SINTAXIS DEL LENGUAJE JAVA

ADOLFO SANZ DE DIEGO

JUNIO DE 2011

1 TIPOS DE DATOS

- Java no es orientado a objetos puro, tiene **tipos primitivos**.
- Para cada tipo primitivo hay una clase envoltorio (**wrapper**) que le añade funcionalidad, entre ellas la posibilidad de convertir cadenas al tipo primitivo y viceversa.
- Las clases envoltorio se llaman igual que el tipo primitivo, empezando por mayúscula en vez de minúscula.
- Hay 2 excepciones a la regla anterior: el wrapper de int es Integer y el de char es Character.

2 TIPOS PRIMITIVOS

Tipo	Nombre	Rango de valores	Wrapp
Enteros	byte	[-128 a 127]	Byte
	short	[-32768 a 32767]	Short
	int	[-2147483648 a 2147483647]	Integer
	long	[-9223372036854775808 a 9223372036854775807]	Long

Tipo	Nombre	Rango de valores	Wrapp
Coma Flotante	float	[32 bits, precisión simple; 3,4E-38 a 3,4E38]	Float
	double	[64 bits, precisión doble; 1,7E-308 a 1,7E308]	Double
Booleano	boolean	[true o false]	Boolea
Carácter	char	[carácter alfanumérico]	Charac

3 MATRICES

- En Java se pueden crear **matrices** multidimensionales.
- Pero por lo general se usan las **estructuras de datos** de Java que veremos más adelante.

4 CADENAS

- Las cadenas se representan con la clase **String**.
- Es una clase envoltorio de una matriz de chars.
- Es una clase especial:
  - permite "" como constructor.
  - permite el operador + para concatenar.

5 OPERADORES

	Aritméticos		**Lógicos**		**Condicional**
+	[adición]	>	[mayor que]	&&	[ambos ciertos]
-	[sustracción]	>=	[mayor o igual que]	||	[cierto al menos uno]
*	[multiplicación]	<	[menor que]	!	[negación]
/	[división]	<=	[menor o igual que]		

	Aritméticos		**Lógicos**	**Condicional**
%	[resto]	==	[igual a]	
++	[incremento]	!=	[distinto de]	
--	[decremento]			

6 LLAMADAS A MÉTODOS

- En Java todos los pasos se hacen **por valor**.
  - Si es un tipo primitivo (int, double, char, etc.) -> se pasa el valor.
  - Si es una referencia a un objeto -> se pasa el valor de dicha referencia (un puntero a un objeto)
- Así mismo el operador == compara valores.
  - Si comparamos tipos primitivos (int, double, char, etc.) -> compara que tengan el mismo valor.
  - Si comparamos referencias de objetos -> compara que apunten al mismo objeto (mismos punteros)

7 SINTAXIS GENERAL

Comentarios	// comentario de una línea
	/* comentario multilínea */
	/** javadoc */
Variables	[modificadores] tipo nombreVariable1 [,nombreVariable2,...];
Clase	[modificadores] class NombreClase [extends Clase] [implements Interfaz] { ... }

Clase abstacta	[modificadores] abstract class NombreClaseAbstracta [extends Clase] [implements Interfaz] { ... }
Interfaz	[modificadores] interface NombreInterfaz [implements Interfaz] { ... }
Métodos	[modificadores] tipoRetorno|void nombreMetodo (parámetros) [throws Excepcion] { ... }

8 ESTRUCTURAS DE CONTROL

for	for (inicio; condición; post) { ... }
while	while (condición) { ... }
do..while	do { ... } while (condición);
if..else	if (condición) { ... } else if (condición) { ... } else { ... }
switch	switch (variable) { case n1: ... break; case n2: ... break; default: ... }

9 CONVENCIONES NOMENCLATURA (I)

- En general, los nombres deben ser **simples y descriptivos**.
- Se utilizan **palabras completas**, en mayúsculas y minúsculas, con la primera letra de cada palabra en mayúscula.
- Los nombres no pueden empezar por números aunque si contenerlos.
- No se permiten caracteres extraños (salvo guión '-' y guión bajo '_')

