19,7,6};
```

La diferencia con C es que se puede poner cualquier expresión en la lista de inicializadores. De la misma manera que con cualquier otro objeto, los arreglos son liberados automáticamente.

Variables Java

Arreglos multidimensionales, de la misma manera que “C”

```
int misDatos[][] = new int[256][256];
```

- creación de una variable misDatos,
- creación de un arreglo de 256 arreglos, asignado a misDatos,
- para cada uno de los 256 arreglos, creación de un arreglo de 256 enteros, asignados a los misDatos[i], y inicializados con 0.

Variables Java

Puedes hacer alocación parcial haciendo arreglos de referencias a arreglos no alocados,

```
int arreglo[][][] = new int[5][][]; // 5 arreglos dob

int arreglo[][][] = new int[5][3][]; // 5x3 arreglos

int arreglo[][][] = new int[5][][3]; // UH ???
```

Variables Java

Inicialización “a la C”

```
String[][] nombres = {{"Pepito", "Alberto", "Juan"},
                      {"Pérez", "Gómez"}};
```

Los arreglos multi-dimensionales no son necesariamente rectangulares:

```
int triangle[][] = new int[10][];
for(int i = 0; i < triangle.length; i++) {
    triangle[i] = new int[i+1];
for(int j=0; j < i+1; j++)
    triangle[i][j] = i + j;
}

static int[][] triangle
  = {{1, 2}, {3, 4, 5}, {5, 6, 7, 8}};
```

Variables Java

- Acceso por corchetes.
- El tamaño de un arreglo por un elemento especial del arreglo, LENGTH (elemento solo accesible en lectura).
- El acceso es seguro: el índex es verificado contra el rango de indices admisibles. En el caso de que sale del rango, se lanza una excepción ARRAYINDEXOUTOFBOUNDSEXCEPTION.

Variables Java

Declaración, en C

```
char buffer[500];
```

Declaración, en JAVA

```
char buffer[];
buffer = new char[500];
```

También está permitido (¿más intuitivo ?)

```
char[] buffer = new char[500];
```

Puedes hasta hacer mezclas

```
int[] row, column, matrix[];

public void mult(byte[] matrix[],int[] column) { ...
```

Variables Java

- Manejadas a la C++, en gran parte.
- Constructores a partir de cadenas de caracteres.
- No se pueden cambiar directamente, sólo pasando por un StringBuffer.
- Métodos

```
length(), charAt(), equals(), compareTo(),
indexOf(), lastIndexOf(), substring()
```

Operadores

- Casi todos los mismos que en C,C++, con las mismas precedencias e asociatividad.
- No operador coma.
- No operadores ligados a apuntadore (*,− >,&).
- No SIZEOF().
- No operadores sobrecargados.

Variables Java

Nuevos operadores con respeto a C:
+

Concatenación, en el caso de manipulación de String. Uno de los operandos tiene que ser String, el otro operando tiene que ser convertible a String. De la misma manera, un operador +=

INSTANCEOF

Operador que checa si un objeto pasado como primer operando es del tipo de la clase pasada de segundo operando (o implementa una interfaz dada).

```
if (tutu instanceof Toto)
    return "El objeto tutu es de tipo Toto !!"
```

Variables Java

>>>

El operador >> hace un shift completando las casillas con el bit de signo. El operador >>> más bien completa todo con 0.

& Y |

Para los tipos enteros, igual que en C. Ahora para los boolean, hacen el AND y OR lógicos que evalúen los dos operandos. Para la versión que no evalua necesariamente los dos operandos (si el resultado ya es conocido) se puede usar && y ||.

Ciclos de control

Los if/else, while, y do/while son exactamente como los de C. La única diferencia es que la condición es un booleano y no puede estar convertido directamente en otro tipo; todos los ciclos condicionales hacen el test con un booleano. Entonces no se puede hacer cosas como:

```
int i = 10;
while(i--) { // NO
 Toto t = recupera();
  if (t) { // NO
    do { j=algo(); } while(j); // NO
  }
}
```

Ciclos de control

Entonces, se tiene que usar las condiciones escritas explícitamente

```
int i = 10;
while(i-->0) { // SI
 Toto t = recupera();
  if (t!=null) { // SI
    do { j=algo(); } while(j != 0); // SI
  }
}
```

Ciclos de control

Los ciclos SWITCH y FOR son también presentes. El FOR tiene una diferencia: no hay operador coma, entonces el lenguaje simula la concatenación de instrucciones, pero sólo por la inicialización y la terminación

```
int i;
int k;
for(i=0, k=10;(i < 10) && (k >= 1); i++, k--) {
    System.out.println(i*k);
}
```

Además, como en C++, se puede declarar las variables directamente dentro del for.

Ciclos de control

No GOTO. Las palabras BREAK y CONTINUE se usan igual que en C, pero también se pueden usar especificando un label para salir en un bloque dado entre los bloques en que está anidado el CONTINUE o el BREAK. Sin label y como en C, sale al nivel del ciclo inmediatamente anidador

```
search:
    for (i = 0; i < arrayOfInts.length; i++) {
        for (j = 0; j < arrayOfInts[i].length; j++) {
            if (arrayOfInts[i][j] == searchfor) {
                foundIt = true;
                break search;
            }
        }
    }
```

Ciclos de control

Igual con CONTINUE:

```
test:
    for (int i = 0; i <= max; i++) {
        int n = substring.length();
        int j = i;
        int k = 0;
        while (n-- != 0) {
            if (searchMe.charAt(j++)
                    != substring.charAt(k++)) {
                continue test;
            }
        }
        foundIt = true;
        break test;
    }
```

Ciclos de control

Una construcción nueva es la que se puede formar con SYNCHRONIZED. Es una construcción que integra la particularidad de JAVA de tener hilos (threads, procesos simplificados, corriendo en "paralelo") en nativo:

