Algoritmos y Estructuras de Datos
Cursada 2025

Prof. Alejandra Schiavoni (ales@info.unlp.edu.ar)
Prof. Catalina Mostaccio (catty@lifia.info.unlp.edu.ar)
Prof. Laura Fava (lfava@info.unlp.edu.ar)
Prof. Pablo Iuliano (piuliano@info.unlp.edu.ar)

Algoritmos y Estructuras de Datos
Cursada 2024

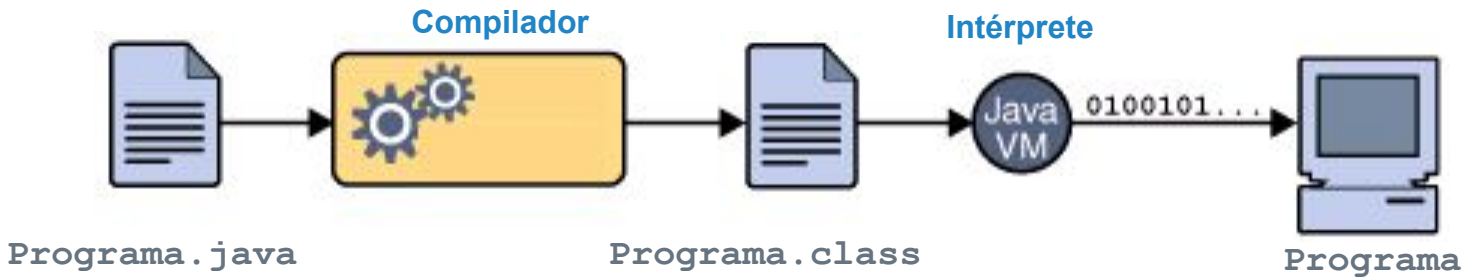
- Temas que abarca la asignatura
  - Estructuras de datos lineales: Listas, Pilas y Colas
  - Estructuras de datos no lineales: Arboles Binarios, Arboles Generales, Colas de prioridades, Grafos
  - Análisis de algoritmos: cálculo del tiempo de ejecución y orden de ejecución de algoritmos.
- Modalidad de teorías y prácticas
- Modalidad de aprobación
  - Parciales
  - Promoción

Repaso de conceptos básicos de JAVA

Clases e instancias

La Plataforma JAVA

En java, el código fuente es escrito en archivos con texto plano con extensión .java. Esos archivos son posteriormente compilados en archivos con extensión .class por el compilador java.
Un archivo con extensión .class no contiene código nativo/específico para un procesador determinado, sino que contiene bytecodes (el lenguaje de la máquina virtual de java).



El **javac.exe** es el compilador, viene con la plataforma.
El **java.exe** es un programa que viene con la plataforma java, que permite ejecutar los bytecodes, es el intérprete java.

Programación Orientada a Objetos

Los programas orientados a objetos están compuestos por varios **objetos**. Estos objetos se comunican entre ellos, mediante el envío de **mensajes**.

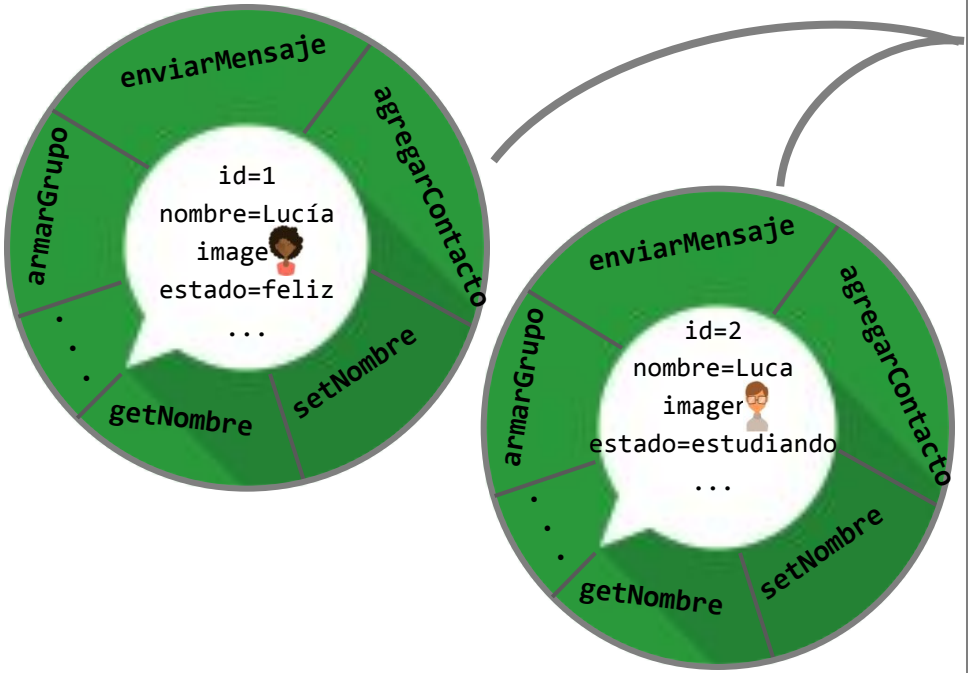
Cuando un programa se está ejecutando, los objetos necesarios se van creando en la memoria **HEAP.**

- Cada uno de estos objetos es una entidad de software que combina un **estado o datos** y **comportamiento o métodos**
- Estos objetos se crean a partir de un molde o **clase**.

