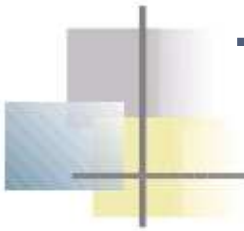
Programación en JAVA

Unidad 3
Elementos Básicos del Lenguaje

Universidad de Chile
Departamento de Ciencias de la Computación
Profesor: Felipe Aguilera V.
faguiler@dcc.uchile.cl, felipe@aguilera.cl

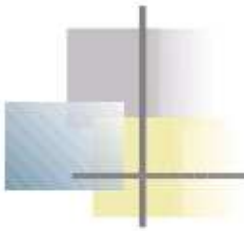


Temario

- El lenguaje de programación Java
- Tipos de aplicaciones en Java
- Tipos de datos primitivos
- Variables
- Constantes
- Arreglos
- Operadores
- Control de flujo
- String y StringBuffer

Java

- Objetivo inicial: un lenguaje de programación para dispositivos de consumo
- Requerimientos: pequeño, rápido, confiable y portable
- En 1994 se produce la explosión del Web, y Sun advierte que Java es ideal para aplicaciones Internet:
  - Independiente de la plataforma
  - Pequeño
  - Seguro



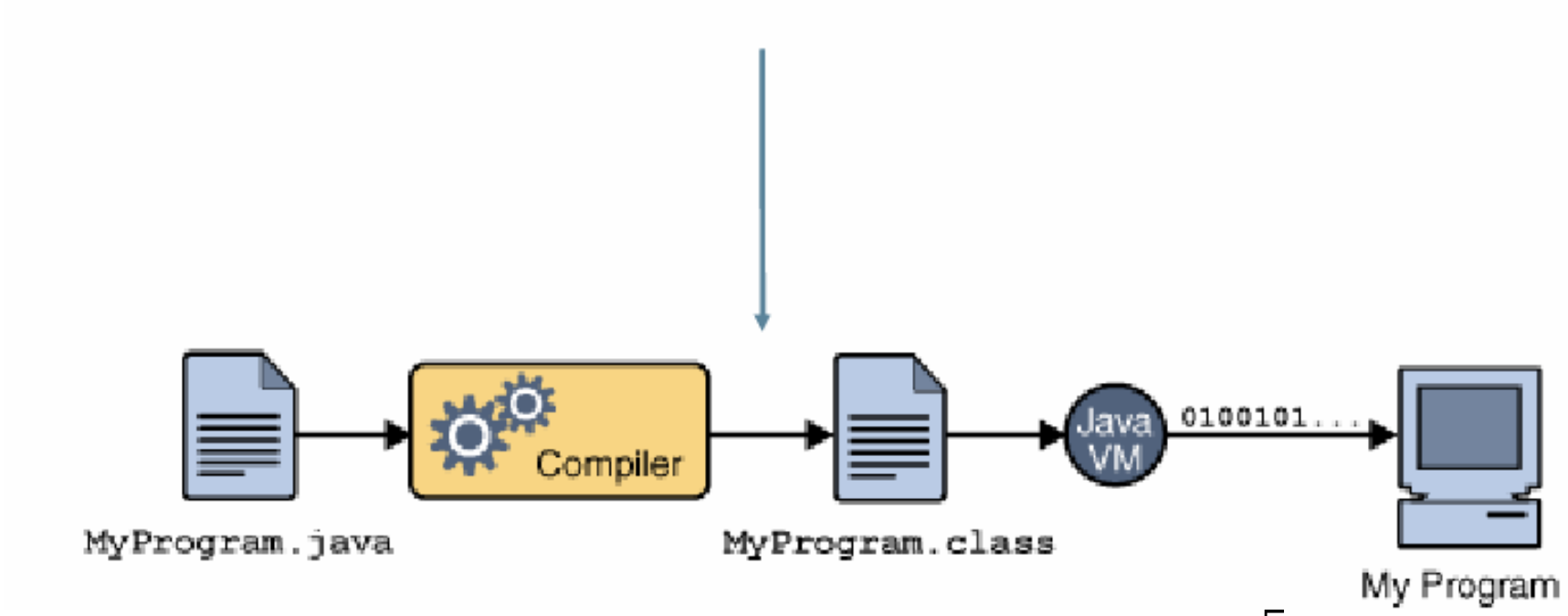
El Lenguaje Java

- Independiente de la plataforma
- Seguro
- Simple
- Robusto
- Orientado a Objetos
- Distribuido
- Multi-threaded

