

Java

☰ Categoría	segundaParte
⚙ Estado	En curso
☰ Evaluaciones	ex2

▼ Introducción y características

JAVA SE CREÓ CON LA FINALIDAD:

- Utilidad: aws para conectarse a otras máquinas
- Compartir los programas y que puedan ser ejecutados en distintos sistemas operativos , a nivel multiplataforma
- Juego basado en java ejeje juego único
- Virtual Machine → dentro de nuestro sist operativo , tener otro sist op → java tomó esa idea
- Basado en C se puede trabajar con todo (punteros , objetos, etc) → No funcionan la sintaxis exclusiva de c++
- Java es orientado a objetos puro → es un objeto , el int es un ob, una funcion sería un MÉTODO relacionado a una clase
- Debe existir un método denominado main
- WRITE ONE , RUN ANYWHERE
- archivo fuente → .java
- el ejecutable → .class → es multiplataforma → java virtual machine

▼ Punteros ocultos

En java no se puede hacer Mochila M;

- todo elemento generado como Class ob1 → se crea como puntero (oculto)
- Ello ocasiona que se tenga que reservar memoria
- Los int, double , etc no necesitan de reserva

```
Clase objeto; //INSTANCIA
objeto = new Clase; //RESERVA DE MEMORIA
```

▼ Parámetros por valor

- No existe los parametros por referencia

Recolector de basura

- Garbage collector → se ejecuta de manera automatica para liberar memoria

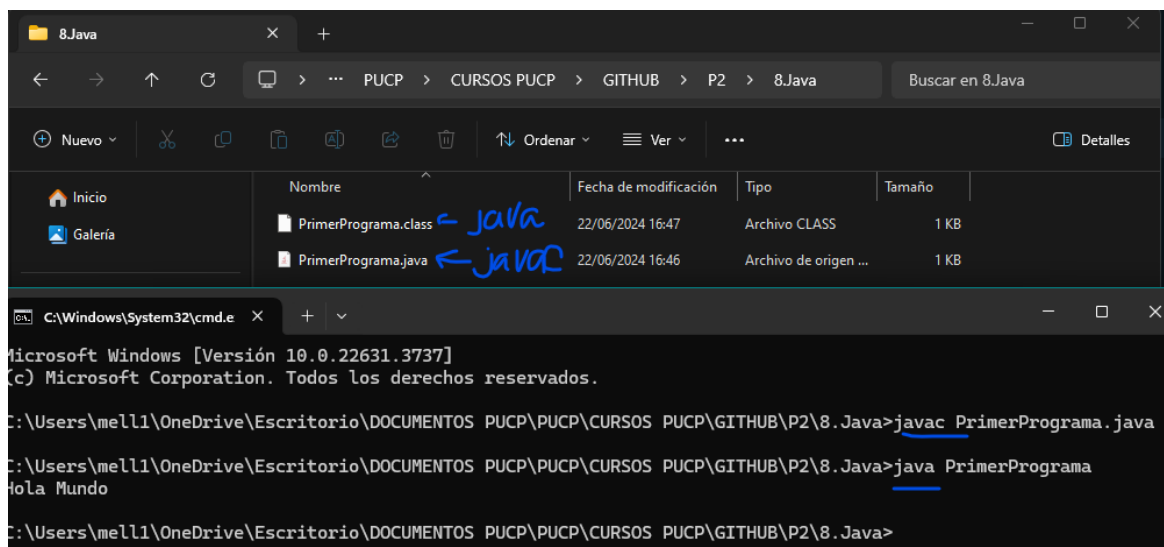
OJOOOOOOOOOO

- JAVA NO TIENE OPERADORES SOBRECARGADOS

▼ Comandos Notepad++

- Ctr + D: copia la linea hacia abajo
- Ctr + K : comenta una linea

▼ Compilación y ejecución desde CMD



- JAVAC: Compila el programa
- JAVA : Ejecuta el programa (imprime resultados). Es el output.

