UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INFORMÁTICA Y ANÁLISIS NUMÉRICO

INTRODUCCIÓN A JAVA

INGENIERÍA INFORMÁTICA

PRIMER CURSO DE SEGUNDO CICLO

SEGUNDO CUATRIMESTRE

Observación

- Este documento es una adaptación del documento
  - ***"Resumen de JAVA para programadores de C y C++"*** de Alejandro Castán
  - que está disponible en

    http://www.xtec.net/~acastan/textos/Java.pdf
- Se otorga el permiso para copiar, distribuir o modificar este documento bajo los términos de la licencia de documentación libre GNU, versión 1.2 o cualquier otra versión posterior publicada por la *Free Software Foundation*.
- Puedes consultar dicha licencia en

  http://www.gnu.org/copyleft/fdl.html.

Índice (1/2)

Índice (2/2)

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

1. CARATERÍSTICAS BÁSICAS DE JAVA

- **Sintaxis similar a C y C++**
  - pero sin punteros: la gestión de la memoria dinámica es automática
- **Interpretado**
  - Un programa (.java) se compila a código *bytecode* (.class)
  - El código *bytecode* lo interpreta una máquina virtual de Java.
- **Multiplataforma**
  - El programa de java se compila una única vez
  - El fichero de *bytecode* se ejecuta igual en la máquina virtual de Java de cualquier plataforma.
- **Seguro**:
  - La máquina virtual de Java impide que el programa intente ejecutar operaciones no permitidas sobre los recursos del sistema.

1. CARACTERÍSTICAS BÁSICAS DE JAVA

- **Utilidades de la línea de comandos:**
  - **Compilar**

    javac [-classpath camino] [-d carpeta] [-g] nombre.java
  - **Ejecutar**

    java [-classpath camino] nombre_clase argumentos
  - **Ejecutar applet**

    appletviewer url
  - **Depurar**

    jdb nombre_clase
  - **Documentación**

    javadoc [-classpath camino] [-d carpeta] nombre.java

1. JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

2. COMENTARIOS

- Comentarios de varias líneas (similares a los de C y C++)

```
/* Ejemplo
   maravilloso de comentario
   con varias líneas */
```

- Comentarios de una línea (similares a los del lenguaje C++)

```
// Ejemplo maravilloso de comentario de una línea
```

- Comentarios de Javadoc (similares a los de doxygen)

```
/** Comentarios que generan
   documentación automática */
```

2. COMENTARIOS

- Más sobre sobre los comentarios que generan documentación
  - Pueden incluir etiquetas **HTML**

    <code> texto /code>
  - y etiquetas **@ de campos de información**

    @author

    @version

    @see

    @param

    @return

    @throws

    ...

2. COMENTARIOS

```
/**  Clase <code>Universidad</code>
   * Contiene una lista de alumnos.
   * @author Nombre
   * @version 1.0
 */
public class Universidad {
  /** Alumnos de la universidad */
      public Vector alumnos;
     /** Método que añade un nuevo alumno a la universidad
      * @param a Alumno a añadir
      * @return Número de alumnos de la universidad
      * @throws UniversidadException Si el alumno a añadir ya existía
      */
      public int añadir(Alumno a) throws UniversidadException {
      ...
      }
}
```

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

3. DECLARACIÓN DE CONSTANTES

Java keywords				Palabras Reservadas
abstract	double	int	strictfp	
boolean	else	interface	super	false
break	extends	long	switch	null
byte	final	native	synchronized	true
case	finally	new	this	
catch	float	package	throw	
char	for	private	throws	
class	goto	protected	transient	
const	if	public	try	
continue	implements	return	void	
default	import	short	volatile	
do	instanceof	static	while	

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

4. DECLARACIÓN DE CONSTANTES

- **Constante**
  - La declaración contiene la palabra **final** delante:

    final <tipo_datos> <nombre_constante> = <valor>;
  - Ejemplo:

    final *int MAX_ELEM = 20;*
- **Tipo enumerado** (desde la versión de Java 1.5.0)
  - Java crea una nueva clase (puede tener campos y métodos)
  - Sintaxis (versión más simple):

    enum *<nombre_enumeracion>*

    {<nombre_constante>, …, <nombre_constante> }
  - Ejemplo:

    enum *Estacion { PRIMAVERA, VERANO, OTOÑO, INVIERNO }*

    Estacion a = Estacion.VERANO;

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS

- **Sintaxis** (similar a C y C++):

  <tipo_datos> <nombre_variable>;

- **Características del nombre de una variable**
  - Empieza por una **letra, _ o $**
  - Puede ir seguido por cualquier carácter unicode (incluidas vocales con acentos y letras de otros alfabetos).
  - Distingue mayúsculas de minúsculas

- **Tipos de datos simples** (no son clases):

  <tipo_datos_simple> <nombre_variable> [= <valor>];

  - Ejemplo:

    int i = 0;

5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS

- **Tipos de datos simples** (no son clases):

	Tipo	Tamaño	Rango
o	**byte**:	8 bits	–128 .. 127
o	**short** :	16 bits	-32.768 .. 32.767
o	**int**:	32 bits	-2.147.483.648 .. 2.147.483.647
o	**long**:	64 bits	-9.223.372.036.854.775.808 .. 9.223.372.036.854.775.807
o	**float**:	32 bits	-3.4x10 38 .. 3.4x10 38 (mínimo 1.4x10 –45)
o	**double**:	64 bits	-1.8x10308 .. 1.8x10308 (mínimo 4.9x10-324)
o	**boolean**:		true o false
o	**char**:	16 bits	unicode

5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS

- **Tipos de datos compuestos** (son clases):
  - ***String***
  - Vectores y matrices
  - Colecciones,
  - Clases
  - wrappers/envolventes

  <tipo_datos_compuesto> <nombre_variable> =

  new <tipo_datos> (<valor_inicial>);

- **Ejemplos**

  int *v[] = n****ew int****[100]; // Tipo compuesto vector*

  String *s = new* ***String****("Hola"); // Tipo compuesto String*

  // Tipo compuesto ***Integer*** *(wrapper del tipo* ***int****)*

  Integer *n = new* ***Integer(10)****;*

5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS

- **Ejemplos de conversión**

```
int i = Integer.parseInt("12"); // cadena caracteres a entero
float f = Float.parseFloat("1.15"); // cadena caracteres a real
String s1 = new Float(f).toString(); // real a cadena caracteres
String s2 = new Integer(i).toString(); // entero a cadena caracteres

// más fácil: números a cadena caracteres
String s3 = "Números:" + i + " y " + f;

i = (int) f; // convertir real a entero
f = i; // convertir entero a real
```

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. DECLARACIÓN DE CONSTANTES
4. PALABRAS CLAVES Y PALABRAS RESERVADAS
5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

6. EL TIPO DE DATOS CADENA DE CARACTERES

- **String**: tipo de datos cadena de caracteres

  String <nombre_variable> [= "<cadena de caracteres>"];

- **Ejemplos:**

  String s1 = "hola";

  String s2 = "adios";

  String s3 = s1 + " y " + s2 + " : " + 2004;