- Ref: http://java.sun.com/docs/overviews/java/java-overview-1.html

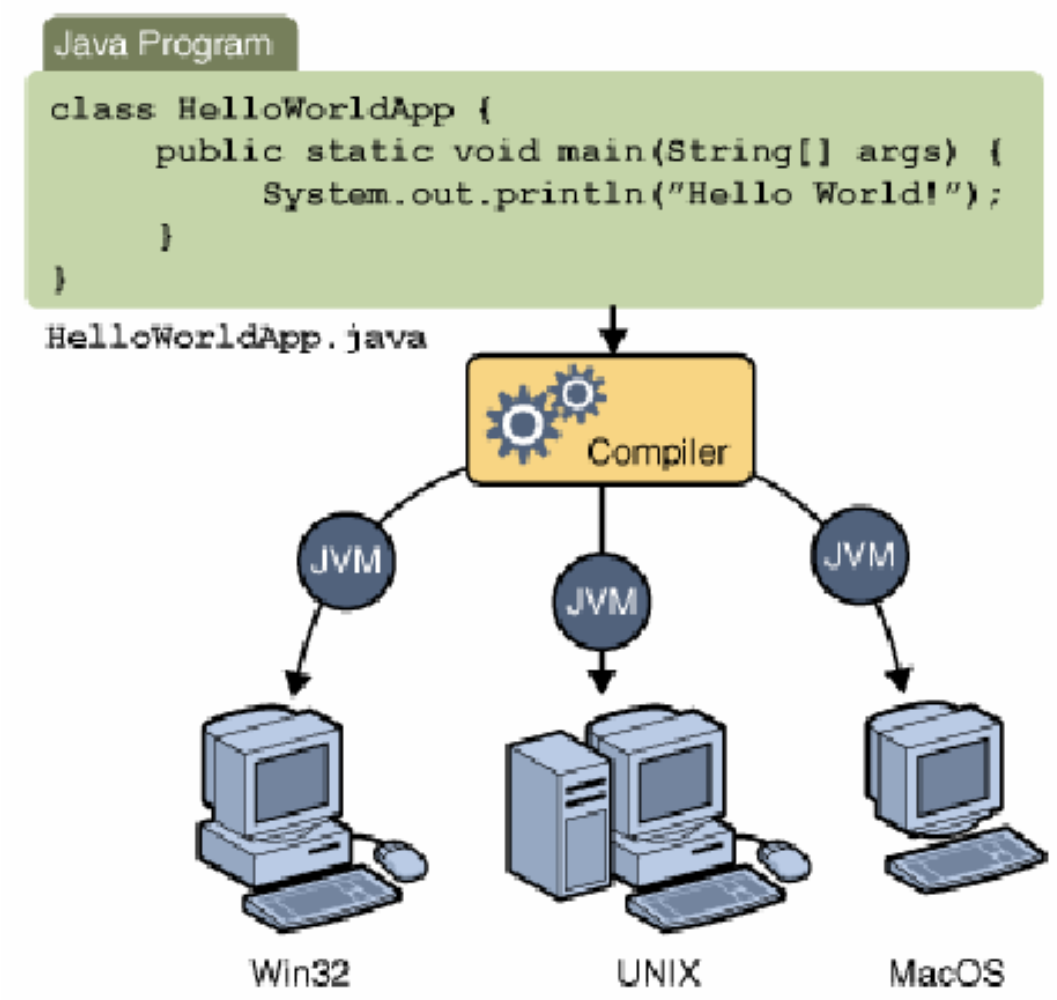
El Modelo Java

- Al compilar un programa Java, se genera un código de máquina intermedio definido por Sun, que recibe el nombre de bytecode