CREACION DE UN PROYECTO JAVA

1. Parte de compilación

2. Parte de ejecución (o interna)

```
javac -cp .PrimerProyecto.java

//esa compilacion bota un archivo .class
java -cp .PrimerProyecto
```

Lectura desde un archivo:

1. Borramos todas las clases existentes (método eficiente y recomendado)

```
del *.class
```

2. Compilamos el programa principal

```
javac -cp . Principal.java //Principal hace referencia al a
```

3. Ejecutamos el programa principal, por comodidad lo redigirimos a un archivo txt de salida

```
java -cp . Principal /*sin .java, ya que es el .class */< c
//                                     ir
```

El comando `java -cp . Principal < datos.txt > reporte.txt` ejecuta la clase `Principal` (en el archivo `Principal.class`) y redirige la entrada estándar desde el archivo `datos.txt` y la salida estándar hacia el archivo `reporte.txt`.

Vamos a desglosar el comando:

1. `java -cp . Principal`: Este es el comando para ejecutar la clase `Principal` en el classpath actual (indicado por `.`).
2. `< datos.txt`: Redirige la entrada estándar desde el archivo `datos.txt`. Esto significa que todo el contenido de `datos.txt` será leído como entrada por el programa `Principal`.
3. `> reporte.txt`: Redirige la salida estándar hacia el archivo `reporte.txt`. Esto significa que cualquier salida generada por el programa `Principal` se guardará en `reporte.txt`.

Por lo tanto, el comando completo ejecutará el programa `Principal`, utilizando el contenido de `datos.txt` como entrada y guardará la salida

generada en `reporte.txt`.

▼ Instrucciones de impresión



En el curso solo se usa `println()` para imprimir el reporte.

En el lenguaje de programación Java, `println` y `printf` son métodos utilizados para mostrar texto en la consola, pero se utilizan en contextos diferentes y tienen funcionalidades distintas. Aquí te explico las diferencias entre ellos y cuándo se usan:

1. `println`:



Solo imprime cadena de texto

- **Propósito:** Este método se utiliza para imprimir texto seguido de un salto de línea.
- **Uso:** Se usa cuando deseas mostrar un mensaje en la consola y automáticamente pasar a la siguiente línea.
- **Sintaxis:** `System.out.println("Texto a imprimir");`
- **Ejemplo:** `System.out.println("Hola, mundo!");` imprimirá `Hola, mundo!` y luego moverá el cursor a la siguiente línea.

2. `printf`:

- **Propósito:** Este método se utiliza para imprimir texto formateado. Es más flexible y poderoso que `println` ya que permite especificar un formato para el texto de salida.
- **Uso:** Se usa cuando necesitas controlar el formato del texto que se va a mostrar, como especificar el número de decimales en un número, alinear el texto, etc.
- **Sintaxis:** `System.out.printf("Formato", valores);`
- **Ejemplo:** `System.out.printf("Valor de x = %.2f\n", x);` imprimirá el valor de `x` con dos decimales y luego moverá el cursor a la siguiente línea debido a `\n`.

3. Salto de línea:

- En Java, un salto de línea se puede lograr de varias maneras:
 - Usando `println` que automáticamente incluye un salto de línea.
 - Usando la secuencia de escape `\n` dentro de una cadena.
- **Ejemplo con `println`**: Esto imprimirá:

```
javaCopy code
System.out.println("Primera línea");
System.out.println("Segunda línea");
```

```
Copy code
Primera línea
Segunda línea
```

- **Ejemplo con `\n`**: Esto imprimirá:

```
javaCopy code
System.out.print("Primera línea\nSegunda línea\n");
```

```
Copy code
Primera línea
Segunda línea
```

▼ printf

En Java, el método `printf` se usa para imprimir texto formateado. La cadena de formato especifica cómo deben ser presentados los valores. Vamos a desglosar el formato `"%10.2f"` y explicar qué significa cada parte:

1. `%`: Indica el inicio de una especificación de formato.
2. `10`: Especifica el ancho mínimo del campo en el que se imprimirá el número. En este caso, el número se imprimirá en un campo de al

menos 10 caracteres de ancho. Si el número tiene menos de 10 caracteres, se añadirán espacios en blanco a la izquierda.

3. `.2`: Especifica el número de dígitos a la derecha del punto decimal. Aquí, `2` indica que se deben mostrar dos decimales.
4. `f`: Indica que el valor a formatear es un número de punto flotante (float o double).

Desglose de `"%10.2f"`

- **Ancho del campo (`10`)**: El número total de caracteres usados para mostrar el número (incluyendo puntos decimales y signos).
- **Precisión (`.2`)**: El número de dígitos a mostrar después del punto decimal.
- **Tipo (`f`)**: Especifica que el valor es un número de punto flotante.