6. EL TIPO DE DATOS CADENA DE CARACTERES

- **Atributos y métodos más importantes:**

string1.**equals**(string2) // Compara dos strings

string1.**clone**() // crea una copia de un string

string1.**charAt**(posicion) // retorna el carácter en una posición

string1.**concat**(string2) // concatena dos strings

string1.**indexOF**(caracter, posicion) // devuelve la posición de un carácter

string1.**length**() // devuelve la longitud del string

string1.**replace**(caracter1, caracter2) // reemplaza un carácter por otro

string1.**substring**(posicion1, posicion2) // extrae una porción del string

string1.**toLowerCase**() // convierte el string a minúsculas

string1.**toUpperCase**() // convierte el string a mayúsculas

string1.**valueOf**(numero) // convierte un número a string

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

7. LOS TIPOS DE DATOS ENVOLVENTES

- Existe una **clase equivalente** para cada tipo de dato simple

Tipo Simple		Clase Equivalente
byte, short, int, long	→	Number, Byte, Short, Integer, Long
float, double	→	Number, Float, Double
boolean	→	Boolean
char	→	Character

  - Estas clases poseen constantes y métodos que pueden ser útiles

- **Ejemplos**:

```
int i = 4; // "i" es una variable entera (tipo simple)

// "j" es un objeto de la clase Integer (envolvente)
Integer j = new Integer(5);
```

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

8. LOS TIPOS DE DATOS VECTOR Y MATRIZ

- **Sintaxis similar a C y C++**
  - Se indexa con [ ]
  - El primer elemento está en la posición 0
- **Deferencias con C y C++**
  - El número de elementos se declara dinámicamente con new
  - Se puede obtener su longitud con ***length***

8. LOS TIPOS DE DATOS VECTOR Y MATRIZ

- **Declaración**
  - Una dimensión:

    <tipo_datos> <nombre_array>[];

    <nombre_array> = ***new*** *<tipo_datos>[<num_elementos>];*
  - Varias dimensiones:

    <tipo_datos> <nombre_array>[][]…[] ;

    <nombre_array> = ***new*** *<tipo_datos> [<n1]…[<nk>];*

8. LOS TIPOS DE DATOS VECTOR Y MATRIZ

- **Ejemplos:**

```
float v[] = new float[10]; // Una dimensión con 10 elementos
float m[][]= new float[3][4]; // Dos dimensiones con 3x4 elementos
// Una dimensión con dos elementos inicializados
String s[] = {"hola", "adios"};

for (int i = 0; i < v.length; i++)
        v[i] = i;

for (int i = 0; i < m.length; i++)
        for (int j = 0; j < m[i].length; j++)
                m[i][j] = 0;
```

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES, Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

9. LAS COLECCIONES

- Proporcionadas por la API (Interfaz de Programación de Aplicaciones) de Java
  - List
  - ArrayList
  - Vector
  - Set
  - Map
  - ...
- Características
  - **No** forman parte del lenguaje original
  - Son clases definidas en el paquete java.util

9. LAS COLECCIONES

- Ejemplo:

```
ArrayList<Alumno> miclase = new ArrayList<Alumno>();

miclase.add( new Alumno("Pepe", 5.0) );
miclase.add( new Alumno("Alex", 4.2) );
miclase.add( new Alumno("Pepa", 6.3) );

for (Iterator i = miclase.iterator(); i.hasNext();  )
{
    System.out.println( i.next() );
}
```

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES, Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

10. CÓDIGO

- Java utiliza la misma sintaxis que C y C++:
  - Una instrucción
    - puede ocupar varias líneas
    - acaba en punto y coma: “;”

      instrucción;
  - Los bloques de instrucciones van entre llaves “{“ “}”:

    {

    instrucción1;

    instrucción2;

    …

    }

1. CARACTERÍSTICAS BÁSICAS DE JAVA
2. COMENTARIOS
3. PALABRAS CLAVES Y PALABRAS RESERVADAS
4. DECLARACIÓN DE CONSTANTES
5. DECLARACIÓN DE VARIABLES Y TIPOS DE DATOS
6. EL TIPO DE DATOS CADENA DE CARACTERES
7. LOS TIPOS DE DATOS ENVOLVENTES
8. LOS TIPOS DE DATOS VECTOR Y MATRIZ
9. LAS COLECCIONES
10. CÓDIGO
11. EXPRESIONES Y OPERADORES

...

11. Java utiliza los mismos operadores que C y C++:

- Java utiliza los mismos operadores que C y C++:
  - Asignación:
    - =
  - Aritméticos:
    - ++, --, +, -, *, /, %
  - Relacionales:
    - ==, !=, <, <=, >, >=, !, &&, ||, ?:
  - Bits:
    - &, |, ^, ~, <<, >>, >>>
  - Conversión:
    - (tipo)

12. SENTENCIAS
13. LIBRERÍAS
14. ENTRADA Y SALIDA POR PANTALLA
15. ENTRADA Y SALIDA POR FICHEROS
16. PROGRAMA PRINCIPAL Y ARGUMENTOS
17. CLASES
18. OBJETOS
19. MÁS SOBRE CLASES Y OBJETOS
20. INTERFACES
21. APPLETS
22. EXCEPCIONES

12. SENTENCIAS

- **Sentencias de control del flujo de ejecución** (igual que en C y C++):
  - Condicional simple y compuesta: ***if … else***
  - Condicional múltiple: ***switch***
  - Iterativa con condición inicial: ***while***
  - Iterativa final condición final: ***do…while***
  - Repetitiva: **for**
  - Otras
    - ***break;***
    - ***continue;***
    - ***label:***
    - ***return*** *<valor>;*
    - ***exit;***

12. SENTENCIAS

- **Condicional simple y compuesta**

```
if (condición) {
        instrucciones;
    }
[else  {
        instrucciones;
    }]
```

Ejemplo:

```
if (a != 0) {
    System.out.println("x = " + -a/b);
    }
 else {
    System.out.println("Error");
    }
```

12. SENTENCIAS

- **Condicional múltiple**

```
switch (expresión) {
    case <valor>: instrucciones;
            [break;]
    case <valor>: instrucciones;
            [break;]
    ...
    [default: instrucciones;]
    }
```

12. SENTENCIAS

- **Condicional múltiple**
  - **Ejemplo**

```
switch (opcion) {
        case 1: x = x * Math.sqrt(y);
                break;
        case 2:
        case 3: x = x / Math.log(y);
                break;
        default: System.out.println("Error");
}
```

12. SENTENCIAS

- **Iterativa con condición inicial:**

```
while (condición)
 {
    instrucciones;
}
```

  - **Ejemplo**

```
i = 0;
while (i < 10)
{
        System.out.println(v[i]);
        i++;
}
```

12. SENTENCIAS

- **Iterativa con condición final:**

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

- **Ejemplo**

```
i = 0;
do {
        System.out.println(v[i]);
        i++;
} while (i < 10)
```

12. SENTENCIAS

- **Repetitiva**

```
for (inicialización; comparación; incremento)
{
    instrucciones;
}
```

  - **Ejemplo:**

```
for (i = 0; i < 10; i++)
{
    System.out.println(v[i]);
}
```

12. SENTENCIAS

- **Repetitiva:**
  - **Nuevo for (desde Java 1.5.0)**

```
int primeros_primos[] = {1, 2, 3, 5, 7,11};

for (int j : primeros_primos)
    System.out.println(j);
```

