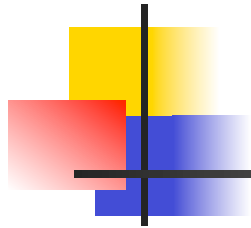


Introducción a Java



Indice

- La filosofía de Java
- El lenguaje Java
- Clases y objetos
- Uso del Java SDK
- Librerías (*packages*)
- Herencia
- Tratamiento de errores
- El API de Java

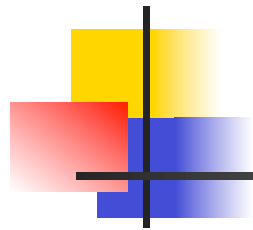


1. La filosofía de Java



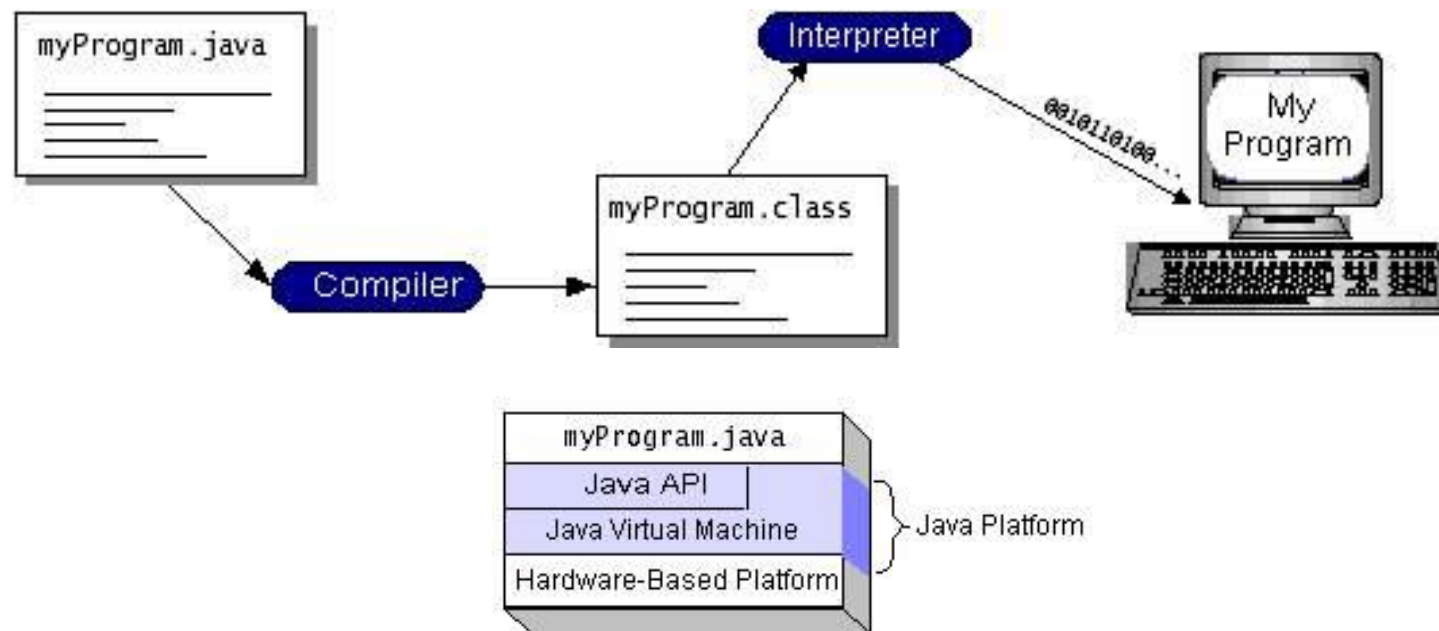
Ventajas de Java

- Multiplataforma
- Orientado a objetos
- Integrado con la web (*Applets*)
- Extensa API
 - GUI, redes, threads, B.D., 3D, ...
- Seguridad en aplicaciones cliente-servidor



La plataforma Java

- Compilador (.java \Rightarrow .class)
- Intérprete (máquina virtual Java)



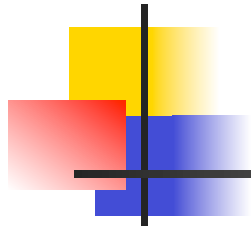


Software de desarrollo

- Gratuito
 - Sun SDK - línea de comandos
 - FORTE - IDE
 - JCreator
 -
- De pago
 - J++, VisualJ++ (Microsoft)
 - Café (Symantec)