Ejemplo completo en Java:

```
javaCopy code
public class FormatoEjemplo {
    public static void main(String[] args) {
        double x = 123.456;
        int y = 42;
        String nombre = "Juan";
        // 123.46 (10 caracteres de ancho, 2 d
ecimales)
        System.out.printf("%10.2f\n", x);
        System.out.printf("%5d\n", y);    //    42 (5
caracteres de ancho)
        System.out.printf("%-10s\n", nombre); // Juan
(10 caract de ancho, alineado a la izquierda)
        System.out.printf("%08d\n", y);    // 00000042
(8 caracteres de ancho, relleno con ceros)
        System.out.printf("%.3f\n", x);    // 123.456
(3 decimales)
    }
}
```

1. `%5d`:

- Formato: `"%5d"`
- Significado: Un entero (`d`) en un campo de 5 caracteres de ancho.
- Ejemplo:

```
javaCopy code
int n = 42;
System.out.printf("%5d\n", n);
```

- Salida: `42` (con 3 espacios en blanco a la izquierda).

2. `%-10s` :

- Formato: `"%-10s"`
- Significado: Una cadena de texto (`s`) en un campo de 10 caracteres de ancho, alineada a la izquierda (el signo `-` indica alineación a la izquierda).
- Ejemplo:

```
javaCopy code
String str = "Hola";
System.out.printf("%-10s\n", str);
```

- Salida: `Hola` (con 6 espacios en blanco a la derecha).

3. `%.3f` :

- Formato: `"%.3f"`
- Significado: Un número de punto flotante (`f`) con 3 dígitos después del punto decimal.
- Ejemplo:

```
javaCopy code
double num = 3.141592;
System.out.printf("%.3f\n", num);
```

- Salida: 3.142

4. %08d :

- Formato: "%08d"
- Significado: Un entero (d) en un campo de 8 caracteres de ancho, relleno con ceros a la izquierda.
- Ejemplo:

```
javaCopy code
int num = 123;
System.out.printf("%08d\n", num);
```

- Salida: 00000123

5. %6.2f :

- Formato: "%6.2f"
- Significado: Un número de punto flotante (f) en un campo de 6 caracteres de ancho con 2 dígitos después del punto decimal.
- Ejemplo:

```
javaCopy code
double num = 7.5;
System.out.printf("%6.2f\n", num);
```

- Salida: 7.50 (con 2 espacios en blanco a la izquierda).

▼ concatenación

En Java, la concatenación de cadenas (strings) se realiza utilizando el operador `+`. Cuando escribes `System.out.println("A=" + a);`, estás combinando (concatenando) la cadena `"A="` con el valor de la variable `a`.

Aquí tienes una explicación paso a paso de lo que sucede:

1. Parte de la cadena: "A="

- Esta es una cadena literal que contiene los caracteres `A` y `=`.

2. Operador de concatenación: `+`

- El operador `+` se utiliza para concatenar (unir) la cadena literal `"A="` con el valor de la variable `a`.

3. Variable `a`: `a`

- `a` es una variable de tipo entero en tu programa. En este caso, `a` tiene el valor `23`.

Concatenación en acción

Cuando `System.out.println("A=" + a);` se ejecuta, se realiza lo siguiente:

- Java convierte el valor de `a` (que es un entero) en una cadena.
- Luego, concatena la cadena `"A="` con la cadena resultante de la conversión del entero `a`.
- El resultado es la cadena `"A=23"`, que se imprime en la consola.

Ejemplo completo

```
public class PrimerPrograma {  
    public static void main(String[] args) {  
        int a = 23;  
        System.out.println("A=" + a);  
    }  
}
```

En este ejemplo, la salida en la consola será:

```
cssCopy code  
A=23
```

Más ejemplos de concatenación

1. Concatenación de varias variables:

```
javaCopy code  
int x = 10;  
double y = 20.5;  
String str = "Valor de x y y: ";
```

```
System.out.println(str + x + " y " + y);
```

- Salida: Valor de x y y: 10 y 20.5

2. Concatenación de texto y variables:

```
javaCopy code
String nombre = "Juan";
int edad = 25;
System.out.println("Nombre: " + nombre + ", Edad: "
+ edad);
```