¿Cómo declarar una clase en JAVA? (1/2)

- **Una clase java es un bloque de código o un molde, que describe el estado y el comportamiento que tendrán los objetos que con ella se creen.**
- Un archivo origen java debe tener como mínimo:
  - en la primera línea la palabra clave **package** seguida del nombre del paquete.
  - la palabra clave **class** seguida del nombre de la clase.
- Un archivo origen java debe guardarse con el mismo nombre que la clase (y con extensión **.java**). Se deben respetar las mayúsculas.

objetos de tipo **Contacto**



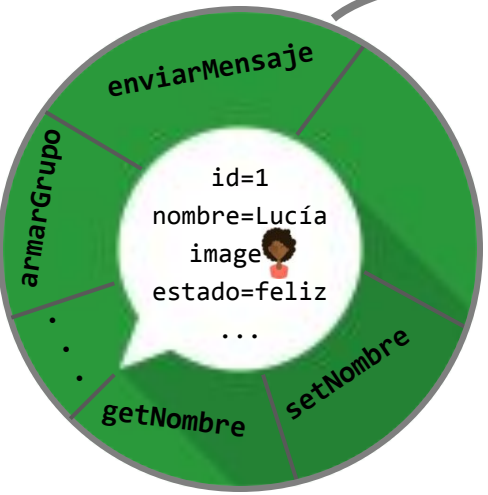
Contacto.java

```
package whatsapp;
import java.awt.Image;

public class Contacto {
 private String nombre;
 private Image imagen;
 private String estado;
 private int id;

 public String getNombre() {
  return nombre;
 }
 public void setNombre(String nombre) {
  this.nombre = nombre;
 }
 . . .
}
```

¿Cómo declarar una clase en JAVA? (2/2)

Para agregar estado y comportamiento debemos incluir:

- **variables de instancia**: constituyen el estado de un objeto. Normalmente, las variables de instancia se declaran **private**, lo que significa que se puede acceder a ellas directamente sólo desde la clase donde se definen.
- **métodos de instancia**: definen las operaciones que pueden realizar los objetos de un tipo de clase. Un método es un bloque de código, similar a lo que es una función o procedimiento en los lenguajes procedurales, como PASCAL.




```
package whatsapp;
import java.awt.Image;

public class Contacto {
```

Estado

```
 private String nombre;
 private Image imagen;
 private String estado;
 private int id;
```

Comportamiento

```
 public String getNombre() {
  return nombre;
 }
 public void setNombre(String nombre){
  this.nombre = nombre;
 }
 . . .
}
```

La declaración de una variables incluye:

- un **identificador o** nombre de la variable.
- un **tipo** de dato primitivo o referencial
- un especificador de acceso (opcional)

La declaración de un método especifica:

- un **nombre**
- una lista opcional de **argumentos**
- un **tipo** de retorno.
- un **modificador de acceso** (opcional)

Tipos de datos en Java

En **java** hay 2 categorías de tipos de datos: tipo primitivo y tipo referencial o de una clase particular.

- **Tipos primitivos**: las variables de tipo primitivo mantienen valores simples y NO son objetos. Existen 8 tipos de datos primitivos:

Entero: `byte, short, int, long`
Punto flotante: `float y double`
Un carácter de texto: `char`
Lógico: `boolean`

Declaración e inicialización de variables primitivas

```
float pi = 3.14;
double saldo = 0;
char letra = 'A';
int hora = 12;
boolean es_am = (hora>12);
```

`true/false`

- **Tipos de una clase:** las variables que referencian a un **objeto** son llamadas *variables referencias* y contienen la ubicación (dirección de memoria) de objetos en memoria.

Declaración e inicialización de variables referencias

```
Cliente cli;
cli = new Cliente();
```

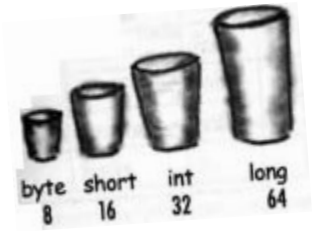
```
Fecha diaCumple = new Fecha();
```

Tipos de datos en Java

Inicialización

Si la definición de una clase, no inicializa variables de instancia, las mismas toman valores por defecto.

- Las variables de instancia de **tipo primitivo** se inicializan con los siguientes valores por defecto:



Tipo primitivo	Valor por defecto
`boolean`	`false`
`char`	`'\uoooo' (nulo)`
`byte/short/int/long`	`0`
`float/double`	`0.0`

- Las variables de instancia que son referencias a objetos, se inicializan con el valor por defecto: `null`.

Nota: las variables locales, es decir, las variables declaradas dentro de un método, deben inicializarse explícitamente antes de usarse.

Tipos de datos en Java

Clases *wrappers*

- Java no considera a los tipos de datos primitivos como objetos. Los datos numéricos, booleanos y de caracteres se tratan en su forma primitiva por razones de eficiencia.
- Java proporciona clases ***wrappers*** para manipular a los datos primitivos como objetos. Los datos primitivos están envueltos (*"wrapped"*) en un objeto que se crea en torno a ellos.
- Cada tipo de datos primitivo de Java, posee una clase *wrapper* correspondiente en el paquete **java.lang**. Cada objeto de la clase *wrapper* encapsula a un único valor primitivo.