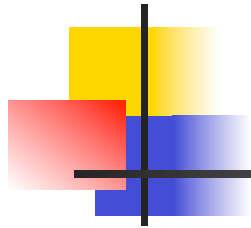


2. El lenguaje Java



Tipos de datos

- Primitivos (paso por valor)
 - boolean
 - char (16 bits)
 - byte
 - short (16 bits)
 - int (32 bits)
 - long (64 bits)
 - float (32 bits)
 - double (64 bits)
- Objetos (paso por referencia)

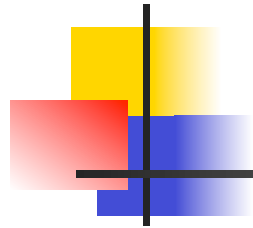


Sentencias

- Misma sintaxis que en C
 - if
 - for
 - while



3. Clases y objetos



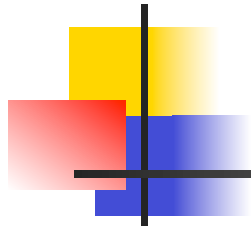
Java = C--

- No se pueden definir variables o métodos fuera de las clases
- No se permite herencia múltiple
- No existen punteros
- Los objetos se destruyen con un recolector de basura



Definir una clase

```
class Circulo {  
    Var. miembro {  
        float centroX, centroY;  
        float radio;  
    }  
    Constructores {  
        Circulo() {}  
        Circulo(float cx, float cy, float r) {  
            centroX = cx; centroY = cy; radio = r;  
        }  
    }  
    Método {  
        float area() {  
            return 3.141592*radio*radio;  
        }  
    }  
}
```



Trabajar con objetos

- Creación

```
Circulo c;  
c = new Circulo(0,0,1);
```

- Llamada a métodos

```
float area = c.area();
```

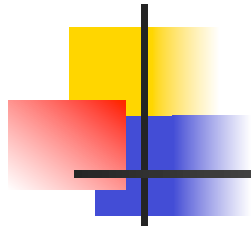
- Destrucción: recolector de basura



Variables y métodos de clase

```
class Circulo {  
    static int anchoLinea=1;  
    static final float PI=3.141592;  
    ...  
    static void ponAnchoLinea(int ancho) {  
        anchoLinea = ancho;  
    }  
}
```

```
System.out.println("PI vale" + Circulo.PI());  
Circulo.ponAnchoLinea(3);
```



Objetos: paso por referencia

- Asignación: apuntar al mismo objeto

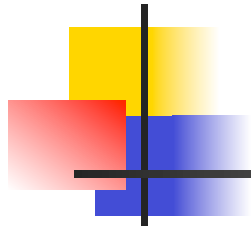
```
Circulo c1 = new Circulo(0,0,10);
```

```
Circulo c2 = c1;
```

```
c1.ponRadio(15);
```

```
System.out.println(c2.radio);
```

```
/*;;15!!*/
```



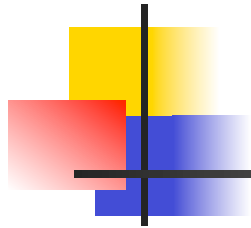
Objetos: paso por referencia

- Operador `==` \Rightarrow misma referencia

```
Circulo c1 = new Circulo(0,0,10);  
Circulo c2 = new Circulo(0,0,10);  
if (c1==c2)                /*false*/  
...  
...
```



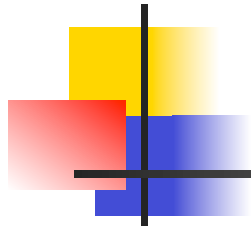

4. uso del Java *sdk*



Herramientas del Java SDK

- **Javac:** compilador
- **Java:** intérprete
- **Jdb:** depurador
- **Appletviewer:** visor de applets
- **Javadoc:** generador de documentación
- **Jar:** gestor de "**J**ava **A**rchives"

...