12. SENTENCIAS
13. LIBRERÍAS
14. ENTRADA Y SALIDA POR PANTALLA
15. ENTRADA Y SALIDA POR FICHEROS
16. PROGRAMA PRINCIPAL Y ARGUMENTOS
17. CLASES
18. OBJETOS
19. MÁS SOBRE CLASES Y OBJETOS
20. INTERFACES
21. APPLETS
22. EXCEPCIONES

13. LIBRERÍAS

- Creación de clases dentro de una librería
  - Al principio del fichero de la clase, se debe escribir la librería dónde se insertará la clase:

    package *carpeta.subcarpeta. …;*

- Para utilizar código de otro fichero:

  import *carpeta.subcarpeta. … .clase;*

  - Ejemplo:

    import *java.lang.Math;*

    import *java.io.*;*

13. LIBRERÍAS

- **Librería más comunes**
  - **java.lang**
    - Clases básicas o fundamentales del lenguaje
  - **java.awt**
    - Permite definir interfaces gráficas
  - **java.io**
    - Entrada/salida de datos
  - **java.net**
    - Comunicación por red
  - **java.applet**
    - Declaración de *applets*
  - **java.util**
    - Colecciones, eventos, fecha, hora, etc.

13. LIBRERÍAS

- **Otras librerías**
  - **java.beans**
    - Clases de la arquitectura JavaBeans
  - **java.math**
    - Operaciones matemáticas.
  - **java.nio**
    - Define *buffers* (contenedores de datos) y proporciona un acceso a otros paquetes de NIO (*New Input / Output*).
  - **java.rmi**
    - El paquete RMI (*Remote Method Invocation*) permite acceder a datos o métotos "remotos".
  - **java.security**
    - Permite el control de seguridad.

13. LIBRERÍAS

- **Otras librerías**
  - **java.sql**
    - API para acceder y procesar bases de datos
  - **java.text**
    - Maneja texto, fechas, números y mensajes independientes del lenguaje natural
  - **javax.accessibility**
    - Favorece el acceso a componentes de la interfaz del usuario.
  - **javax.crypto**
    - Permite realizar operaciones criptográficas.
  - **javax.imageio**
    - Paquete principal para procesar entrada y salida de imágenes.

13. LIBRERÍAS

- **Otras librerías: extensiones estándares de Java**

  - **javax.management**
  - **javax.naming**
  - **javax.print**
  - **javax.rmi**
  - **javax.security.auth**
  - **javax.sound.midi**
  - **javax.sound.sampled**
  - **javax.sql**
  - **javax.swing**
  - **javax.xml**
  - **org.ietf.jgss**
  - **org.omg.CORBA**
  - **org.w3c.dom**
  - **org.xml.sax**

12. SENTENCIAS
13. LIBRERÍAS
14. ENTRADA Y SALIDA POR PANTALLA
15. ENTRADA Y SALIDA POR FICHEROS
16. PROGRAMA PRINCIPAL Y ARGUMENTOS
17. CLASES
18. OBJETOS
19. MÁS SOBRE CLASES Y OBJETOS
20. INTERFACES
21. APPLETS
22. EXCEPCIONES

14. ENTRADA Y SALIDA POR PANTALLA

- **Lectura desde el teclado**
  - Sólo se pueden leer líneas de texto, que después deben ser convertidas al tipo correspondiente:

```
BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
String  s = in.readLine();
int  i = Integer.parseInt(in.readLine());
```

- Lectura más fácil con la clase **java.util.Scanner** (desde Java 1.5.0)

```
Scanner in = new Scanner(System.in);
String s = in.next();
int i = in.nextInt();
in.close();
```

14. ENTRADA Y SALIDA POR PANTALLA

- **Escritura en la pantalla**
  - Se concatenan cadenas de caracteres y variables en un único String:

    System.out.println("Hola " + s + i);

  - *También se puede escribir texto formateado al estilo de C (desde la versión 1.5.0 de Java)*

    System.out.printf("Hola %s %5d", s, i);

12. SENTENCIAS
13. LIBRERÍAS
14. ENTRADA Y SALIDA POR PANTALLA
15. ENTRADA Y SALIDA POR FICHEROS
16. PROGRAMA PRINCIPAL Y ARGUMENTOS
17. CLASES
18. OBJETOS
19. MÁS SOBRE CLASES Y OBJETOS
20. INTERFACES
21. APPLETS
22. EXCEPCIONES

15. ENTRADA Y SALIDA POR FICHEROS

- Multitud de clases para trabajar con entrada y salida de datos:
  - Clases para acceder directamente a los dispositivos:
    - Teclado
    - Pantalla
    - Ficheros
    - ...
  - Clases para filtrar, acumular o comprimir los datos.

15. ENTRADA Y SALIDA POR FICHEROS

- **Byte Input Stream**
  - InputStream:
    - La superclase de todos las clases "*byte input streams*".

  - BufferedInputStream
  - ByteArrayInputStream
  - CheckedInputStream
  - DataInputStream
  - FileInputStream
  - FilterInputStream
  - GZIPInputStream
  - InflaterInputStream
  - LineNumberReader
  - ObjectInputStream
  - PipedInputStream
  - PushbackInputStream
  - SequenceInputStream
  - StringBufferInputStream

15. ENTRADA Y SALIDA POR FICHEROS

- **Character (unicode) Input Stream**
  - BufferedReader
  - CharArrayReader
  - FileReader
  - FilterReader
    - La superclase de todas las clases "*character input stream filter*".
  - InputStreamReader
  - LineNumberReader
  - PipedReader
  - PushbackReader
  - Reader
    - La superclase de todas las clases "*character input streams*"
  - StringReader

15. ENTRADA Y SALIDA POR FICHEROS

- **Byte Output Stream (1/2)**
  - BufferedOutputStream
  - ByteArrayOutputStream
  - CheckedOutputStream.
  - DataOutputStream
  - DeflaterOutputStream
    - Superclase de GZIPOutputStream y ZipOutputStream.
  - FileOutputStream
  - FilterOutputStream
    - Superclase de todas las clases "*byte output stream filters*".

15. ENTRADA Y SALIDA POR FICHEROS

- **Byte Output Stream (2/2)**
  - GZIPOutputStream
  - ObjectOutputStream
  - OutputStream
    - Superclase de todas las clases "*byte output streams*".
  - PipedOutputStream
  - PrintStream
  - ZipOutputStream

15. ENTRADA Y SALIDA POR FICHEROS

- **Character (unicode) Output Stream**
  - BufferedWriter
  - CharArrayWriter
  - FileWriter
  - FilterWriter
    - La superclase de todas las clases "*character output streams filters*".
  - OutputStreamWriter
  - PipedWriter
  - PrintWriter
  - StringWriter
  - Writer
    - La superclase de todas las clase "*character output streams*".