Tipo primitivo	Clase Wrapper
`char`	`Character`
`boolean`	`Boolean`
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`

Las clases wrapper son inmutables (String también)

Tipos de datos en Java

Clases *wrappers*

Autoboxing

Es la conversión automática que realiza el compilador de Java entre los tipos primitivos y sus clases wrappers correspondientes. Por ejemplo, convertir un int en un Integer, un double en un Double, etc.

```
Character c = 'a';
Integer i = 7;
```

Unboxing

Es la conversión es al revés, es decir conversión de wrapper a un primitivo. Por ejemplo de un Character a char o Double a double.

```
char c1 = c;
int i1 = i;
```

Pensemos en un código que calcule la sumatoria de los números enteros desde 0 hasta el entero más grande. Sumemos primero el valor en un objeto de tipo Long y luego en un primitivo long. ***¿Cuál es el impacto?***

```
Long suma = 0;
long antes = System.currentTimeMillis();
for (int i = 0; i < Integer.MAX_VALUE; i++){
   suma += i;
}
long despues = System.currentTimeMillis();
```

tarda: 37.66 seg.

```
long suma = 0;
long antes = System.currentTimeMillis();
for (int i = 0; i < Integer.MAX_VALUE; i++){
    suma += i;
}
long despues= System.currentTimeMillis();
```

tarda: 5.526 seg.

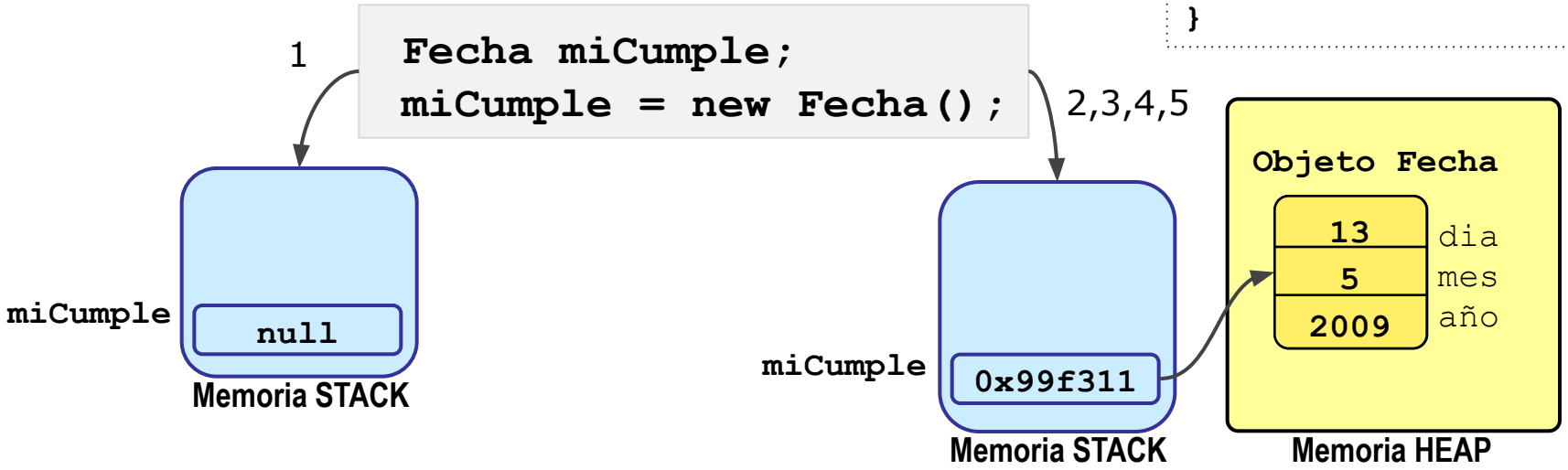
¿Cómo se crean instancias de una clase?

Para instanciar una clase, es decir, para crear un objeto de una clase, se usa el operador **new**.

La creación e inicialización de un objeto involucra los siguientes pasos:

1. Se aloca espacio para la variable
2. Se aloca espacio para el objeto en la HEAP y se inicializan los atributos con valores por defecto.
3. Se inicializan explícitamente los atributos del objeto.
4. Se ejecuta el **constructor** (*parecido* a un método que tienen el mismo nombre de la clase)
5. Se asigna la referencia del nuevo objeto a la variable.

```
public class Fecha {
 private int dia = 13;
 private int mes = 5;
 private int año = 2009;
 // métodos de instancia
}
```



¿Cómo se manipulan los datos de este objeto de tipo Fecha?

¿Qué son los Constructores?

Los constructores son piezas de código -sintácticamente similares a los métodos- que permiten definir un estado inicial específico de un objeto, en el momento de su creación.

Se diferencian de los métodos tradicionales porque:

- Deben tener el mismo nombre que la clase. La regla de que el nombre de los métodos debe comenzar con minúscula, no se aplica a los constructores.
- No retornan valor.
- Son invocados automáticamente.

```
public class Vehiculo {
 private String marca;
 private double precio;

 public Vehiculo() {
 }
}
```

NO retorna nada

La inicialización está garantizada: cuando un objeto es creado, se aloca almacenamiento en la memoria HEAP y se invoca al constructor.

```
Vehiculo v = new Vehiculo();
```

El operador **new()** se puede utilizar en cualquier lugar del código.

- La expresión **new** retorna una referencia al objeto creado recientemente, pero el constructor no retorna un valor.
- Java siempre llama automáticamente a un constructor cuando crea un objeto (antes de que el objeto sea usado). De esta forma la inicialización del objeto está garantizada.

Constructor sin argumentos

Un constructor sin argumento o constructor *Default*, es usado para crear un objeto básico.