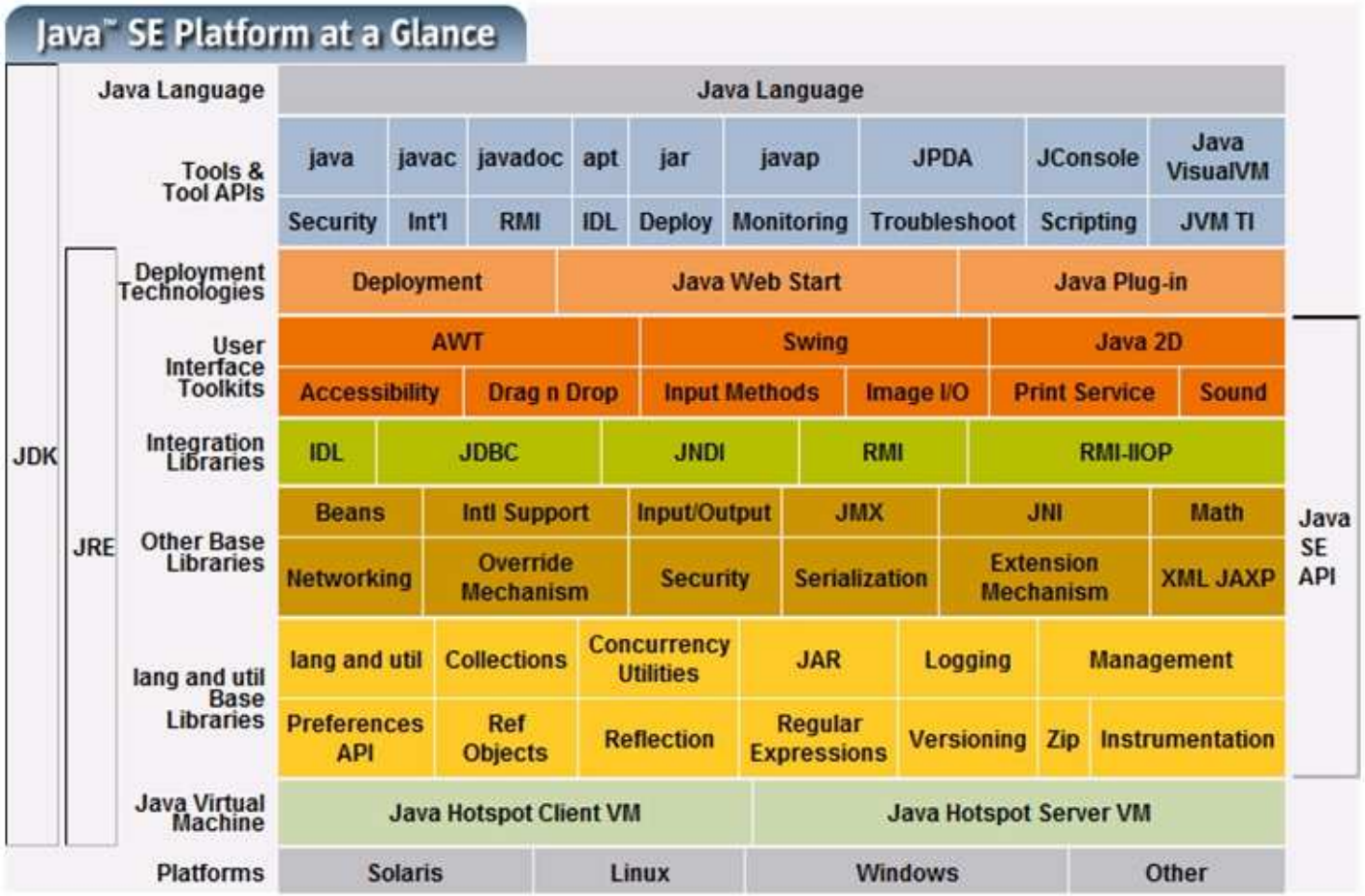
El Modelo Java

- El código bytecode es portable entre diferentes plataformas

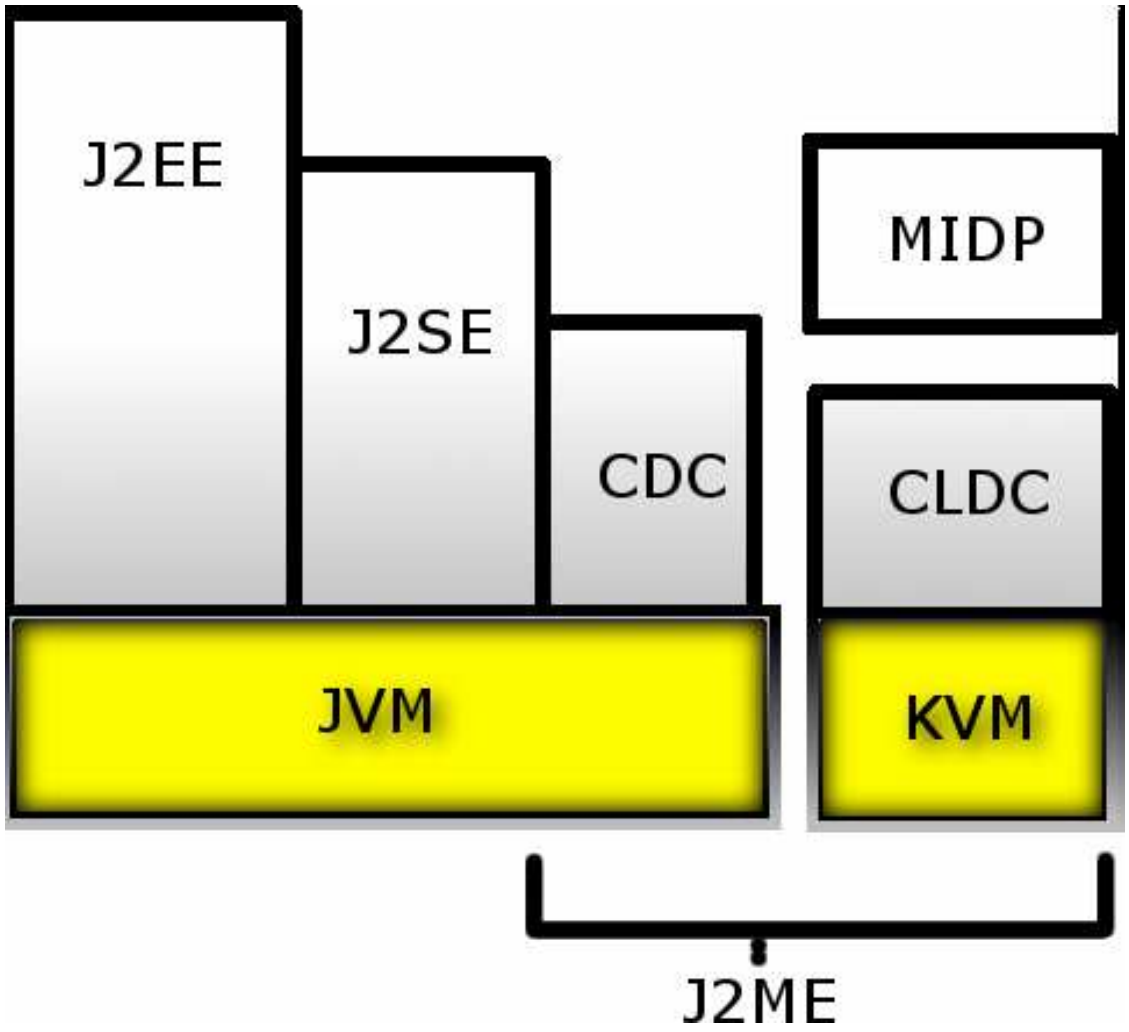


Java Standard Edition

- Java SE es la edición estándar de Java, sobre la cual están construidas Java ME (Mobile Edition) y Java EE (Enterprise Edition)

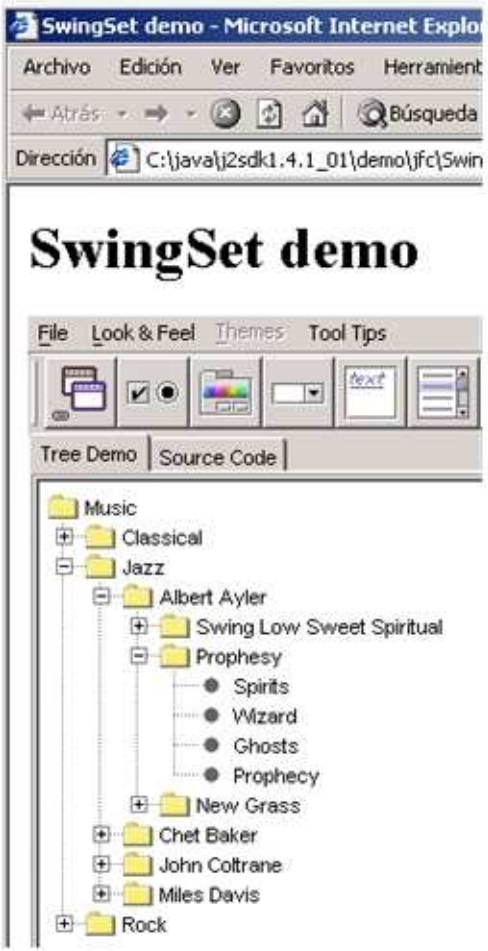


Otras Ediciones



Tipos de Aplicaciones

- Usando Java es posible escribir:
  - Aplicaciones stand-alone
  - Aplicaciones Web (servlets, JSP, applets)
  - Componentes (JavaBeans, Enterprise JavaBeans)
  - Web Services
  - ...