15. ENTRADA Y SALIDA POR FICHEROS

- **Ejemplo 1**
  - **Trabajar con ficheros binarios de datos "crudos" (imágenes, …):**

```
int c;
FileInputStream fichero1 = new FileInputStream("entrada");
FileOutputStream fichero2 = new FileOutputStream("salida");
while ((c = fichero1.read()) != -1)
 {
    fichero2.write(c);
}
fichero1.close();
fichero2.close();
```

15. ENTRADA Y SALIDA POR FICHEROS

- **Ejemplo 2**
  - **Trabajar con ficheros binarios de tipos simples (*int*, *float*, ...) y acceso directo:**

```
RandomAccessFile fichero1= new RandomAccessFile("radio", "r");
RandomAccessFile fichero2= new RandomAccessFile("perimetro", "w");
try {
    while (true) {
            fichero2.writeFloat( 2 * 3.1416 * fichero1.readFloat() );
    }
    catch (EOFException e) {
            fichero1.close();
            fichero2.close();
}
```

15. ENTRADA Y SALIDA POR FICHEROS

- **Ejemplo 3**
  - **Trabajar con ficheros de texto:**

```
String salario;
FileReader      fichero1 = new FileReader("salarios.txt");
FileWriter      fichero2 = new FileWriter("salarios.new");
BufferedReader in = new BufferedReader(fichero1);
BufferedWriter out = new BufferedWriter(fichero2);
while ((salario = in.readLine()) != null) {
    salario = (new Integer(Integer.parseInt(salario)*10).toString();
    out.write(salario);
}
fichero1.close();
fichero2.close();
```

15. ENTRADA Y SALIDA POR FICHEROS

- **Ejemplo 4 (1/2)**
  - **Guardar y recuperar objetos (serialización):**

```
class Alumno implements Serializable {
    private String nom;
    private String id;
    private int edad;
    private transient double nota;
    ...
}
class Grupo implements Serializable {
    private Alumno [] alumnos = new Alumno[20];
    ...
}
```

15. ENTRADA Y SALIDA POR FICHEROS

- **Ejemplo 4 (2/2)**
  - **Guardar y recuperar objetos (serialización):**

```
Grupo a1 = new Grupo();
a1.insertar(new Alumno("Pepe", "43510987F", 25, 8.5));
a1.insertar(new Alumno("Pepa", "65471234H", 18, 9.3));
...
ObjectOutputStream out =
      new ObjectOutputStream(new FileOutputStream("grupoA1"));
out.writeObject(a1);
out.close();
Grupo a2 = new Grupo();
ObjectInputStream in =
      new ObjectInputStream(new FileInputStream("grupoA1"));
a2 = (Grupo) in.readObject();
in.close();
```

15. ENTRADA Y SALIDA POR FICHEROS

- **Obtener información de ficheros y directorios (1/2)**
  - *Métodos de la clase File para extraer cualquier tipo de información:*
    - *canRead*
    - *canWrite*
    - *exists*
    - *isHidden*
    - *isDirectory*
    - *isFile*
    - *createNewFile*
    - *createTempFile*
    - *delete*
    - *renameTo*
    - *mkdirs*
    - *setReadOnly*
    - *getName*
    - *getPath*
    - *toURL*
    - *length*
    - *lastModified,*
    - *list*
    - *listFiles*
    - *....*

15. ENTRADA Y SALIDA POR FICHEROS

- **Obtener información de ficheros y directorios (2/2):**
  - **Ejemplo**

```
File raiz = new File(“./");
String [] dir = raiz.list();

for (i = 0; i < dir.length; i++)
    System.out.println(dir[i]);
```

12. SENTENCIAS
13. LIBRERÍAS
14. ENTRADA Y SALIDA POR PANTALLA
15. ENTRADA Y SALIDA POR FICHEROS
16. PROGRAMA PRINCIPAL Y ARGUMENTOS
17. CLASES
18. OBJETOS
19. MÁS SOBRE CLASES Y OBJETOS
20. INTERFACES
21. APPLETS
22. EXCEPCIONES

16. PROGRAMA PRINCIPAL Y ARGUMENTOS

- **Programa principal en una clase Java:**

```
class <Nombre>
{
    public static void main (String[] args)
    {
        instrucciones;
    }
}
```

16. PROGRAMA PRINCIPAL Y ARGUMENTOS

- **Programa principal en una clase Java:**
  - **Ejemplo**

```
// Imprime una palabra (primer argumento)
// un número determinado de veces (segundo argumento)
class Mensaje {
    public static void main(String [] args) {
        if (args.length == 2) {
            for (int i = 1; i <= Integer.parseInt(args[1]); i++) {
                System.out.println(args[0] + " : " + i);
            }
        }
}
```

16. PROGRAMA PRINCIPAL Y ARGUMENTOS

- **Programa principal en un applet Java:**

```
class <Nombre> extends Applet
{
    public void init()
    {
        instrucciones;
    }
}
```

16. PROGRAMA PRINCIPAL Y ARGUMENTOS

- **Programa principal en un applet Java:**
  - **Ejemplo**

```
import java.awt.*;
import java.applet.*;


public class HolaMundoApplet extends Applet {
    public void init() {
            repaint();
    }
public void paint(Graphics g) {
            g.drawString("Hola Mundo!", 25, 25);
    }
}
```

16. PROGRAMA PRINCIPAL Y ARGUMENTOS

- **Programa principal en un applet Java:**
  - **Ejemplo**
    - **Insertar en una página web HTML el applet:**

```
<html>
    <body>
        <applet code="HolaMundoApplet“
                width=300
                height=50>
        </applet>
    </body>
</html>
```

17. CLASES

```
class <Nombre> [extends <Nombre_clase_padre>]
{  // declaración de atributos
     visibilidad [modificadores] tipo atributo1 [= valor];
     visibilidad [modificadores] tipo atributo2 [= valor];
     ...
     // declaración de métodos
     visibilidad [modificadores] tipo metodo1(argumentos) {
          instrucciones;
     }
     visibilidad [modificadores] tipo metodo2(argumentos) {
               instrucciones;
     }
     ...
}
```

17. CLASES

- Donde:
  - **visibilidad**:
    - *public*
    - *protected*
    - *private*
  - **modificadores**
    - *final*
    - *static*
    - *abstract*
  - **argumentos**
    - declaración de variables separadas por comas

17. CLASES

- **Ejemplo (1/2)**

```
class Complejo
{
    private double re, im;
    public Complejo(double re, double im) {
        this.re = re;
        this.im = im;
    }
    public String toString() {
        return (new String(re + "+" + im + "i"));
    }
...
```

17. CLASES

- **Ejemplo (2/2)**

```
...
    public boolean equals(Complejo v) {
        return ((re == v.re) && (im == v.im));
    }
    public double modulo() {
        return (Math.sqrt(re*re + im*im));
    }
    public void suma(Complejo v) {
        re = re + v.re;
        im = im + v.im;
    }
} // Fin de la clase Complejo
```

17. CLASES

- **Número variable de argumentos** (desde la versión 1.5.0 de Java)

```
visibilidad [modificadores] tipo metodo(Object ... args) {
        instrucciones;
}
```

- **Ejemplo:**

```
public void suma(Complejo ... args) {
    for (int i=0; i<args.length; i++) {
        re = re + args[i].re;
        im = im + args[i].im;
    }
}
```

18. OBJETOS

- ***Declaración***

  <NombreClasse> <NombreObjeto>;

  <NombreObjeto> = new <NombreClase>(inicialización);

  ...