- Si una clase NO tiene constructores, el compilador inserta automáticamente un constructor default, con cuerpo vacío:

```
public class Vehiculo {
  private String marca;
  private double precio;

  //métodos
}
```

Cuando se compila

```
public  Vehiculo(){
}
```

Cuando se crea un objeto de la clase Vehiculo, con `new Vehiculo()`, se invocará el constructor por defecto, aún cuando no se encuentre explícitamente en la clase.

- Si la clase tiene al menos un constructor, con o sin argumentos, el compilador NO insertará nada.

Constructores con argumentos

En general los constructores son usados para inicializar los valores del objeto que se está creando. *¿Cómo especificar los valores para la inicialización?* Los constructores pueden tener parámetros para la inicialización del objeto.

```
public class Vehiculo {
  private String marca;
  private double precio;

  public Vehiculo(String mar,double pre){
    marca = mar;
    precio = pre;
  }
}
```

```
public class Vehiculo {
  private String marca;
  private double precio;

  public Vehiculo(String marca,double precio){
     this.marca = marca;
     this.precio = precio;
  }
}
```

Codificaciones equivalentes

Si este constructor es el único de la clase, el compilador no permitirá crear un objeto **`Vehiculo`** de otra manera que no sea usando este constructor .

```
public class Automotores {
   public static void main(String[] args){
     Vehiculo auto1 = new Vehiculo("CITROEN", 13500.00);
     Vehiculo auto2 = new Vehiculo("HONDA",12400.50);
  }
}
```

Sobrecarga de Constructores

¿Qué pasa si se quiere construir un objeto `Vehiculo` de distintas maneras?

Se escriben en la clase más de un constructor ➡ sobrecarga de constructores.

```
public class Vehiculo {
 private String nroPatente="";
 private String propietario="SinDueño";

 public Vehiculo(){
 }

 public Vehiculo(String marca){
    this.marca = marca;
 }

 public Vehiculo(String marca,double precio){
  this.marca = marca;
  this.precio = precio;
 }
}
```

La **sobrecarga de métodos** permite que el mismo nombre de método sea usado con distintos tipos y cantidad de argumentos.

```
public class Botanico {
  public static void main(String[] args){
    Vehiculo a1=new Vehiculo();
    Vehiculo a2=new Vehiculo("HONDA");
    Vehiculo a3=new Vehiculo("HONDA",12300.50);
  }
}
```

Creación y uso de un objeto Contacto

Una vez que se ha creado un objeto, seguramente es para usarlo: cambiar su estado, obtener información o ejecutar alguna acción. Para poder hacerlo se necesita conocer la variable que lo referencia y utilizar la notación “.”

```
package whatsapp;
import java.awt.Image;

public class Contacto {
 private String nombre;
 private Image imagen;
 private String estado;
 private int id;

 public Contacto(){}
 public contacto (String n, String e){
   nombre=n;
   estado=e;
 }
 public String getNombre() {
  return nombre;
 }
 public void setNombre(String nombre) {
  this.nombre = nombre;
 }
 . . .
}
```

Instanciación de objetos Contacto e invocación de sus métodos

```
package whatsapp;

public class ContactoTest {

public static void main(String[] args) {
  Contacto c1 = new Contacto();
  c1.setNombre("Lucia");
  c1.setEstado(“feliz");
  Contacto c2 = new Contacto(“Juan”, “ocupado”);
  System.out.println("Se creó el contacto:"+ c1.getNombre());
  System.out.println("Se creó el contacto:"+ c2.getNombre());
 }
}
```

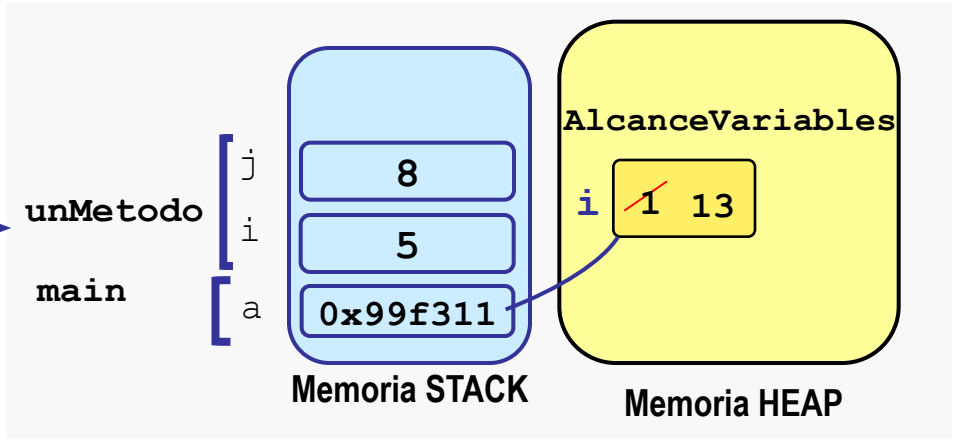
Se recomienda declarar todos los atributos privados (**private**) y utilizar métodos públicos (**public**) para acceder al estado.