10 CONVENCIONES NOMENCLATURA (II)

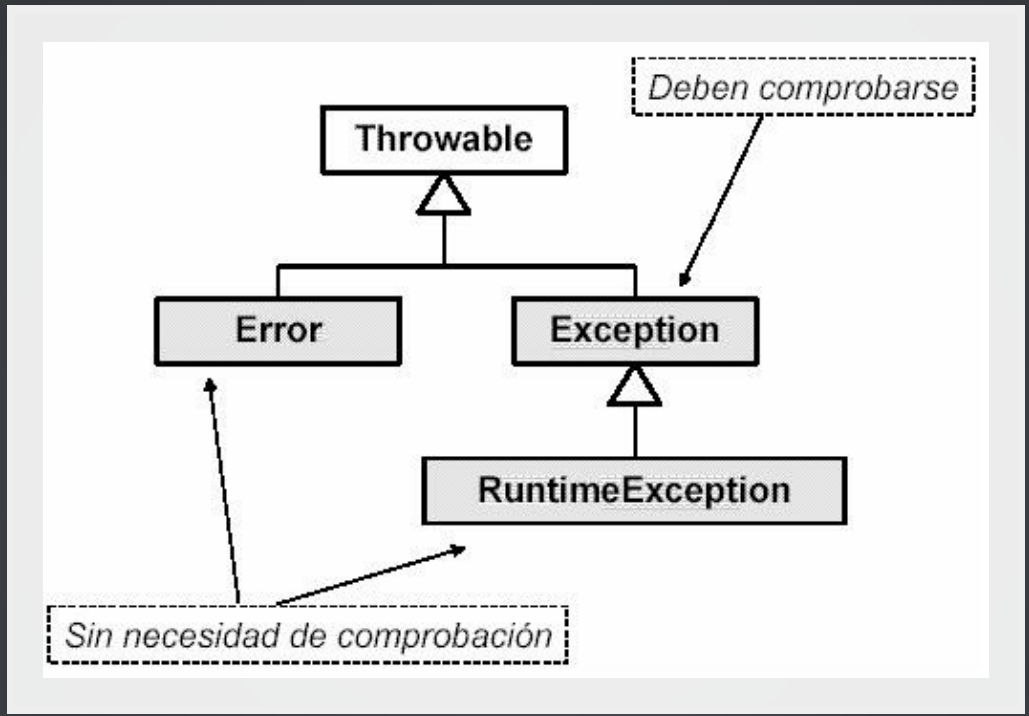
Clases	Primera letra también en mayúscula. Nombres comunes.	CocheDeCarreras

Interfaces	Primera letra también en mayúscula. Nombres comunes aunque también adjetivos. Suelen empezar por I.	IDesmontable

Métodos	Primera letra en minúsculas. Verbos o preguntas con 'is' si devuelven un boolean.	.acelerarConTurbo() .isDesmontable()
Variables	Primera letra en minúsculas. Evitar variables de una sola letra excepto (i, j, k, etc.).	numeroDeRuedas isDesmontable

Constantes	Todas en mayúsculas, con las palabras internas separadas por el signo de subrayado	MAXIMA_VELOCIDAD

11 JERARQUÍA DE EXCEPCIONES



12 CONTROL DE EXCEPCIONES

- Tipos
  - **Error**: indican problemas muy graves, que suelen ser no recuperables y no deberían ser capturadas.
  - **Exception**: problemas no definitivos, que se pueden capturar y tratar (te obligan a capturarlas o relanzarlas).
  - **RuntimeException**: problemas que se dan durante la ejecución del programa (no te obligan a capturarlas o relanzarlas).
- Tratamiento

```
try {
  // Código posiblemente problemático
} catch( tipo_de_excepcion e1) {
  // Código para solucionar la excepción e1
```

```
} catch( tipo_de_excepcion_mas_general e2) {
  // Código para solucionar la excepción e2
} finally {
  // se ejecuta siempre
}
```

13 NOVEDADES JAVA 5.0 (I)

- **Genéricos**
  - Permite no concretar en la definición con que tipo de objetos van a trabajar sus instancias.
    - [modificadores] class NombreClase<A,B,C... > [extends Clase] [implements Interfaz] { ... }
      - pudiendo sustituir <A> por <A [[extends Clase] [& Interfaz1] [& Interfaz1] ...]>
- **Enumeraciones**
  - Se caracterizan por tener un número finito y normalmente bajo de posibles valores.
    - [modificadores] enum NombreTipoEnumerado { valor1, valor2, ... }

14 NOVEDADES JAVA 5.0 (II)

- **Anotaciones**
  - Nos dan información sobre el código (clase, propiedad o método)
    - @TipoAnotacion (nombre1=valor1, nombre2=valor2, ...)
    - Sirven para:
      - que el compilador compruebe posibles errores,
      - generar documentación y/o código
      - programar con **AOP** (Aspect Oriented Programming o Programación Orientada a Aspectos)
- **Autoboxing**
  - Utilizar indistintamente un tipo primitivo o su

Utilizar indistintamente un tipo primitivo o su wrapper.

- Aunque resulta muy cómodo las conversiones tienen un alto coste computacional.
- Además hay que tener mucho cuidado con el operador de igualdad "==".

15 NOVEDADES JAVA 5.0 (III)

- **varargs**
  - Posibilidad de declarar que un método admita varios argumentos de un mismo tipo sin determinar la cantidad exacta.
    - Se añaden tres puntos al último parámetro.
- **for-each**
  - for (tipo nombreVariableIteracion: nombreContenedorIterable) { ... }
- **static imports**
  - Permite importar propiedades o métodos estáticos de una clase, interfaz o enumeración, para evitar anteponer el nombre de la clase, interfaz o enumeración en cada aparición.