- ***Acceso a un atributo***

  NombreObjeto.atributo

- ***Llamada a un método de la clase de objeto***

  NombreObjeto.metodo(argumentos);

18. OBJETOS

- **Ejemplo:**

```
Complejo z, w;

z = new Complejo(-1.5, 3.0);
w = new Complejo(-1.2, 2.4);

z.suma(w);

System.out.println("Complejo: " + z.toString());
System.out.println("Modulo: " + z.modulo());
```

12. SENTENCIAS
13. LIBRERÍAS
14. ENTRADA Y SALIDA POR PANTALLA
15. ENTRADA Y SALIDA POR FICHEROS
16. PROGRAMA PRINCIPAL Y ARGUMENTOS
17. CLASES
18. OBJETOS
19. MÁS SOBRE CLASES Y OBJETOS
20. INTERFACES
21. APPLETS
22. EXCEPCIONES

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
  - Dos o más métodos con el **mismo nombre** dentro de la misma clase.
  - Se **diferencian** en los parámetros

```
class nombre_clase {
    public tipo_retorno nombre_método(parámetros) {
        código;
    }
    public tipo_retorno nombre_método(otros parámetros) {
        otro código;
    }
    ...
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
  - **Ejemplo (1/3)**

```
class Complejo {
    private double re, im;
    public Complejo(double r, double i) {
        re = r;
        im = i;
    }
    ...
```

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
  - **Ejemplo (2/3)**

```
...
    public Complejo sumar (Complejo c) {
        return new Complejo(re + c.re, im + c.im);
    }
    public Complejo sumar (double r, double i) {
        return new Complejo(re + r, im + i);
    }
    public String toString() {
        return re + " + " + im + "i";
    }
} // Fin de la clase Complejo
```

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
  - **Ejemplo (1/3)**

```
Complejo c1 = new Complejo(1, 3);
Complejo c2 = new Complejo(-4, 3.5);

c2 = c1.sumar(c2);
c2 = c2.sumar(0.5, -4);

System.out.println(c1 + "\n" + c2 + "\n");
```

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- CONSTRUCTORES
  - Se ejecuta automáticamente cuando se **declara** un objeto de una clase.
  - Tiene el **mismo nombre** que la **clase**.
  - Una clase puede tener más de un constructor (**sobrecarga**).
  - Por defecto,
    - toda clase tiene un **constructor sin parámetros** y sin código,
    - que desaparece una vez se escribe un método constructor para dicha clase.
  - Por defecto, en **las clases con herencia**
    - antes de ejecutarse todo constructor, llama al código del **constructor** sin parámetros de la **clase padre**.
    - Se puede cambiar escribiendo como primera línea de código del constructor: **super(parámetros**).

19. MÁS SOBRE CLASES Y OBJETOS:

- CONSTRUCTORES
  - **Sintaxis**

```
class nombre_clase {
    public nombre_clase(parámetros) {
        código;
    }
    public nombre_clase(otros parámetros) {
        código;
    }
    …
}
nombre_clase objeto =
        new nombre_clase(parámetros_constructor);
```

19. MÁS SOBRE CLASES Y OBJETOS:

- CONSTRUCTORES
  - **Ejemplo 1**

```
class Complejo {
    private double re, im;

    public Complejo(double r, double i) {
            re = r;
            im = i;
    }
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

- CONSTRUCTORES
  - Ejemplo 2: se *imprimirá por pantalla AB:*

```
class A {
    public A() { System.out.print("A"); }
}
class B extends A {
        public B() { System.out.print("B");}
}
class Principal {
        public static void main(String[] args) {
        B b = new B();
        }
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- THIS
  - Se utiliza como **referencia del objeto actual** que está ejecutando el método.
  - Es útil para
    - **diferenciar** los atributos de la clase de los parámetros, cuando éstos tengan el mismo nombre
    - y también cuando haya que llamar a un método **pasando como referencia** el **objeto actual** que está ejecutando el código.

19. MÁS SOBRE CLASES Y OBJETOS:

- THIS
  - **Ejemplo 1:**

```
class Complejo {
    private double re, im;
    public Complejo(double re, double im) {
      // this.re es el atributo, y re es el parámetro
      this.re = re;
    // this.im es el atributo, e im es el parámetro
      this.im = im;
    }
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

- THIS
  - **Ejemplo 2:**

```
class Actor {
    private Vector peliculas;
    public void incluirPelicula(Pelicula p) {
            peliculas.add(p);
            p.incluirActor(this);
    }
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

• HERENCIA (1/4)

- Una clase puede **heredar** o derivar de otra
  - Pasa a disponer automáticamente de todos los métodos y atributos de esta otra clase como si fueran propios.
  - La clase que hereda se llama "**hija**" y la clase de la cual hereda se llama "**padre**".
- Por defecto,
  - toda clase hereda de la clase **Object**,
  - a menos que especifiquemos que hereda de otra clase mediante la palabra ***extends***:

```
class clase_hija extends clase_padre {
    ...
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

- HERENCIA (2/4)
  - La clase hija puede **redefinir** los métodos heredados de la clase padre.
  - Un método de la clase hija puede llamar al código de un método de la clase padre mediante la palabra "**super**":

```
tipo_retorno metodo_clase_hija(argumentos) {
    ...
    ... super. metodo_clase_padre(argumentos) ...
    ...
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

- HERENCIA (3/4)
  - Un **objeto** de la clase **hija** es del tipo de la clase hija, pero también del
    - tipo de la clase **padre**
    - y del tipo de todos sus **antecesores**.
  - Operador lógico **instanceof**:
    - Permite preguntar por el tipo de un objeto

      objeto instanceof *nombre_clase*
  - Se puede cambiar el tipo de un objeto escribiendo previamente su nuevo tipo entre paréntesis:

    (nueva_clase) objeto

19. MÁS SOBRE CLASES Y OBJETOS:

- HERENCIA (4/4)
  - Una clase sólo puede heredar
    - de **otra clase**
    - y de varias **interfaces**.
  - En Java **no** existe la herencia múltiple.

19. MÁS SOBRE CLASES Y OBJETOS:

- Ejemplo de sobrecarga, constructor, herencia, this y super:

```
class Persona {
    private String nombre;
    private int edad;

    public Persona() {
    }

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
}
...
```

19. MÁS SOBRE CLASES Y OBJETOS:

- Ejemplo de sobrecarga, constructor, herencia, this y super:

```
...
public String toString() {
    return "Nombre: " + nombre + "\n" + "Edad: " + edad + "\n";
    }
}
class Alumno extends Persona {
    private double nota;
    public Alumno(String nombre, int edad, double nota) {
            super(nombre, edad);
            this.nota = nota;
    }
...
```

19. MÁS SOBRE CLASES Y OBJETOS:

- Ejemplo de sobrecarga, constructor, herencia, this y super:

```
...
public String toString() {
    return super.toString() + "Nota: " + nota + "\n";
    }
} // Fin de la clase Persona
Persona p1 = new Persona();
Persona p2 = new Persona("Alex", 22);
Alumno a1 = new Alumno("Pepe", 20, 8.5);
System.out.println(p1 + "\n" + p2 + "\n" + a1);
System.out.println(p2 instanceof Alumno);
System.out.println(a1 instanceof Persona);
System.out.println((Persona) a1);
```

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- VISIBILIDAD

  - ***public***
    - La accesibilidad de los métodos y atributos de una clase son accesibles para **cualquier** clase

  - ***private***
    - los métodos y atributos **sólo** son accesibles para **la clase** que los ha declarado

  - ***protected***
    - los métodos y atributos son accesibles para la **clase** que los ha declarado y para sus clases **hijas o descendientes**

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- FINAL
  - Se utiliza para
    - declarar una **constante**, cuando lo encontramos delante de un atributo

      final ***double*** *PI = 3.141592;*

    - un **método que no se podrá redefinir**, cuando lo encontramos delante de un método

      final bool *par () { … }*

    - o **una clase de la que ya no se podrá heredar**, cuando lo encontramos delante de una clase.

      final class *Estudiante{ … }*

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- ABSTRACT

  - Denota un método que **no** va a tener código.

  - Las clases con métodos abstractos
    - **no se pueden instanciar**
    - y sus **clases herederas deberán escribir el código de sus métodos abstractos** si se quiere crear alguna instancia suya.

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- STATIC

  - Se aplica a los atributos y métodos de una clase que pueden utilizarse **sin crear un objeto** que instancie dicha clase.

  - El **valor** de un **atributo estático** es **compartido** por todos los objetos de dicha clase.

19. MÁS SOBRE CLASES Y OBJETOS:

- STATIC
  - **Ejemplo (1/2)**

```
class Persona {
    private String nombre;
    public static int num = 0;
    public Persona(String nombre) {
        this.nombre = nombre;
        num++;
    }
    public static int cuantos() {
      return num;
    }
...
```

19. MÁS SOBRE CLASES Y OBJETOS:

- STATIC
  - **Ejemplo (2/2)**

```
public void finalize() throws Throwable {
    num--;
    super.finalize();
    }
} // Fin de la clase Persona
class C {
    public static void main(String[] args) {
        Persona p1 = new Persona("Maria");
        Persona p2 = new Persona("Alex");
        System.out.println(Persona.cuantos());
    }
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- EQUALS Y CLONE
  - **Variables simples**
    - El **nombre** de una variable de tipo simple indica la **dirección de memoria** que contiene el valor de la variable (referencia directa).

      int *a, b;*

    - **Operador "=="**: permite comparar valores de variables simples

      ... (a == 3) ...

      ... (a == b) ...

    - **Operador "="**: permite asignar un valor a una variable simple

      a = 3;

      a = b;

19. MÁS SOBRE CLASES Y OBJETOS:

• EQUALS Y CLONE

o **Objetos**

- El **nombre** de un objeto
  - ❑ no contiene los valores de los atributos,
  - ❑ sino la **posición de memoria** donde residen dichos valores de los atributos (referencia indirecta)

  Complejo *w, z;*

- **Operador "==":** permite comparar si dos objetos ocupan la misma posición (**comprueba si son el mismo objeto**)

  ... (w == z) ...

- **Método "equals":** permite comprobar si dos objetos poseen atributos con los mismos valores (**compara los contenidos**)

  w.equals (z)

19. MÁS SOBRE CLASES Y OBJETOS:

- EQUALS Y CLONE
  - **Objetos**
    - **Operador "="**: "asigna" un objeto a otro objeto que ya existe (serán el mismo objeto)

```
Complejo w, z;
w = z;
```

A partir de este instante, "***w***" ocupa la misma posición de memoria que "***z***".

- Método ***clone***: crea una copia de un objeto determinado y la asigna a otro.

```
Complejo w = (Complejo) z.clone ();
```

19. MÁS SOBRE CLASES Y OBJETOS:

- EQUALS Y CLONE
  - **Ejemplo (1/2)**

```
class Student {
    public String name;
    public double test1, test2, test3;

    public double getAverage() {
      return (test1 + test2 + test3) / 3.0;
    }
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

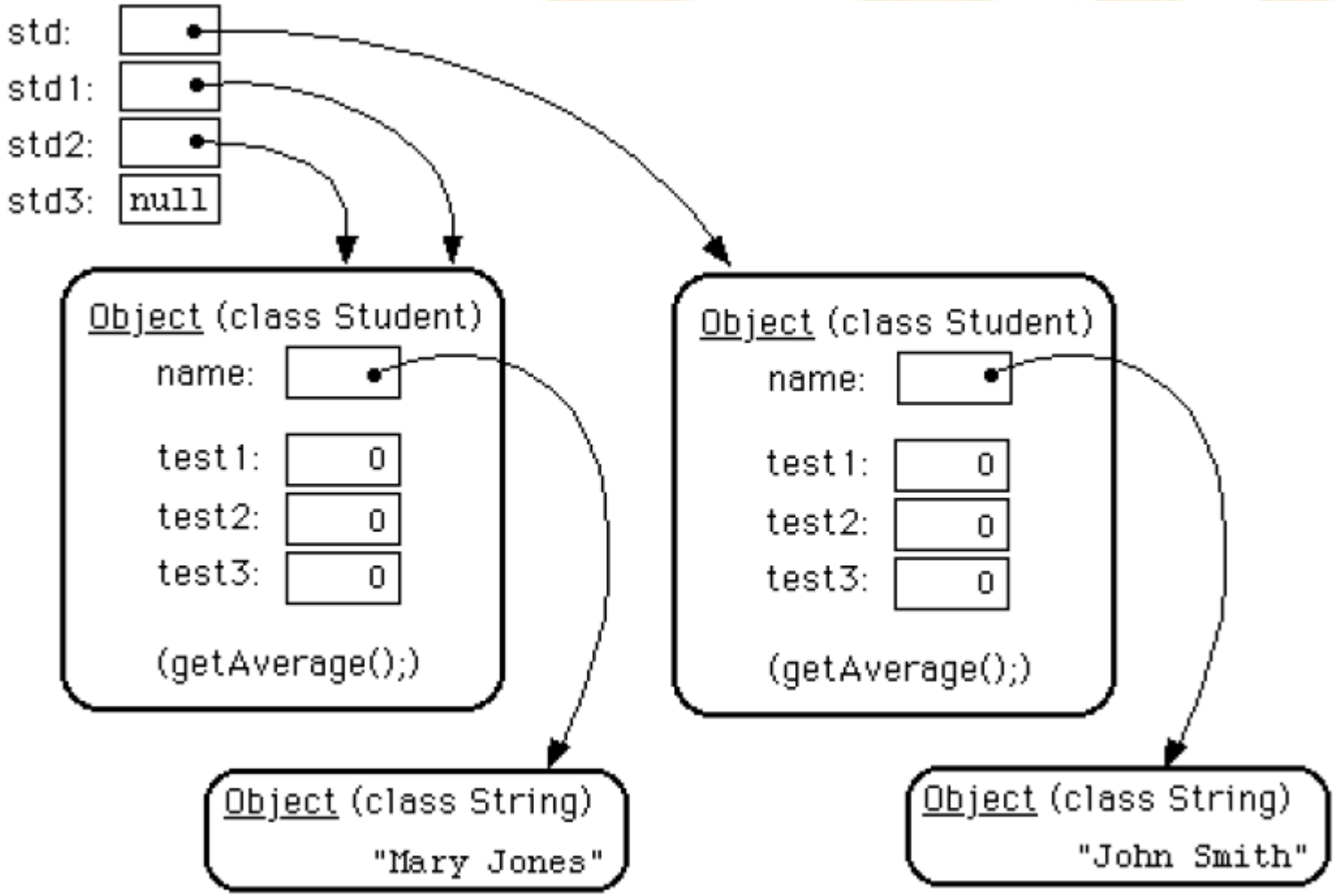
• EQUALS Y CLONE

o **Ejemplo (2/2)**