Variables de instancia y variables locales

Las variables pueden declararse en dos lugares diferentes (siempre adentro de la clase):

- **afuera de cualquier método**. Son las variables de instancia que son creadas cuando el objeto es construido usando el **new()**. Estas variables existen, mientras exista el objeto.
- **adentro de cualquier método**. Estas variables son llamadas variables locales y deben inicializarse antes de ser usadas. Los parámetros de los métodos también son variables locales y las inicializan el código que llama al método. Estas variables son creadas cuando el método comienza a ejecutar y son destruidas cuando el método finaliza su ejecución.

```
public class AlcanceVariables {
 private int i=1;
 public void unMetodo(int i){
    int j=8;
    this.i=i+j;
 }
}
```




```
public class TestAlcanceVariables {
public static void main(String[] args) {
   AlcanceVariables a = new AlcanceVariables();
   a.unMetodo(5);
  }
 }
```

`this` es una referencia al objeto actual. Está disponible automáticamente en todos los métodos

Variables y métodos de clase

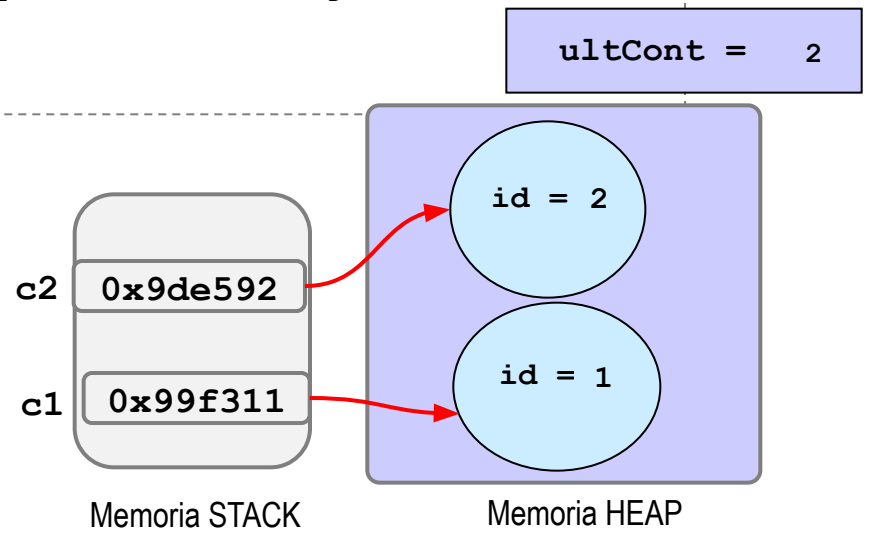
La palabra clave `static`

La palabra clave **`static`** declara atributos (variables) y métodos asociados con la clase en lugar de asociados a cada una de las instancias de la clase.

Las variables de clase son compartidas por todas las instancias de la clase. En el caso de los métodos de clase se utilizan cuando se necesita algún comportamiento que no depende de una instancia particular. En ambos casos se debe anteponer la palabra clave **`static`** al tipo de dato de la variable o de retorno del método.

```
public class Contacto {
 private static int ultCont = 0;
 private String nombre;
 private Image imagen;
 private String estado;
 private int id;

 public Contacto() {
  ultCont++;
  this.id = ultCont;
 }
 public static int getUltCont {
  return ultCont;
 }
 . . .
}
```

```
public class TestContacto{
 public static void main(String[] a){
   System.out.println(Contacto.getUltCont());
   Contacto c1 = new Contacto();
   System.out.println(Contacto.getUltCont());
   Contacto c2 = new Contacto();
   System.out.println(Contacto.getUltCont());
 }
}
```



`Contacto.getUltCont()` puede invocarse desde cualquier lugar aún no habiéndose creado instancias de Contacto

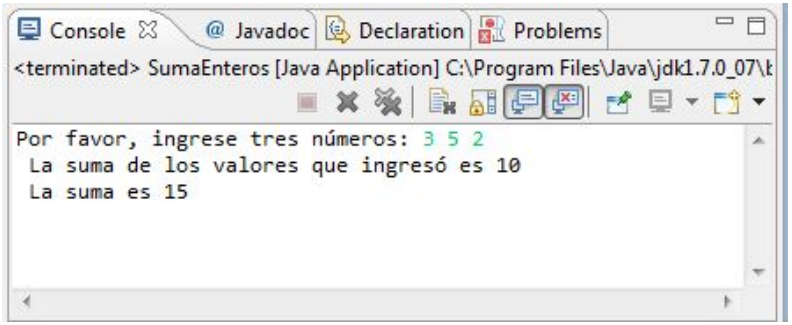
Variables y métodos de clase

La palabra clave `static`

Este es un ejemplo similar a lo pedido en la práctica donde una clase que define un conjunto de métodos de clase.

```
public class SumaEnteros {

public static int sumaValores() {
 Scanner consola = new Scanner(System.in);
 System.out.print("Por favor, ingrese tres números: ");
 int num1 = consola.nextInt();
 int num2 = consola.nextInt();
 int num3 = consola.nextInt();
 int suma = num1 + num2 + num3;
 return suma;
}

public static int sumaArreglo(int[] datos){
 int suma = 0;
 for (int i = 0; i < datos.length; i++) {
  suma = suma + datos[i];
 }
 return suma;
}
 . . .
}
```

Los métodos de clase solo tienen acceso a sus variables locales, parámetros y variables de clase y no tiene acceso a las variables de instancia.