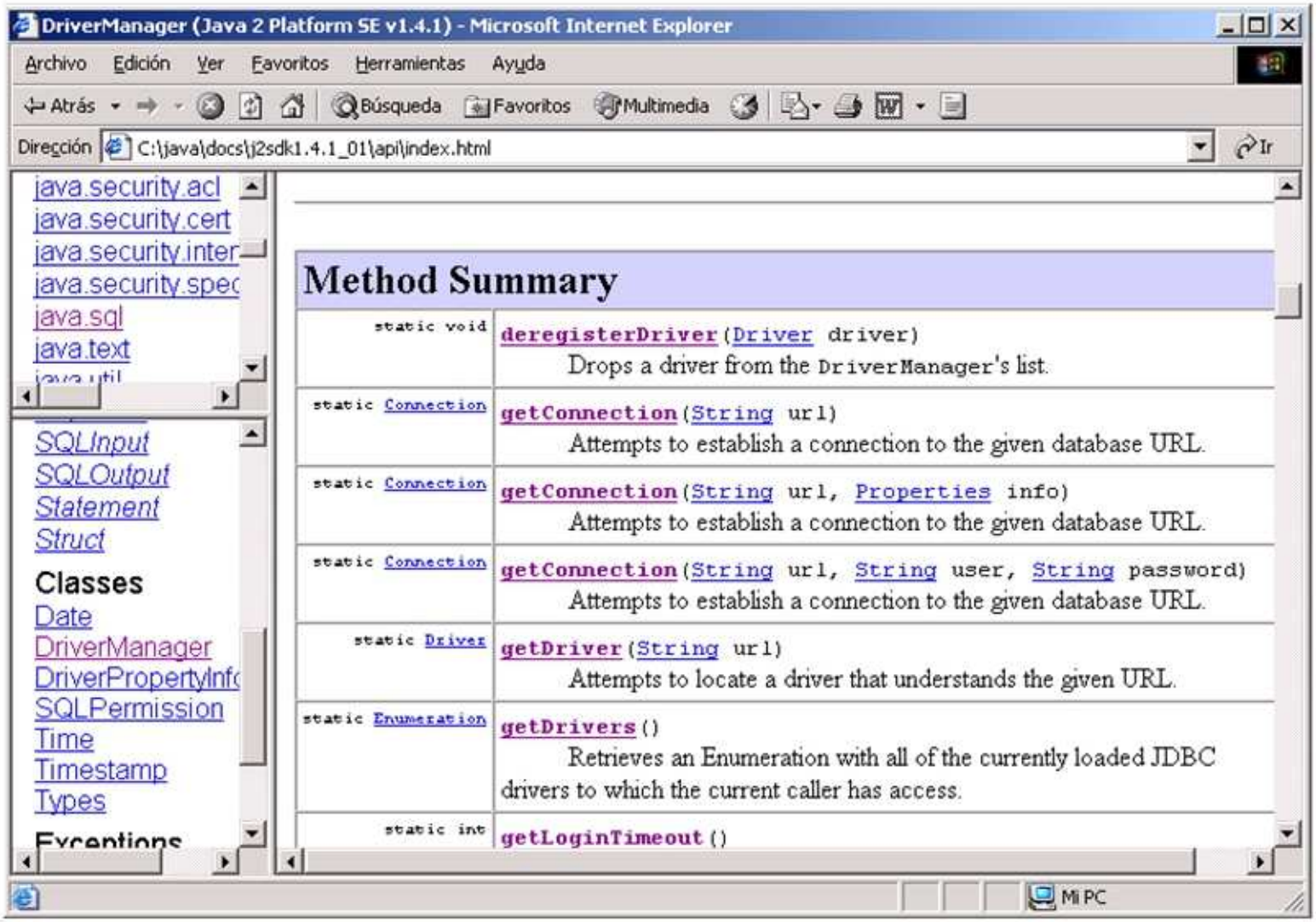
Características del Lenguaje

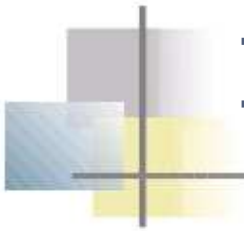
- Case-sensitive
- Todas las sentencias terminan con un punto-coma (;)
- Los espacios blancos (incluyendo tabs y fines de línea) se ignoran, excepto al interior de strings

Comentarios

// comentario	Caracteres desde // hasta el fin de línea son ignorados
/* comentario */	Caracteres entre /* y */ son ignorados
/** comentario */	Caracteres entre /** y */ son ignorados e incluidos en la documentación javadoc

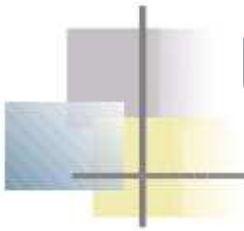
Javadoc


DriverManager (Java 2 Platform SE v1.4.1) - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos Multimedia
Dirección C:\java\docs\j2sdk1.4.1_01\api\index.html
Ir
java.security.acl
java.security.cert
java.security.inter
java.security.spec
java.sql
java.text
SQLInput
SQLOutput
Statement
Struct
Classes
Date
DriverManager
DriverPropertyInfo
SQLPermission
Time
Timestamp
Types
Exceptions
Method Summary
static void deregisterDriver(Driver driver)
Drops a driver from the DriverManager's list.
static Connection getConnection(String url)
Attempts to establish a connection to the given database URL.
static Connection getConnection(String url, Properties info)
Attempts to establish a connection to the given database URL.
static Connection getConnection(String url, String user, String password)
Attempts to establish a connection to the given database URL.
static Driver getDriver(String url)
Attempts to locate a driver that understands the given URL.
static Enumeration getDrivers()
Retrieves an Enumeration with all of the currently loaded JDBC drivers to which the current caller has access.
static int getLoginTimeout()
Mi PC



Identificadores

- Nombre dado a una variable, clase, o método
- Comienza con una letra Unicode (incluyendo _ y $), a la que pueden seguir letras y dígitos
- Puede tener cualquier tamaño
- Ejemplos
  - apellido
  - $
  - π



Unicode

- Versión extendida de la tabla de caracteres ASCII
  - Los caracteres ASCII tienen 7 bits
  - Los caracteres Unicode tienen 16 bits
- Diseñado para manejar múltiples lenguajes
- La secuencia \udddd permite codificar caracteres Unicode (d es un dígito hexadecimal)

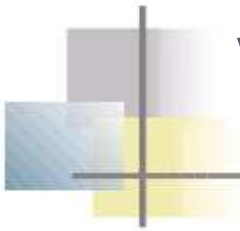
Tipos, Valores y Variables

- En Java existen 2 categorías de tipos:
  - Primitivos: boolean, char, byte, short, int, long, float, double
  - Referencias: clases, interfaces, arreglos
- Un objeto es una instancia de una clase o de un arreglo
- Una variable es un espacio de memoria que puede almacenar:

Tipo de la variable	Puede almacenar
Primitivo	Un valor del tipo
Clase	El valor null, o una referencia a una instancia de la clase o de alguna subclase de la clase
Interfaz	El valor null, o una referencia a una instancia de alguna clase que implemente la interfaz
Arreglo	El valor null, o una referencia a un arreglo de elementos del tipo del arreglo

Tipos de Datos Primitivos

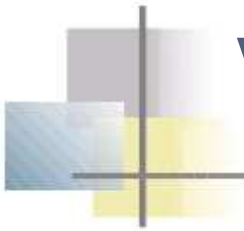
Tipo	Valores
boolean	true o false
char	Carácter Unicode (de 16 bits)
byte	Entero de 8 bits (con signo)
short	Entero de 16 bits (con signo)
int	Entero de 32 bits (con signo)
long	Entero de 64 bits (con signo)
float	Número flotante de 32 bits
double	Número flotante de 64 bits



Variables

- Una variable es un espacio de almacenamiento de datos, con un nombre y un tipo de datos asociado