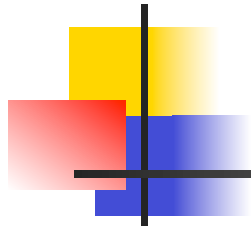
Fichero Circulo.java

```
class Circulo {  
    float centroX, centroY;  
    float radio;  
    Circulo() {}  
    Circulo(float cx, float cy, float r) {  
        centroX = cx; centroY = cy; radio = r;  
    }  
  
    float area() {  
        return (float) (3.141592*radio*radio);  
    }  
}
```



Fichero MiPrograma.java

```
class MiPrograma {  
    public static void main(String[] args) {  
        Circulo c;  
  
        c = new Circulo(0,0,1);  
        System.out.println("El área es "+c.area());  
    }  
}
```



Puntos importantes

- Cada **.java** debe definir solo una clase (pública)
- El nombre de la clase debe ser igual que el del fichero
- La clase “principal” debe definir un método **main**
- Las clases referenciadas deben estar en el CLASSPATH

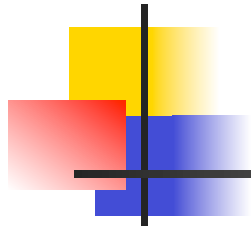


La variable CLASSPATH

- Le sirve al compilador y al intérprete para saber **dónde buscar las clases**
- Si está en un fichero .class, *añadir directorio al CLASSPATH*
- Si está en un .jar, *añadir fichero.jar al CLASSPATH*



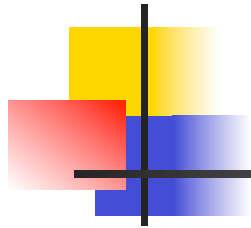
5. Librerías Java (*packages*)



Paquetes

- **Package** (librería): conjunto de ficheros *.class* que están en el mismo directorio, o fichero *.JAR*
- Todos los ficheros *.java* del *package* deben comenzar por

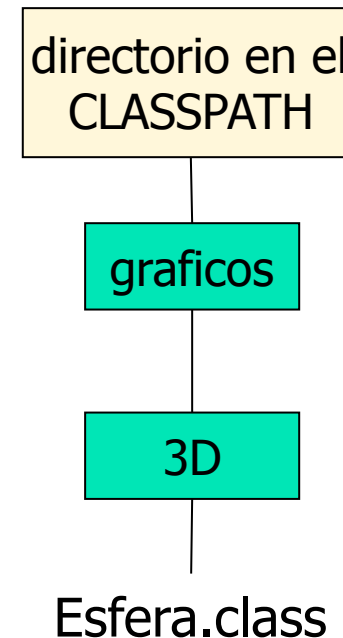
```
package nombrePaquete;
```

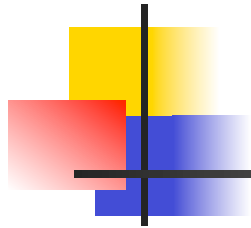



Nombres de paquetes

- Estilo DNS, deben reflejar la estructura de directorios en la que residen los *.class*

```
/*fichero Esfera.java */  
package graficos.3D;  
class Esfera {  
...  
}
```





Referenciar un paquete

- import

```
import graficos.3D.*;           /* todo el paquete */  
import graficos.3D.esfera;     /* solo una clase */
```

- Nombre completo del objeto o método

```
graficos.3D.esfera.dibujar();
```



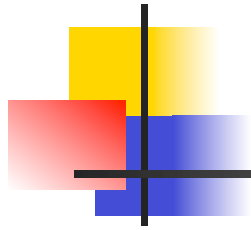
6. Herencia



Clases

■ Definición de clases

```
public class rectangulo {  
    public double x, y, ancho, alto;  
    public void mover (int a, int b) {  
        x=a; y=b;  
    }  
    public void redimensionar (int a, int b) {  
        ancho = a; alto = b;  
    }  
    public double area () {return x*y}
```



Herencia

- Subclases:

```
public class rectanguloColoreado
    extends rectangulo
{
    Color relleno;
    public void nuevoColor(Color nuevo)
    {
        relleno = nuevo;
    }
}
```

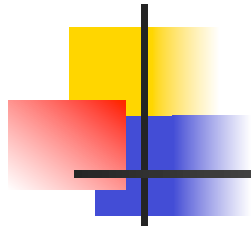


Herencia (2)

■ Redefinición:

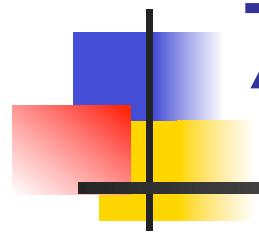
```
class superclase {  
    double numero = 0.0;  
    double f() { return numero };  
}
```

```
class subclase extends superclase {  
    double f() { return (super.f() + 3.0);  
}
```