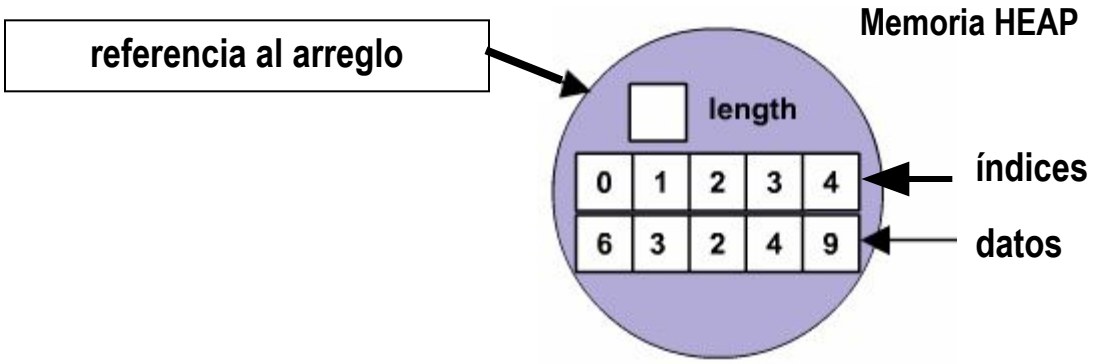



```
public class TestSumaEnterso {

 public static void main(String[] args) {
  System.out.println("La suma de los valores que
      ingresó es " + SumaEnteros.sumaValores());

  int[] datos = { 1, 2, 3, 4, 5 };
  int sumas = SumaEnteros.sumaArreglo(datos);
  System.out.println(" La suma es " + sumas);
 }
}
```

Arreglos

¿Cómo se manejan en JAVA?

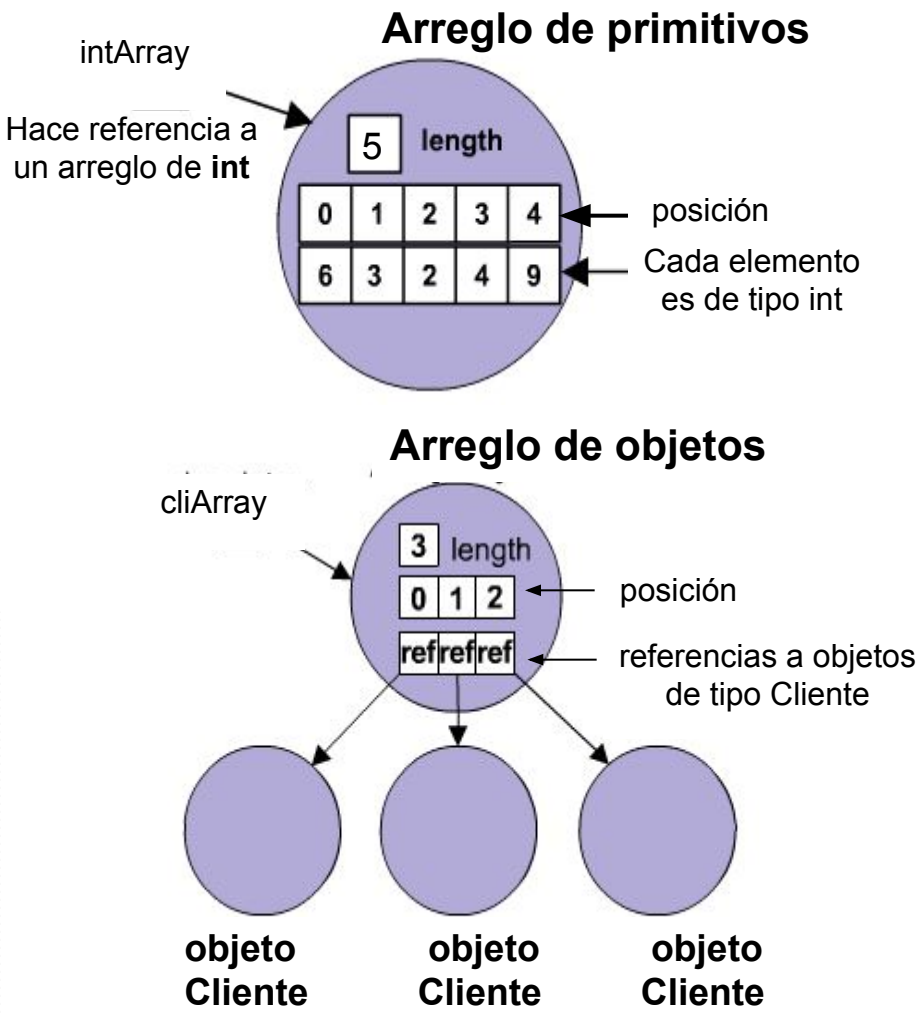
- Un arreglo **es un objeto que hace referencia a un conjunto** de valores primitivos u objetos, a través de una única variable.
- Los arreglos permiten manipular un conjunto de **valores del mismo tipo** de datos usando un único nombre.
- Los datos almacenados en un arreglo, se guardan en **posiciones contiguas**.
- Los arreglos tienen una cantidad fija de objetos o primitivos. Una vez creado la **dimensión no puede cambiar**.
- Cada arreglo mantiene una propiedad **length** con el tamaño del mismo. Al primer elemento del arreglo se le asigna el índice **0**, al siguiente elemento el índice **1**, etc.; por lo tanto, el índice del último elemento del arreglo es **length-1**.



Arreglos
de Tipos Primitivos y de Objetos

```
public class ArreglodePrimitivos {
public static void main(String[] args){
    int[] intArray = new int[5];
    intArray[0] = 6;
    intArray[1] = 3;
    intArray[2] = 2;
    intArray[3] = 4;
    intArray[4] = 9;
    }
}
```

```
public class ArreglodeClientes {
 public static void main(String[] args){
    Cliente[] cliArray = new Cliente[3];
    cliArray[0] = new Cliente();
    cliArray[1] = new Cliente();
    cliArray[2] = new Cliente();
 }
}
```



Arreglos

declaración e inicialización

La declaración, creación e inicialización de un arreglo pueden hacerse en 1 simple paso o en varios.