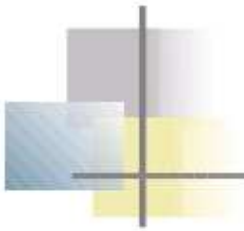
- La existencia de una variable está dada por su alcance:
  - static (variable "de clase"): se crea una vez, sin importar cuántas instancias de la clase existan
  - no static (variable "de instancia"): se crea una vez por cada instancia de la clase
  - local: se crea al ejecutarse la declaración, y desaparece al finalizar el bloque en el que fue creada



Variables Locales

- La declaración puede aparecer en cualquier punto del código fuente de un método

- La variable existe mientras se ejecuta el bloque que contiene la declaración



Bloque

- Cualquier número de sentencias agrupadas entre un par de llaves { }
- Puede ser usado en cualquier lugar donde se acepta una sentencia simple
- El bloque define el ámbito (scope) de las variables
- El bloque provee contornos para el control de flujo del procesamiento

Ambito de Variables

```
public class MiClase {
    int a;
    static int b;
    public void miMetodo( ) {
        int c;
        if (condicion) {
            int d;
        }
    }
}
```

d fuera de ámbito

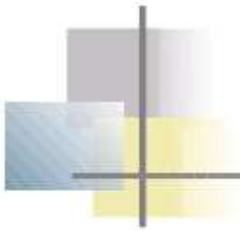
c fuera de ámbito

a y b fuera de ámbito

Valores Iniciales

- Variables de instancia y de clase
  - tipos primitivos numéricos 0
  - char '\u0000'
  - boolean false
  - referencias null

- Variables locales
  - Deben ser inicializadas explícitamente antes de ser usadas; de lo contrario se produce un error de compilación



Precedencia de Variables

1. Variables locales
2. Parámetros de métodos
3. Variables de clase y de instancia

Ej:

```
class Punto {
    private double x, y;
    public void setX(double x) {
        this.x = x;
    }
}
```

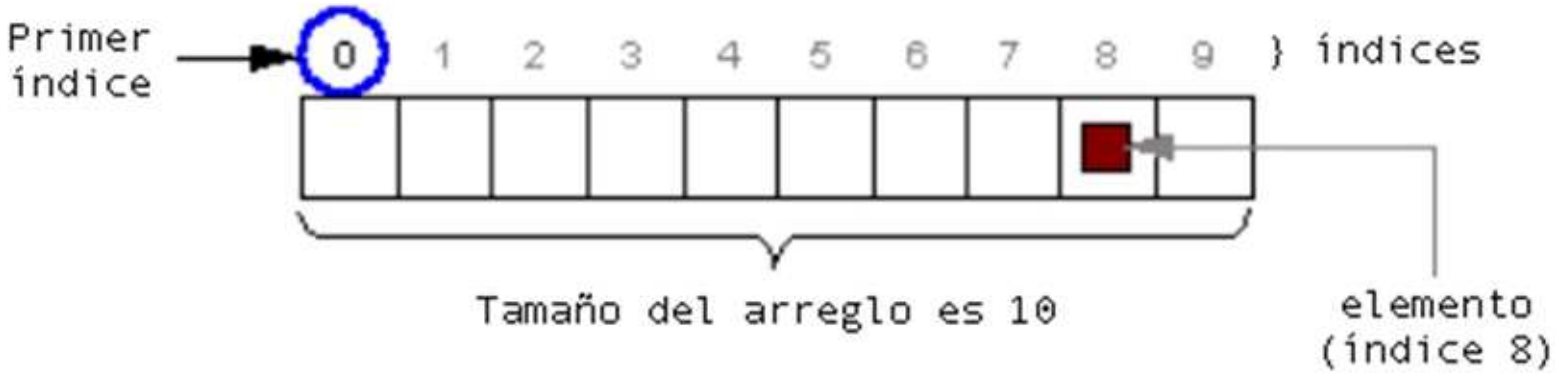
Constantes

- Una vez inicializadas, no pueden ser modificadas
- Se utiliza la palabra reservada final para definir una constante
- Típicamente las constantes se definen como static , para no instanciarlas con cada objeto
- Ejemplo

```
class Circulo {
    static final double PI = 3.1416;
}
```

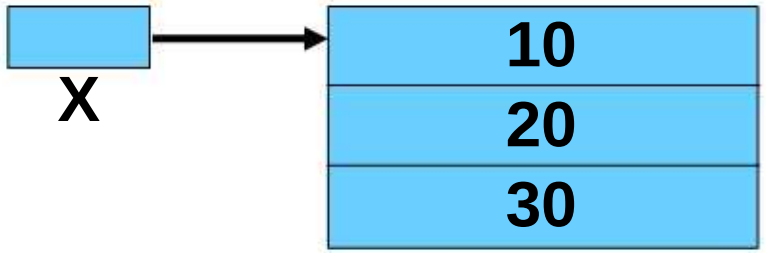
Arreglos

- Un arreglo es un objeto
- Colecciones ordenadas de elementos
  - Tipos de datos primitivos (int, …)
  - Referencias a objetos
- El tamaño es definido en la creación (new) y no puede ser modificado



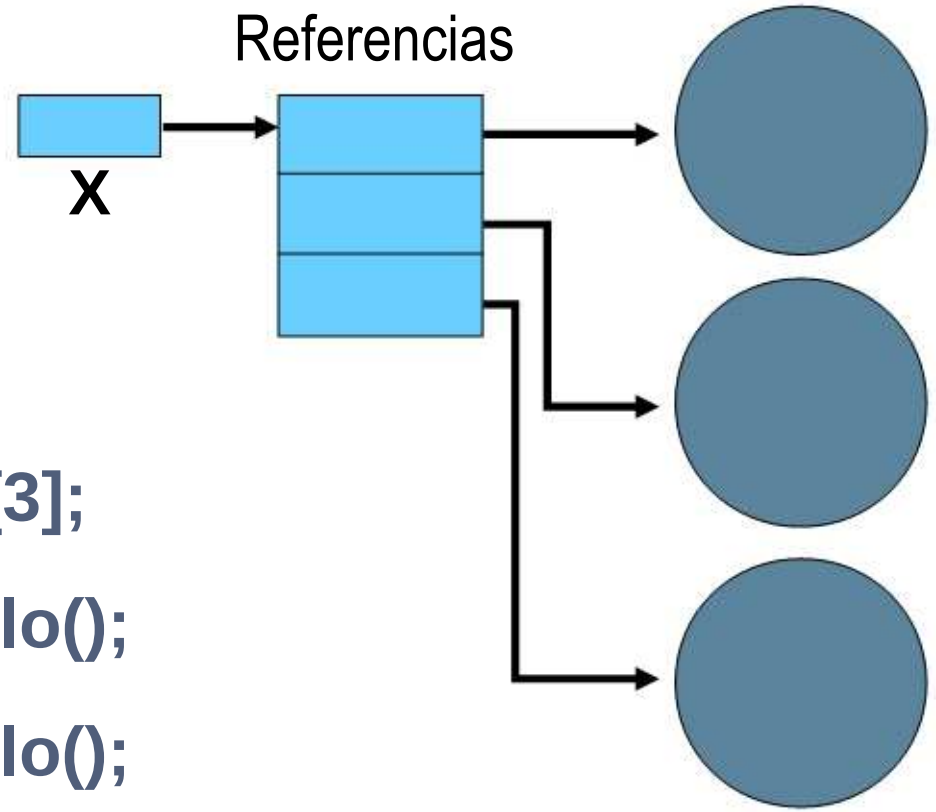
Arreglos de Tipos Primitivos

```
int[] x;          // equivalente a int x[]
x = new int[3];
x[0] = 10;
x[1] = 20;
x[2] = 30;
```