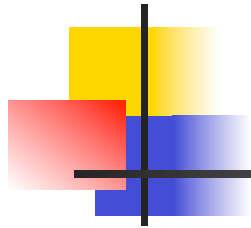


Encapsulamiento

- **Ámbitos:** clase -> sub-clase -> package -> todos.
- **private:** clase (no heredable)
- por defecto: mismo package
- **protected:** hasta package (no en todos)
- **public:** en todos

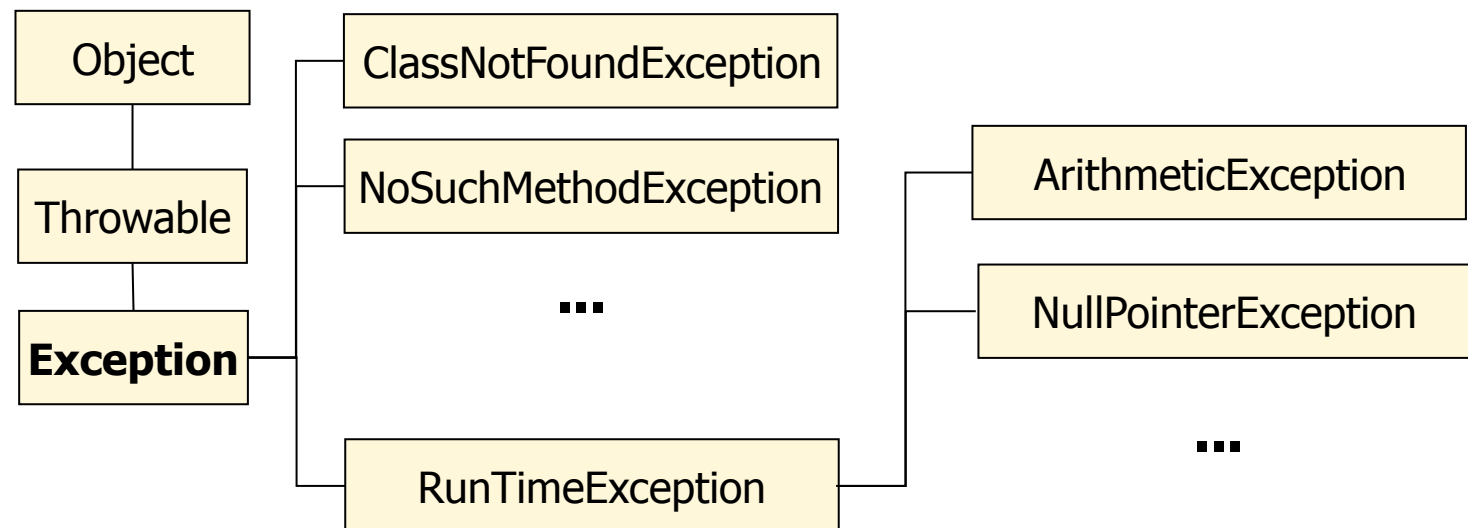


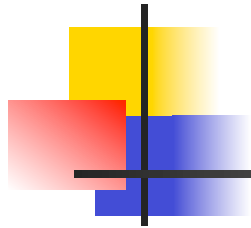
7. Tratamiento de errores



Excepciones

- Eventos que interrumpen el flujo de ejecución
- Son objetos

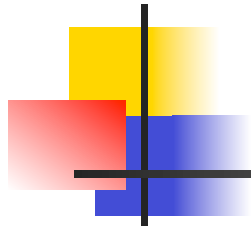




Try/catch

- Try: bloque en el que se puede producir un error
- Catch: manejador de un tipo de excepción

```
try {  
    ...  
    z = x/y;  
    ...  
}  
catch (ArithmeticException e) {  
    System.out.println("error: "+ e);  
}
```



¿Qué hacer con una excepción?