Varios Pasos

La **declaración** crea la variable arreglo, no el objeto arreglo. Para **crear** el arreglo se usa el operador **new.** Cuando se crea el objeto se debe indicar la cantidad de elementos o longitud del arreglo.

Finalmente, se lo **inicializa**, elemento por elemento. La variable **i** está disponible sólo en el bloque for.

```
int[] intArray;           // sólo declara
intArray = new int[5];    //crea el declarado arriba
double montos[]=new double[12]; //declara y crea
String[] items=new String[4];   //declara y crea
```

```
for (int i=0;i<4;i++){
  items[i] = "item"+i;
}
```

Un Paso

La **declaración** de la variable arreglo, la creación del arreglo y la inicialización, también pueden hacerse en un solo paso. La longitud del arreglo se determina según la cantidad de elementos separados por comas descriptos dentro del bloque { }.

```
int[]  intArray = {6, 3, 2, 4, 9};
Cliente[] cliArray = {new Cliente(), new Cliente(), new Cliente()};
String[] items = { "item1", "item2", "item3", "item4"};
```

Arreglos
recorrido

El recorrido de una arreglo puede hacerse de 2 maneras:

```
public class SumaArreglo {
 public int suma1(int[] a) {
   int result = 0;
   for (int i=0; i<a.length;i++)
        result = result + a[i];
   return result;
 }
 public int suma2(int[] a) {
   int result = 0;
   for (int elto: a)
        result = result + elto;
   return result;
 }
}
```

(1) For tradicional: Recorre el arreglo desde la primer posición a la última. El **i** toma el índice actual y **a[i]** es el elemento en esa posición, en cada iteración.

(2) F*or-each*: La línea **`for(int elto:a)`** se lee asi: para cada **elemento elto de tipo int**, en el arreglo **a**

```
public class Test {
 public static void main(String[] args){
   int[] datos = {2,4,6};
   SumaArreglo a = new SumaArreglo();
   System.out.println("La suma es:"+a.suma1(datos));
   System.out.println("La suma es:"+a.suma2(datos));
 }}
```

12
12

Arreglos Multidimensionales

Matrices

(1) Tradicional

```
public class RecorridoMatriz {
    public static void main (String args []){
     int [][] notas =
            {{66,78,78,89,88,90},
            {76,80,80,82,90,90},
            {90,92,87,83,99,94}};
      for (int x = 0; x < notas.length; x ++){
            for(int y = 0; y < notas[x].length; y++) {
              System.out.println(notas[x][y]);
            }
      }
    }
}
```

Cant. de filas de la matriz (seria 3)

Cant. de columnas de esa fila (seria 6)

(2) Usando for-each

```
for(int[] fila: notas){
 for(int elto: fila)
  System.out.print(elto);
}
```

1º fila

```
66
78
78
89
88
90
76
80
.
.
```

salida

[] []	0	1	2	3	4	5
0	66	78	78	89	88	90
1	76	80	80	82	90	90
2	90	92	87	83	99	94

Primera dimensión hace referencia al alumno

Segunda dimensión hace referencia a los exámenes

notas[2,3] es 83, hace referencia al cuarto puntaje del tercer alumno

Pasaje de parámetros en Java

Pasaje de parámetros en Java

En Java los parámetros se pasan por valor. Pasaje por valor significa que cuando se invoca a un método, se pasan como argumentos al método una copia de cada parámetro actual.

```
package  ayed.tp02;

public class PasajePorValor {
  public static int mult(int x, int y) {
    return x * y;
  }
  public static void main(String[] args) {
    int alto = 10;
    int ancho = 5;
    int area = mult(alto, ancho);
  }
}
```

Parámetros formales
Son los parámetros en la definición del método

Parámetros actuales/reales
Son los parámetros en la invocación al método

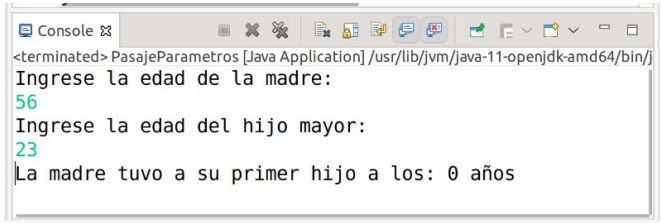
Dentro del cuerpo de un método se puede cambiar el valor de la copia que se recibe como parámetro, pero no tendrá efecto en el parámetro actual.