Nota: x.length es una variable read-only que entrega el tamaño del arreglo

Arreglos de Objetos



Circulo[] x;

x = new Circulo[3];

x[0] = new Circulo();

x[1] = new Circulo();

x[2] = new Circulo();

Arreglos de Arreglos

- Java permite crear arreglos de arreglos con la siguiente sintaxis

```
int[][] matriz = new int[4][];
for (int i = 0; i < matriz.length; i++) {
    matriz[i] = new int[5];
    for (int j = 0; j < matriz[i].length; j++) {
        matriz[i][j] = i + j;
    }
}
```

Inicialización de Arreglos

- Un arreglo puede inicializarse con la siguiente sintaxis:

```
boolean[] respuestas = {true, false, true};
String[]  nombres            = {"Ana María", "Carlos"};
Circulo[] circulos           = {
        new Circulo(),
        new Circulo(20),
        new Circulo(5.5)
};
String[][] humor = {
    { "Coco Legrand", "Alvaro Salas" },
    { "Les Luthiers" },
    { "Groucho Marx", "Buster Keaton",
        "Jerry Lewis", "Woody Allen" }
};
```

Tipos de Operadores

- Asignación
- Aritméticos
- Relacionales
- Lógicos

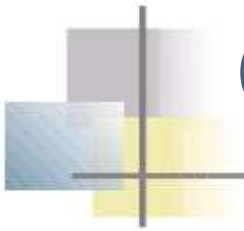
Operador de Asignación

- Se usa el símbolo = para asignar un valor
- Ejemplo

```
sueldo = 1000000;
validado = true;
```

- La asignación puede ocurrir en la declaración

```
String nombre = "Paula";
```



Operadores Aritméticos

+ Suma

- Resta

* Multiplicación

/ División

% Resto

Incremento y Decremento

- **i++** es equivalente a **i = i + 1**
  i-- es equivalente a **i = i - 1**

- El valor de la expresión es el valor de la variable antes o después de la operación, según si el operador se encuentra a la izquierda o a la derecha del operando

- Ejemplo

```
int x = 10;
System.out.println( x++ );        // x=11, print 10
System.out.println( ++x );        // x=12, print 12
System.out.println( x );          // print 12
```

Operadores Abreviados

var op= expression

es equivalente a:

var = var op (expression)

- Ejemplo

a *= b + 1;

es equivalente a:

a = a * (b + 1);

Concatenación de Strings

- Ejemplo

```
String s1 = "hola, ";
String s2 = s1 + "qué tal";
s2 += "!";
System.out.println(s2);
```

- Ejemplo

Resultado: hola, qué tal!

Operadores Relacionales

- `>` Mayor
- `>=` Mayor o igual
- `<` Menor
- `<=` Menor o igual
- `==` Igual
- `!=` Distinto

Operadores Lógicos

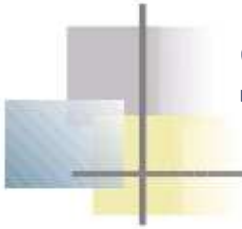
&& and

|| or

! not

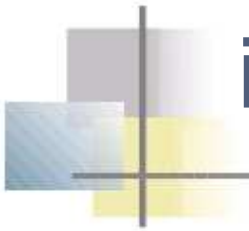
- Ejemplo

```
boolean fin =
    eof ||
    (str != null && str.equals(patron));
```



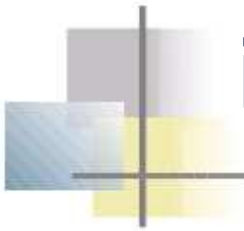
Sentencias de Control de Flujo

- if / else
- switch
- for
- while
- do / while



if

```
if ( result == 0 ) {
    System.out.println("No encontrado!");
}
```



if / else

```
if (rol.equals("administrador")) {
    // acceso a administración;
} else {
    // no tiene acceso a administración
}
```

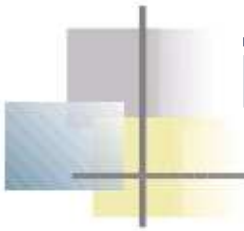
Operador "?"

- Expresión condicional:
  - condición ? expresión 1 : expresión 2
- El valor de la expresión es expresión 1 si la condición es verdadera, y expresión 2 de lo contrario
- Ejemplo

```
montoNeto = hayImpuesto ? p * 1.05 : p;
```

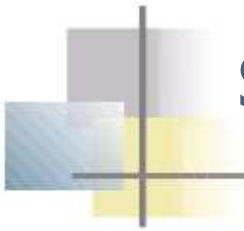
es equivalente a:

```
if (hayImpuesto) {
    montoNeto = p * 1.05;
} else {
    montoNeto = p;
}
```



if / else if / else

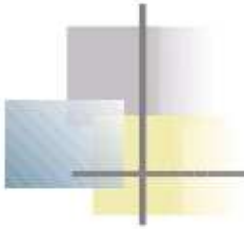
```
if ( años < 5 ) {
    vacaciones = 10;
} else if          ( años < 10 ) {
    vacaciones = 15;
} else {
    vacaciones = 20;
}
```



switch

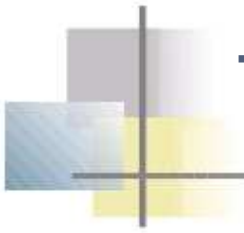
- Selección múltiple
- switch <expresión> debe evaluar un int
- case <expresión> debe ser un literal o un campo static final
- break abandona el bloque del switch (de otra manera los siguientes cases son ejecutados sin ser evaluados)
- El bloque default es opcional

```
switch (<expresión>)
{
    case <expresión>:
        break;

    case <expresión>:
        break;

    default:

}
```