- Posibilidades
 - Ignorarla
 - Capturarla (**catch**)
 - Pasarla al nivel superior (**throw**)
- El tratamiento difiere según el tipo
 - Exc. *comprobadas*: **catch** o **throw**
 - Exc. *sin comprobar*: cualquier posibilidad



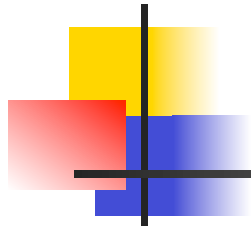
Propagación de excepciones

■ "Pasando la pelota"

```
abrirConexion() throws IOException  
{  
    abrirSocket();  
    enviarPetición();  
    recibirRespuesta();  
}
```

```
enviarPetición() throws IOException {  
    escribirCabecera();  
    escribirCuerpo();  
}
```

```
obtenerContenido() {  
    try {  
        ← abrirConexion();  
        leerDatos();  
    }  
    catch (Exception e)
```



Excepciones comprobadas

- Cuando llamamos a un método que puede generar una excepción comprobada hay que capturarla o lanzarla

```
miMetodo() {  
    /* el siguiente constructor puede generar una  
    excepción comprobada de tipo FileNotFoundException */  
    FileReader f=new FileReader("datos.txt");  
}
```

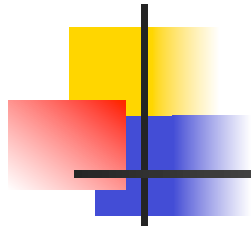
- Para especificar que la excepción se lanza

```
miMetodo() throws FileNotFoundException;
```



Excepciones sin comprobar

- Errores graves en tiempo de ejecución
 - Salirse de un array
 - Referencia nula
 - Dividir por 0
 - ...
- Si se ignoran se muestra un mensaje de error apropiado y el programa aborta



Excepciones propias

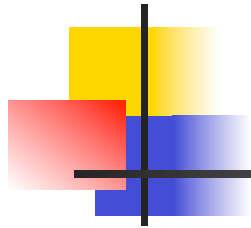
```
class miExcepcion extends Exception {  
    miExcepcion(String mens) {  
        super(mens);  
    }  
}
```

...

```
throw new miExcepcion("La cosa está muy malita");
```

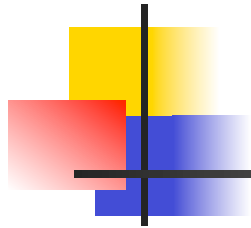


8. El API básico de Java



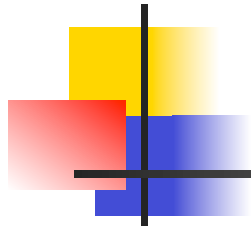
Algunos paquetes

- **java.lang:** clases básicas del lenguaje
- **java.io:** entrada/salida
- **java.util:** utilidades, colecciones (vectores,...)
- **java.net:** sockets y URLs
- **java.applet**



La clase `java.lang.Math`

- Variables y métodos estáticos
 - `Math.PI`
 - `Math.cos`, `Math.sin`, `Math.tan`
 - `Math.sqrt`
 - `Math.max(a,b)`, `Math.min(a,b)`
 - `Math.pow(a,b)`



Entrada/Salida (java.io.*)

- Clasificación de las operaciones de E/S
 - Tipo de operación: entrada, salida
 - Fuente/destino: memoria, cadena, fichero, *pipe*,...
 - Funcionalidad: *buffer*, e/s de datos,...
- ¿Una clase para cada caso?, ¡idemasiadas!!



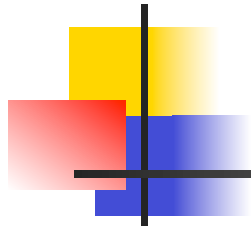
Filosofía de las clases de E/S

- Clases que encapsulan tipo de fuente/destino

```
FileInputStream f = new FileInputStream("datos.txt")
```

- Clases que **envuelven** a las anteriores y encapsulan funcionalidad

```
BufferedReader b = new BufferedReader(f);
```



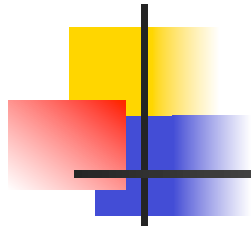
Filosofía de clases de E/S (II)

- Dos jerarquías de clases
 - Orientada a **bytes**: `xxxInputStream`, `xxxOutputStream`
 - Orientada a **caracteres**: `xxxReader`, `xxxWriter`



Ejemplo de E/S (I)

```
import java.io.*;
public class EjemploES {
    public static void main(String[] args) throws IOException{
        String mens="hola\nque tal andamos";
        String lin;
        BufferedReader en=new BufferedReader(new StringReader(mens));
        PrintWriter sal=new PrintWriter(new FileWriter("sal.txt"));
        lin = en.readLine();
        while (lin!=null) {
            sal.println(lin);
            lin = en.readLine();
        }
        sal.close();
    }
}
```