Pasaje de parámetros

Tipo de dato primitivo

Cuando Java llama a un método, hace una copia de sus parámetros reales y envía las copias al método. Cuando el método finaliza, esas copias se descartan y los valores de las variables en el código principal son los mismos que antes.

```
package tp02.pasajeparametros;
import java.util.Scanner;

public class TestPasajeParametros {
  public static void pedirEdades(int edad1, int edad2) {
    Scanner input = new Scanner(System.in); // Create a Scanner object
    System.out.println("Ingrese la edad de la madre:");
    edad1 = input.nextInt();
    System.out.println("Ingrese la edad del hijo mayor:");
    edad2 = input.nextInt();
  }

 public static void main(String[] args) {
  int edadMadre = 0;
  int edadHijo = 0;
  TestPasajeParametros.pedirEdades(edadMadre, edadHijo);
  System.out.println("La madre tuvo a su primer hijo a los:"+(edadMadre-edadHijo)+" años”);
  }
}
```

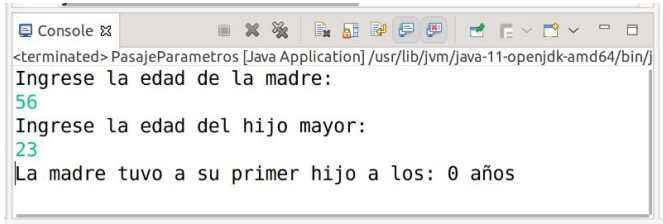


Pasaje de parámetros

Clases wrapper y clase String

Las clases wrapper (**Integer**, **Character**, **Double** ..) y las instancias de **String** son *inmutables*., esto significa que su valor no puede cambiar, de manera que cuando se modifica el valor a una variable de estos tipos, se crea una nueva instancia con el nuevo valor y se le asigna a la variable.

```
package pasajeparametros;
import java.util.Scanner;

public class PasajeParametros {
  public static void pedirEdades(Integer edad1, Integer edad2) {
    Scanner input = new Scanner(System.in); // Create a Scanner object
    System.out.println("Ingrese la edad de la madre:");
    edad1 = input.nextInt();
    System.out.println("Ingrese la edad del hijo mayor:");
    edad2 = input.nextInt();
  }

 public static void main(String[] args) {
   Integer edadMadre = 0;
   Integer edadHijo = 0;
   PasajeParametros.pedirEdades(edadMadre, edadHijo);
   System.out.println("La madre tuvo a su primer hijo a los: "+(edadMadre-edadHijo)+" años”);
  }
}
```



Pasaje de parámetros
Tipo de dato referencial

En Java, podemos pasar como parámetro una referencia a un objeto y con ella cambiar el estado (valor de sus variables) del objeto, pero no la referencia.

```
public class PasajePorValor {
  public static void cambiarNombre(Contacto c) {
    c.setNombre("Pilar");
  }


  public static void main(String[] args){
    Contacto c = new Contacto();
    c.setNombre("Lucia");
    PasajePorValor.cambiarNombre(c);
    System.out.println(c.getNombre());
  }
}
```

¿Qué sucedería si reemplazamos esta línea por estas dos?

```
c=new Contacto();
c.setNombre("Juan");
```

Se pasa una copia de la referencia a c

Calcular el máximo de un arreglo

Dado un arreglo con valores de tipo int, se desea calcular el máximo valor del arreglo.

- ¿Cómo implementamos el método `maximo(arreglo)`?
- ¿Es un método de instancia o de clase?
- ¿Qué parámetros le pasamos?

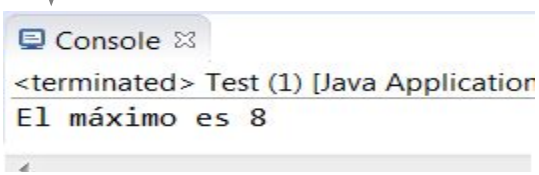
```
package tp02.ejercicio5;

public class Test {

  public static void main(String[] args) {
     int[] datos = {3, 4, 5, 8, 0};
     int max = Calculadora.maximo(datos);
     System.out.println("El máximo es " + max);
   }

}
```

Calcular el máximo de un arreglo

La manera más simple es usar la sentencia **`return`** para devolver el máximo del arreglo.

```
package ayed.tp02;
public class Calculadora {
  public static int maximo(int[] datos) {
    int max = 0;
    for (int i = 0; i < datos.length; i++) {
        if (datos[i] > max)
           max = datos[i];
    }
    return max;
  }
}
```

```
package  ayed.tp02;

public class Test {
  public static void main(String[] args) {
     int[] datos = {3, 4, 5, 8, 0};
     int max = Calculadora.maximo(datos);
     System.out.println("El máximo es " + max);
   }
}
```



¿Cómo podríamos hacer si tenemos que devolver el máximo y el mínimo?

Calcular el máximo y el mínimo de un arreglo

De la misma manera que devolvemos un int, podemos devolver un objeto con el máximo y el mínimo.

```java
package ayed.tp02;

public class Calculadora {
  . . .

 public static Datos maxmin(int[] datos) {
   int max = 0, min = 0;
   for (int i = 0; i < datos.length; i++) {
     if (datos[i] > max) max = datos[i];
     if (datos[i] < min) min = datos[i];
   }
   Datos obj = new Datos();
   obj.setMax(max);
   obj.setMin(min);
   return obj;
 }
}
```

```java
package ayed.tp02;

public class Datos {
  private int min;
  private int max;
  public int getMin() {
    return min;
  }
  public void setMin(int min) {
    this.min = min;
  }
  public int getMax() {
    return max;
  }
  public void setMax(int max) {
    this.max = max;
  }
}
```

En este caso creamos una objeto que mantiene el máximo y el mínimo y devolvemos ese objeto.

Calcular el mínimo y el máximo de un arreglo

Ahora al invocar al método nos devuelve un objeto:

```
package tp02.ejercicio5;

public class Test {

  public static void main(String[] args) {
     int[] datos = {3, 4, 5, 8, 0};
     Datos maxmin = Calculadora.maxmin(datos);
     System.out.println("El máximo es " + maxmin.getMax());
     System.out.println("El mínimo es " + maxmin.getMin());
   }

}
```