Ejemplo de switch

```
public static void main(String[] args) {
    switch (args.length) {
        case 0:
            FileEditor e = new FileEditor();
            break;
        case 1:
            FileEditor e = new FileEditor(args[0]);
            break;
        default:
            // mensaje de error y fin
    }
}
```

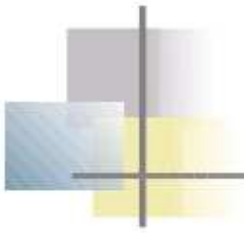


for

- Sintaxis

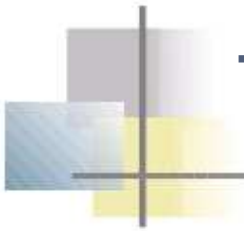
  for (inicialización; condición; reinicialización)
  { sentencias }

- Nota
  - inicialización y reinicialización pueden ser listas de expresiones separadas por comas



Ejemplo de for

```
for ( x=0; x<10; x++ ) {
    // ejecuta el bloque mientras x < 10
}

for ( x=0, y=0; y<20; x++, y+=x ) {
    // ejecuta el bloque mientras y < 20
}
```



for-each (desde Java 5.0)

- Java 5.0 introdujo un for simplificado para recorrer arreglos y colecciones
- El siguiente método retorna la suma de los elementos de un arreglo

```
// Java 1.4 o anterior
int sum(int[] a) {
      int result = 0;
      for (int i=0; i<a.length; i++) {
            result += a[i];
      }
      return result;
}
```

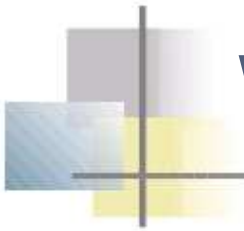
```
// Java 5.0
int sum(int[] a) {
      int result = 0;
      for (int i : a) {
            result += i;
      }
      return result;
}
```

for-each (Java 5.0)

- For-each para colecciones (con uso de generics)

```
// Java 1.4 o anterior
void cancelAll(Collection c) {
    Iterator i = c.iterator();
    while (i.hasNext()) {
        TimerTask t = (TimerTask)i.next();
        t.cancel();
    }
}
```

```
// Java 5.0
void cancelAll(Collection<TimerTask> c) {
    for (TimerTask t : c) {
        t.cancel();
    }
}
```



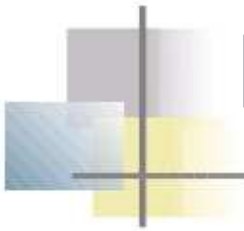
while y do while

- Sintaxis 1: se ejecuta 0 o más veces

```
while ( condición )
   { sentencias }
```

- Sintaxis 2: se ejecuta 1 o más veces

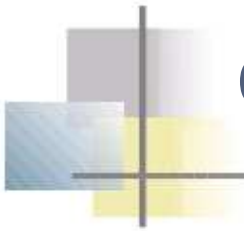
```
do
   { sentencias }
while ( condición );
```



break

- break causa el fin del ciclo en el que se encuentra

```
while ( condición ) {
    sentencias...
    if (condición de salida) {
        break;
    }
    más sentencias...
}
```



continue

- continue causa el fin de la iteración actual y el paso a la siguiente

```
while ( condición ) {
    sentencias...
    if (condición siguiente iteración) {
        continue;
    }
    más sentencias...
}
```

Etiquetas (Labels)

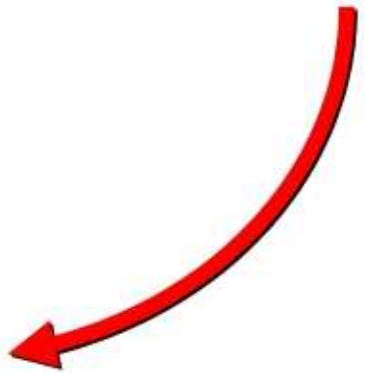
- Los labels se usan típicamente en bloques y ciclos
- Un label es un identificador seguido de dos puntos:

```
Label1:
```

- El label identifica la siguiente sentencia

Ejemplo

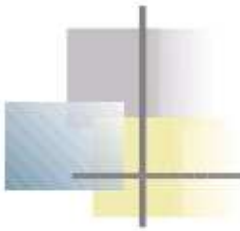
```
search:
for (...) {
    for (…) {
        if (...) {
            break search;
        }
    }
}
```



Recursividad

- Java soporta que un método se invoque a sí mismo, de manera recursiva:

```
// sumatoria
int f(int n)
{
    if (n == 0) {
        return 0;
    } else {
        return n + f(n - 1);
    }
}
```



Conversión de Tipos

- Java es fuertemente tipeado
  - Chequea compatibilidad de tipos en tiempo de compilación
  - Permite hacer cast entre tipos
  - Provee operadores para determinar el tipo de un objeto en tiempo de ejecución

Conversion Implícita

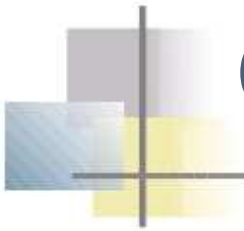
- Cualquier valor numérico puede ser asignado a una variable numérica que seoporte un rango de valores más amplio

  byte short int long float double

  más chico → más grande

- Ejemplo

```
int i = 1000;
double d = i;
```



Cast Explícito

- Cuando una conversión implícita no es posible, un cast explicita la conversión a realizar
- Sólo se permiten casts que tienen sentido

Cast de Tipos Primitivos

- Puede perderse precisión

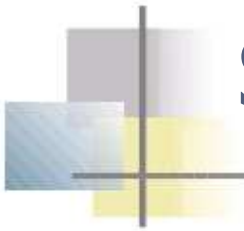
```
double d = 20.5;
long l = (long) d;
System.out.println(l);
```

20

- Pueden perderse dígitos

```
long l = 1000000;
short s;
s = (short) l;
System.out.println(s);
```