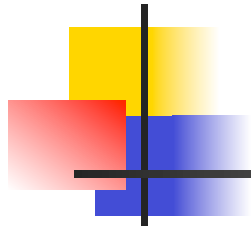
Arrays

- Arrays (declaración):

```
double lecturas[];  
double[] lecturas;  
lecturas = new double[5];  
  
double lecturas = new double[5];
```

- Arrays 2D (declaración):

```
double lecturas[][] = new double[3][];  
lecturas[0] = new double[1];  
lecturas[1] = new double[3];  
lecturas[2] = new double[5];
```

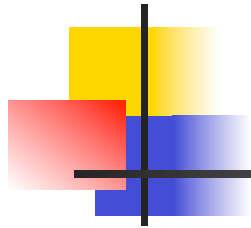


Arrays (2)

- Arrays en funciones:

```
int minimo(int[] lecturas, inicio)
{
    int imin;
    for (int i = inicio; i < lecturas.length; i++)
        if (lecturas[i] < lecturas[imin]) imin = i;
    return imin;
}
```

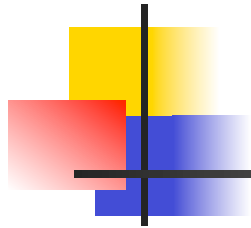
```
min = minimo(lecturas, 2);
```

Arrays (3)

- **Arrays heterogéneos:**

```
public class Test extends Applet {  
    private Component[] c = new Component[3];  
  
    public void init() {  
        c[0] = new Button("boton");  
        c[1] = new Label("etiqueta");  
        c[2] = new TextField("texto");  
        for (int i=0; i < 3; i++) this.add(c[i]);  
    }  
}
```



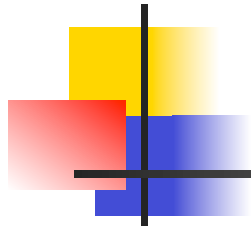
Strings

■ Constructores:

```
String s = "Esto es una cadena";  
String s = "a" + "b" + "c";  
char[] caracteres = {'C', 'a', 'd', 'e', 'n', 'a'};  
String s = new String(caracteres);
```

■ Acceso y comparación:

```
s = "alfabeto"; s.charAt(2) => 'f'  
t = "alfas"; t.compareTo(s) => -17  
s.endsWith("beto") => true   s.startsWith("a") => true  
s.equalsIgnoreCase("ALFABETO") => true  
s.indexOf('f') => 2   s.length() => 8
```



Strings (2)

■ Modificadores:

```
String s = "Java";  
s.concat("doc") => "Javadoc"  
s.replace('v', 'b') => "Jaba"  
s.substring(2) => "va"    s.substring(1, 3) => "av"  
s.toLowerCase() => "java"  s.toUpperCase() => "JAVA"
```

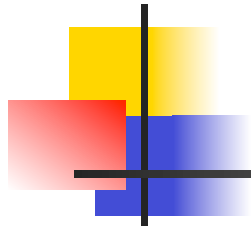
■ Strings para conversión:

```
Point p = new Point(10, 30);  
display.setText(p.toString());  
double d = 266.09; boolean b = (d > 0.0)  
display.setText(String.valueOf(b) + String.valueOf  
    (d));
```



ArrayList

- Array dinámico
- Declaración:
 - `ArrayList nombreDelArray;`
 - `ArrayList<tipoDeDato> nombreDelArray;`
 - Ejemplos:
 - `ArrayList<String> nombres;`
 - `ArrayList<Integer> edades;`



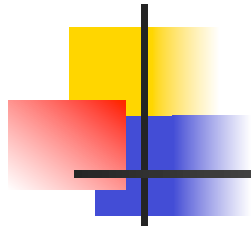
ArrayList(2)

- Creación:
 - nombreDelArray = new ArrayList();
- Añadir elementos al final
 - nombreDelArray.add(Objeto);
 - nombres.add("Luis");
 - edades.add(22);



ArrayList(3)

- Insertar elementos en una posición
 - nombreDelArray.add(posicion, objeto);
 - nombres.add(1, "Luis");
 - Si la posición no existe se produce una excepción (IndexOutOfBoundsException).
- Suprimir elementos
 - nombreDelArray.remove(posicion)



ArrayList(4)

- Otros métodos:
 - nombreDelArray.get(posicion);
 - nombreDelArray.set(posicion, objeto);
 - nombreDelArray.indexOf(objeto);