```
synchronized (tutu) {
        tutu.modifyHeavily();
}
```

al nivel del SYNCHRONIZED, se intenta obtener un lock (es decir una exclusividad del acceso a tutu), y se entra en el bloque que sigue sólo cuando se obtuvo esa exclusividad. También se puede usar como modificador de métodos con el mismo efecto (el método no se ejecuta hasta que el objeto que le llama se pueda bloquear).

Excepciones

- Funcionamiento igual que en C++: un objeto está lanzado y atrapado en algún bloque anidador.
- Los objetos-excepciones son del tipo JAVA.LANG.THROWABLE, que tiene dos clases derivadas JAVA.LANG.ERROR y JAVA.LANG.EXCEPTION.
- Los ERROR corresponden a situaciones críticas en que hay que salir, sin procesar la excepción; las EXCEPTION sí tienen que estar procesadas.
- Como en C++, los JAVA.LANG.THROWABLE incluyen una STRING destinada a documentar la excepción.

```
Throwable.getMessage()
```

Excepciones

Bloques muy similares a C++, con una palabra adicional:

```
try {
        // An exception may be thrown here
}
catch (ExceptionType1 e1) {
        // Handle ExceptionType1
}
catch (ExceptionType1 e2) {
        // Handle ExceptionType2
}
finally {
        // Executed in all these cases to exit proper
        //    1) En el caso normal.
        //    2) En el caso de excepcion interceptada
        //    3) En el caso de excepcion no interce
        //    4) En el caso de break, continue, o
```

Excepciones

- Se puede jugar con diferentes niveles de la pila como en C++.
- El FINALLY se ejecuta en todos los casos, permite asegurar de salir propiamente en todos los casos.
- Se ejecuta incluso si se sale del TRY con un RETURN.

Excepciones

De la misma manera que en C++, las excepciones se pueden declarar

```
public void openFile() throws IOException {
        // May generate java.io.IOException
}

public void fonc() throws MyException1, MyException2
    ...
}
```

Excepciones

- Se verifica en el momento de la compilación que un método tiene que especificar o no excepciones (diferente de C++). Habrá error de compilación si olvidas especificar unas.
- Las excepciones se tiene que crear como todos los Object, es decir

```
throw new MiBonitaExcepcion("Aie aie aie.");
```

Comparaciones diversas

- Declaraciones de variables locales: en cualquier lugar, hasta en los ciclos de control, como en C++.
- Supresión de la necesidad de tener declaración/definición de los variables/funciones antes de su uso.
- Sobrecargar es posible igual que en C++.
- No se puede definir un método que tome void de argumento.

Clases

El mecanismo OO es muy similar al de C++, excepto que:

- hay una sola jerarquía, todas las clases derivan de una clase-madre que se llama Object,
- no hay herencia múltiple,
- el *late binding* es la regla para todas las clases,
- existe explícitamente la noción de interfaz.

Clases

La clase Object es la clase raíz de todo el árbol de la jerarquía de clases Java; tiene métodos:

- con el que un objeto se puede comparar con otro,
- para convertir el objeto a una String,
- para esperar a que esté satisfecha alguna condición,
- para notificar a otros objetos que una condición ha cambiado,
- para devolver la clase del objeto.

Clases

- Constructores similares a C++. Mismas reglas de existencia de un constructor por default que en C++ (si no defines, él define uno; si no, él no define ninguno)
- No constructores por copia.
- No destructores (GC) pero puede ser necesario hacer unas limpiezas. Para eso existe un método FINALIZE() llamado por el GC.
- Importante llamar a

```
super.finalize();
```

Clases

Recientemente aparecieron patrones.

- Los contenedores proveídos por Java son numerosos. Son definidos como contenedores de referencias a derivados de cierta clase: Vector, Stack, Hashtable...
- Las interfaces básicos de esos contenedores son Collection y Map.
- También vienen iteradores.

Clases

JAVA tiene soporte nativo para el multi-thread y eso tiene consecuencia para todos los objetos:

- hay objetos THREAD,
- se puede requerir sincronización del uso de un objeto en particular, en un método en particular: acá interviene la palabra llave SYNCHRONIZED,
- los locks en C o C++ son explícitos y mas pesados (mutex...).

Clases

No hay la lista de inicializadores del C++ pero:

```
public class Toto extends Tutu
{
    public Toto(String msg) {
        super(msg); // Calls base constructor
    }

    public yells(int i) { // Override
        super.yells(i); // Calls base method
    }
}
```

Clases

- La herencia está expresada por la palabra llave extends (hace clara la herencia).
- No hay palabras llave de acceso en la herencia.
- Además de la herencia, se puede especificar que una clase realiza cierta interfaz; en C++ eso corresponde a una herencia de una clase abstracta, aquí es más explícito.

Clases

- Las interfaces se escriben igual que las clases de Java (excepto la palabra llave interface en lugar de class), y resultan iguales al uso que las clases abstractas (no se pueden implementar).
- Una clase implementando una interfaz utiliza la palabra llave implements.
- Hay también una palabra llave abstract para clases, la diferencia es que pueden tener implementación, parcialmente.

```
public interface Tata {
    public void snores();
}

public class Toto extends Tutu implements Tata {
    public void snores( ) {
        System.out.println("rrrrrh...");
    }
```

Clases

- No hay palabra llave virtual porque el mecanismo es de late binding por default !
- Palabra llave final para especificar que no más versiones sobrecargadas de un método pueden existir en clases derivadas.

Clases

Runtime identification

```
X.getClass().getName();
```

El downcast no es explícito como a la C++ pero sí hace las verificaciones (y envia excepciones si necesario)

```
derived d = (derived)base;
```