```
Student std, std1, std2, std3;

std = new Student();
std1 = new Student();

std2 = std1;
std3 = null;

std.name = "John Smith";
std1.name = "Mary Jones";
```

19. MÁS SOBRE CLASES Y OBJETOS:

- EQUALS Y CLONE



19. MÁS SOBRE CLASES Y OBJETOS:

- SOBRECARGA
- CONSTRUCTORES
- THIS
- HERENCIA
- SUPER
- INSTANCEOF
- CONVERSIÓN
- VISIBILIDAD
- FINAL
- ABSTRACT
- STATIC
- EQUALS Y CLONE
- POLIMORFISMO

19. MÁS SOBRE CLASES Y OBJETOS:

- POLIMORFISMO

  - Se puede declarar un objeto de una clase,
    - pero instanciarlo como un **descendiente** de dicha clase
    - lo contrario no es posible

clase_padre objeto =

new *clase_descendiente(parámetros_constructor*)**;**

19. MÁS SOBRE CLASES Y OBJETOS:

- POLIMORFISMO
  - **Ejemplo (1/4)**

```
class Complex {
    double re, im;
    public Complex(double re, double im) {
      this.re = re;
      this.im = im;
    }
// Lo siguiente también podría ser "abstract public void imprimir();"
    public void imprimir() {
      System.out.println(re + " " + im);
    };
} // Fin de la clase Persona
```

19. MÁS SOBRE CLASES Y OBJETOS:

• POLIMORFISMO

o Ejemplo (2/4)

```
class Complex1 extends Complex {
    public Complex1(double re, double im) {
            super(re, im);
    }
    public void imprimir() {
            System.out.println(re + "+" + im + "i");
    }
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

- POLIMORFISMO
  - **Ejemplo (3/4)**

```
class Complex2 extends Complex {
    public Complex2(double re, double im) {
            super(re, im);
    }

    public void imprimir() {
            System.out.println("(" + re + "," + im + ")");
    }
}
```

19. MÁS SOBRE CLASES Y OBJETOS:

• POLIMORFISMO

o Ejemplo (4/4)

```
class Principal {
    public static void main(String[] args) {
      Complex v[] = new Complex[2];
      v[0] = new Complex1(5, 4);
      v[1] = new Complex2(1, 3);

      for (int i = 0; i < v.length; i++) {
            v[i].imprimir();
      }
    }
}
```

20. INTERFACES

- **Interfaz: clase completamente abstracta.**
  - **Ninguno** de sus métodos tiene código.
  - Puede incluir atributos "**constantes**"
    - Todos los atributos de una interface son públicos y estáticos
    - Por tanto es **redundante** escribir ***public static***

- **Java no posee herencia múltiple,**
  - Pero una clase puede implementar una o más interfaces, debiendo **definir** sus métodos.

20. INTERFACES

- **Sintaxis**

interface *<Nombre_interface> [**extends** < Nombre_interface_padre>] {*

*[**public static**] **final int** CONSTANTE 1 = valor 1;*

*[**public static**] **final int** CONSTANTE 2 = valor 2;*

…

visibilidad [modificadores] tipo metodo1(argumentos);

visibilidad [modificadores] tipo metodo2(argumentos);

. . .

}

class *<Nombre_clase>* ***extends*** *<clase_padre>* ***implements*** *<interface1>, <interface2>, . . .*

{

. . .

}

20. INTERFACES

- **Ejemplo (1/7)**
  - Se desea crear una clase llamada **Vector**:
    - sus **elementos** pueden ser de **cualquier tipo**,
    - Sus elementos se han de poder
      - **ordenar** con un método de la clase Vector
      - **imprimir** con un método de la clase Vector

20. INTERFACES

- **Ejemplo (2/7)**

```
interface Comparable {
        int compareTo(Object o);
}

interface Imprimible {
        String toString();
}
```

20. INTERFACES

- **Ejemplo (3/7)**

```
class MiVector {
        Object elementos[];
        int num;
        public MiVector(int capacidad) {
            elementos = new Object[capacidad];
            num = 0;
        }
        public void añadir(Object o) {
          if (num < elementos.length)
                    elementos[num++] = o;
        }
...
```

20. INTERFACES

- **Ejemplo (4/7)**

```
...
public void ordenar() {
        Object aux;
        for (int i = 0; i < num-1; i++)
          for (int j = i+1; j < num; j++)
            if (((Comparable)elementos[i]).compareTo(elementos[j]) > 0) {
                    aux = elementos[i];
                    elementos[i] = elementos[j];
                    elementos[j] = aux;
        }
}
```

20. INTERFACES

- **Ejemplo (5/7)**

```
...
        public void imprimir() {
                for (int i = 0; i < num; i++)
                System.out.println((Imprimible)elementos[i]);
        }
} // Fin de la clase MiVector
```

20. INTERFACES

- **Ejemplo (6/7)**

```
class Persona {
        public String nom;
}
class Alumno extends Persona implements Comparable, Imprimible {
        public double nota;
        public int compareTo(Object o) {
                return nom.compareTo(((Alumno)o).nom);
        }
        public String toString() {
                return nom + " " + nota;
        }
}
```

20. INTERFACES

- **Ejemplo (7/7)**

```
class Principal {
    public static void main(String[] args) {
        MiVector v = new MiVector(5);
        Alumno a = new Alumno();
                a.nom = "Pepe";
                a.nota = 6.7;
                v.añadir(a);
        Alumno b = new Alumno();
                b.nom = "Pepa";
                b.nota = 7.6;
                v.añadir(b);
                v.ordenar();
                v.imprimir();
    }
}
```

21. APPLETS

- Definición de ***applet***
  - Programa diseñado para ser ejecutado **dentro** de otra aplicación, como, por ejemplo, un navegador web

21. APPLETS

- **Sintaxis (1/3)**

```
import java.awt.*;
import java.applet.*;
class <Nombre> extends Applet {
    // declaración de atributos
    visibilidad [modificadores] tipo atributo1 [= valor];
    visibilidad [modificadores] tipo atributo2 [= valor];
    ...
    // declaración de métodos
    visibilidad [modificadores] tipo metodo1(argumentos) {
        instrucciones;
    }
...
```

21. APPLETS

- **Sintaxis (2/3)**

```
...
// método de inicialización (principal)
public void init() {
    instrucciones;
}
// método de visualización (dibujar)
public void paint(Graphics g) {
    instrucciones;
}
...
```

21. APPLETS

- **Sintaxis (3/3)**

```
// método de activación (-> visible)
public void start() {
instrucciones;
}
// método de desactivación (-> invisible)
public void stop() {
    instrucciones;
}
// método de destrucción
public void destroy() {
    instrucciones;
}
... } // Fin de la declaración del "applet" <Nombre>
```