16960



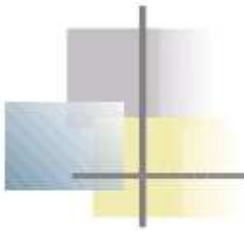
String

- En Java los strings son objetos de la clase String
- Los objetos String son inmutables, es decir, permanecen con el string que se les asignó en la inicialización
- Si se desea modificar un String, debe usarse la clase StringBuffer
- Un literal de tipo String ("hola, mundo!") da origen a un objeto de tipo String instanciado por la máquina virtual

Concatenación de Strings

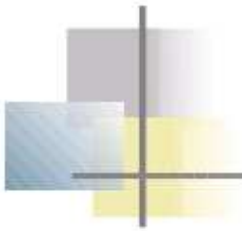
- El operador + permite concatenar Strings
- El operador + no modifica los operandos, sino que genera un nuevo objeto String con la concatenación

```
String s1 = "Hola";
String s2 = ", mundo!";
s1 = s1 + s2; // s1: "Hola, mundo!"
```



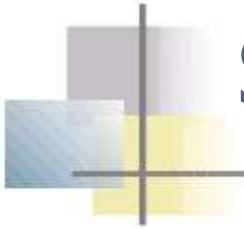
Métodos de String

- char charAt(int index): retorna el carácter en la posición indicada
- int compareTo(String s): compara con el String s, retorna 0 si el string es igual a s, un número negativo si es menor que s, y un número positivo si es mayor que s
- boolean equals(String s): compara con el String s



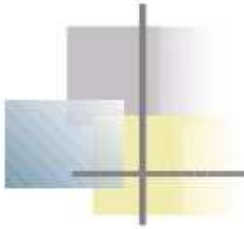
Métodos de String

- int indexOf(String str): Retorna el índice de la primera ocurrencia del substring str en el string
- int length(): retorna el tamaño del string
- String trim(): retorna una copia del string, eliminando blancos a la izquierda y a la derecha
- static String valueOf(int i): retorna un String con la representación de un entero



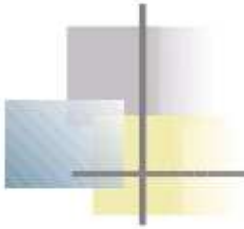
StringBuffer y StringBuilder

- StringBuffer y StringBuilder implementan una secuencia de caracteres modificable
- StringBuffer realiza sincronización, y puede ser utilizado simultáneamente por múltiples threads
- StringBuilder (desde Java 5) no realiza sincronización, está orientado a ser utilizado sin concurrencia



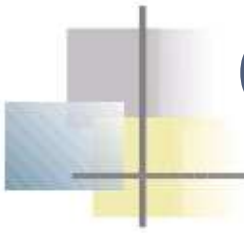
Métodos de StringBuffer

- StringBuffer(String str): crea un StringBuffer a partir de un String
- StringBuffer append(int i): concatena la variable i en el StringBuffer
- StringBuffer append(String str): concatena la variable str en el StringBuffer
- StringBuffer delete(int start, int end): elimina el substring indicado



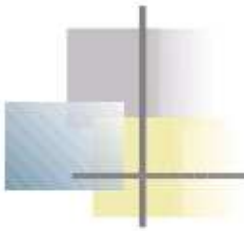
Métodos de StringBuffer

- int length(): retorna el número de caracteres del StringBuffer
- StringBuffer replace(int start, int end, String substr): reemplaza el substring entre start y end por substr
- String substring(int start, int end): retorna un nuevo string con el substring indicado
- String toString(): retorna un nuevo String a partir del StringBuffer



Clases "Wrapper"

- Boolean, Byte, Char, Double, Float, Integer, Long, Number, y Short
- Almacenan un valor de un tipo primitivo en un objeto
- Del mismo modo que la clase String, estas clases son inmutables
- Proveen métodos para manipular y convertir los valores



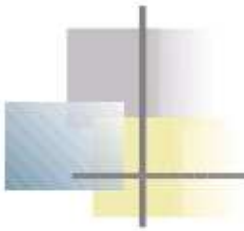
Métodos de Integer

- Integer(int value): constructor
- Integer(String value): constructor
- boolean equals(Object o): compara con otro objeto
- int intValue(): retorna el valor del objeto como un int
- static int parseInt(String s): parsea el string y retorna un int
- String toString(): retorna el objeto como string
- static String toString(int n): retorna el int indicado como un string

Autoboxing (Java 5.0)

- A partir de Java 5.0, la conversión entre tipos primitivos y objetos (entre int e Integer, por ejemplo) se produce de manera automática cuando se requiere

```
import java.util.*;

// Imprime la frecuencia de los parámetros
public class Frequency {
    public static void main(String[] args) {
        Map<String, Integer> m =
                new TreeMap<String, Integer>();
        for (String word : args) {
            Integer fr = m.get(word);
            m.put(word, (fr == null ? 1 : fr + 1));
        }
        System.out.println(m);
    }
}
```



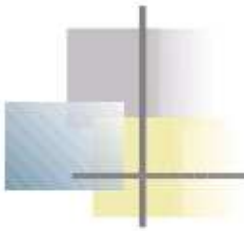
Ejemplos

- Conversión de un número a string

```
String s = String.valueOf(100);
String s = new Double(100).toString();
```

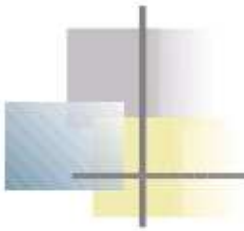
- Conversión de un string a número

```
int i = Integer.parseInt(s);
int i = new Integer(s).intValue();
```



Resumen

- El compilador Java genera bytecode que corre en cualquier sistema que implemente la Java VM
- Java es un lenguaje orientado a objetos, simple, robusto, y seguro
- Java maneja tipos primitivos y clases
- Java permite crear variables, constantes, y arreglos de una o más dimensiones
- Java provee las sentencias de control de flujo definidas en el lenguaje "C": if, switch, while, do - while , for
- Java soporta recursividad



Resumen

- Java provee la clase String, inmutable, para manipular cadenas de caracteres
- La clase StringBuffer implementa una cadena de caracteres modificable
- Las clases "wrapper" de tipos primitivos (Integer, Double, Boolean, etc.), inmutables, facilitan la conversión entre tipos de datos, y el tratamiento de datos primitivos como objetos
- Con Java 5.0, la conversión entre tipos primitivos y objetos “Wrapper” del tipo correspondiente (y viceversa) se produce de manera automática