21. APPLETS

- **Inserción de un *applet* en una página web (HTML)**

```
<html>
<body>
    ...
<applet code="nombre_applet" atributos>
    <param nombre_parametro = "valor">
    <param nombre_parametro = "valor">
    ...
    código HTML si el navegador no puede ejecutar Java
</applet>
</body>
</html>
```

21. APPLETS

- **Ejemplo (1/4)**

```
import java.awt.*;
import java.applet.*;

public class PruebaApplet extends Applet
{
    Image logo;
    AudioClip melodia;
    TextField cuadroTexto;

...
```

21. APPLETS

- **Ejemplo (2/4)**

```
...
    public void init()
    {
        logo = getImage(getDocumentBase(), "imágenes\Logotipo.png");
        melodia = getAudioClip(getDocumentBase(), "Melodia.au");
        cuadroTexto = new TextField(20);
        cuadroTexto.setText("Aquí puedes escribir");
        add(cuadroTexto);
    }
    public void start()
    {
        melodia.loop();
    }
```

21. APPLETS

- **Ejemplo (3/4)**

```
public void paint(Graphics g)
{
    g.drawImage(logo, 0, 0, this);
    g.drawString(cuadroTexto.getText(), 25, 25);
}
public void stop()
{
    melodia.stop();
}
public void destroy()
{
 }
} // Fin de la declaración del applet
```

21. APPLETS

- **Ejemplo (4/4)**

HTML Applet:

```
<html>
<body>
    <applet code="PruebaApplet"
            width=300
            height=50
            align=middle>
    </applet>
</body>
</html>
```

12. SENTENCIAS
13. LIBRERÍAS
14. ENTRADA Y SALIDA POR PANTALLA
15. ENTRADA Y SALIDA POR FICHEROS
16. PROGRAMA PRINCIPAL Y ARGUMENTOS
17. CLASES
18. OBJETOS
19. MÁS SOBRE CLASES Y OBJETOS
20. INTERFACES
21. APPLETS
22. EXCEPCIONES

22. EXCEPCIONES

- Permiten **controlar** posibles situaciones de **error**

```
try {
    código donde se pueden producir excepciones
}
catch (TipoExcepcion1 NombreExcepcion) {
    Código a ejecutar si se produce una excepción del tipo TipoExcepcion1
}
catch (TipoExcepcion2 NombreExcepcion) {
    Código a ejecutar si se produce una excepción del tipo TipoExcepcion1
}
...
finally {
    Código a ejecutar tanto si se produce una excepción como si no
}
```

22. EXCEPCIONES

- **Ejemplo (1/2)**

```
String salario;
BufferedReader fichero1 = null;
BufferedWriter fichero2 = null;
try {
    fichero1 = new BufferedReader(new FileReader("salarios.txt"));
    fichero2 = new BufferedWriter(new FileWriter("salarios.new"));
    while ((salario = fichero1.readLine()) != null) {
     salario = (new Integer(Integer.parseInt(salario)*10).toString());
     fichero2.write(salario+"\n");
    }
}
```

22. EXCEPCIONES

- **Ejemplo (2/2)**

```
catch (IOException e) {
    System.err.println(e);
}
catch (NumberFormatException e) {
    System.err.println("No es un número");
}
finally {
    fichero1.close();
    fichero2.close();
}
```

22. EXCEPCIONES

- Java posee multitud de **excepciones** agrupadas por familias.
  - ArithmeticException
  - IOException
  - EOFException
  - FileNotFoundException
  - NullPointerException
  - NegativeArraySizeException
  - ArrayIndexOutOfBoundsException
  - SecurityException
  - NumberFormatException
  - ...

22. EXCEPCIONES

```
java.lang.Object
 └─ java.lang.Throwable
      ├─ java.lang.Exception
      │    ├─ java.lang.RuntimeException
      │    │    ├─ java.lang.IllegalArgumentException
      │    │    │    └─ java.lang.NumberFormatException
      │    │    ├─ java.lang.ArithmeticException
      │    │    └─ java.lang.IndexOutOfBoundsException
      │    │         ├─ java.lang.ArrayIndexOutOfBoundsException
      │    │         └─ java.lang.StringIndexOutOfBoundsException
      │    ├─ java.io.IOException
      │    │    ├─ java.io.EOFException
      │    │    ├─ java.io.FileNotFoundException
      │    │    ├─ java.net.MalformedURLException
      │    │    └─ java.net.SocketException
      │    │         ├─ java.net.PortUnreachableException
      │    │         ├─ java.net.NoRouteToHostException
      │    │         └─ ...
      │    └─ ...
      └─ java.lang.Error
           ├─ java.lang.VirtualMachineError
           │    ├─ java.lang.OutOfMemoryError
           │    ├─ java.lang.UnknownError
           │    └─ ...
           └─ ...
```

22. EXCEPCIONES

- Java posee multitud de "**errores**"
  - **Fallos de la máquina virtual** que es mejor que no los gestione la aplicación.
  - Por ejemplo:
    - OutOfMemoryError
    - InternalError
    - StackOverflowError
    - UnknownError
    - NoClassDefFoundError,
    - ...

22. EXCEPCIONES

- **Creación de una nueva excepción**

public class *NombreNuevaExcepcion* ***extends*** *NombreExcepcion {*

atributos y métodos

}

- Para declarar que un **método** que **lanza excepciones**:

visibilidad [modificadores] tipo *método(argumentos)* ***throws*** *NombreExcepcion1,*

NombreExcepcion2, ... {

...

... ***throw new*** *NombreExcepcion1(parámetros);*

...

... throw new NombreExcepcion2(parámetros);

...

}

22. EXCEPCIONES

- **Ejemplo (1/4)**

```
class CoeficientZeroException extends ArithmeticException {
    public CoeficientZeroException(String mensaje) {
            super(mensaje);
    }
};
```

22. EXCEPCIONES

- **Ejemplo (2/4)**

```
// Clase que representa una ecuación de primer grado: a x + b = 0
class Ecuacion {
    private double a, b;
    public Ecuacion(double coef1, double coef0) {
                        a = coef1;
                        b = coef0;
    }
  public double Raiz() throws CoeficientZeroException {
     if (a == 0)
        throw new CoeficientZeroException("La ecuación no es de primer grado");
    else return -b/a;
    }
}
```

22. EXCEPCIONES

- **Ejemplo (3/4)**

```
class Principal {
    public static void main(String[] args) throws Exception {
      try {
      BufferedReader in =
                    new BufferedReader(new InputStreamReader(System.in));
    System.out.print("Introduzca el coeficiente del término de grado 1:");
    double c1 = Integer.parseInt(in.readLine());
    System.out.print("Introduzca el coeficiente del término de grado 0:");
    double c2 = Integer.parseInt(in.readLine());
    Ecuacion eq = new Ecuacion(c1, c2);
    System.out.println("La solución es : " + eq.Raiz());
}
```

22. EXCEPCIONES

- **Ejemplo (4/4)**

```
catch (CoeficientZeroException e) {
            System.err.println(e);
}
catch (NumberFormatException e) {
      System.err.println("Número incorrecto ... " + e.getMessage());
}
catch (Exception e) {
  System.err.println("Excepción desconocida");
  throw e;
}
  }
}
```

Muchas gracias