- Salida: Nombre: Juan, Edad: 25

3. Concatenación en un bucle:

```
javaCopy code
for (int i = 1; i <= 5; i++) {
    System.out.println("Número: " + i);
}
```

- Salida:

```
makefileCopy code
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
```

Nota sobre la eficiencia

Si realizas muchas concatenaciones dentro de un bucle, considera usar `StringBuilder` para mejorar la eficiencia, ya que las concatenaciones con `+` crean nuevos objetos `String` cada vez, lo que puede ser ineficiente en operaciones repetitivas:

```

javaCopy code
StringBuilder sb = new StringBuilder();
for (int i = 1; i <= 5; i++) {
    sb.append("Número: ").append(i).append("\n");
}
System.out.println(sb.toString());

```

Usar `StringBuilder` es más eficiente en situaciones donde se realizan muchas concatenaciones, especialmente dentro de bucles.

▼ ArrayList

▼ Acceso a índices

si tengo una variable tipo ArrayList en java, es posible acceder al elemento numero 4 por ejemplo?

ChatGPT

Sí, en Java es posible acceder a un elemento específico de un `ArrayList` usando el método `get(int index)`. Los índices en un `ArrayList` son basados en cero, por lo que el primer elemento tiene el índice 0, el segundo elemento tiene el índice 1, y así sucesivamente. Para acceder al cuarto elemento, necesitas usar el índice 3.

Aquí tienes un ejemplo de cómo hacerlo:

```

javaCopiar código
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        // Crear un ArrayList y agregar algunos elementos
        ArrayList<String> lista = new ArrayList<>();
        lista.add("Elemento 1");
        lista.add("Elemento 2");
        lista.add("Elemento 3");
        lista.add("Elemento 4");
        lista.add("Elemento 5");
    }
}

```

```

        // Acceder al cuarto elemento (índice 3)
        String cuartoElemento = lista.get(3);
        System.out.println("El cuarto elemento es: " +
cuartoElemento);
    }
}

```

En este ejemplo:

1. Se crea un `ArrayList` llamado `lista` que contiene cadenas de texto.
2. Se agregan cinco elementos a la lista.
3. Se accede al cuarto elemento usando `lista.get(3)` y se almacena en la variable `cuartoElemento`.
4. Se imprime el cuarto elemento.

Si intentas acceder a un índice que no existe (por ejemplo, un índice mayor o igual al tamaño de la lista), se lanzará una excepción `IndexOutOfBoundsException`. Por lo tanto, es una buena práctica verificar que el índice esté dentro del rango antes de intentar acceder a un elemento:

```

javaCopiar código
int indice = 3;
if (indice >= 0 && indice < lista.size()) {
    String elemento = lista.get(indice);
    System.out.println("El elemento en el índice " + i
ndice + " es: " + elemento);
} else {
    System.out.println("Índice fuera de rango");
}

```

Este código verifica si el índice es válido antes de intentar acceder al elemento en esa posición.

▼ Función impresión

▼ Pasando parámetro `ArrayList<tipo>`

Claro, aquí tienes una función genérica en Java que imprime los elementos de un `ArrayList`. La función recibe el `ArrayList` como

parámetro:

```
javaCopy code
import java.util.ArrayList;

public class Biblioteca {
    // Función genérica para imprimir un ArrayList
    public static void imprimirArrayList(ArrayList<
String> lista) {
        for (String elemento : lista) {
            System.out.println(elemento);
        }
    }

    public static void main(String[] args) {
        ArrayList<String> librosPrestados = new Arr
ayList<>();
        ArrayList<String> librosNoPrestados = new A
rrayList<>();

        // Agregando algunos libros de ejemplo
        librosPrestados.add("El_viejo_y_el_mar");
        librosPrestados.add("Cien_anios_de_soleda
d");

        librosNoPrestados.add("La_divina_comedia");
        librosNoPrestados.add("Matar_a_un_ruisenio
r");

        // Imprimiendo los libros prestados
        System.out.println("Libros prestados:");
        imprimirArrayList(librosPrestados);

        // Imprimiendo los libros no prestados
        System.out.println("Libros no prestados:");
        imprimirArrayList(librosNoPrestados);
    }
}
```

```
}
```

En este ejemplo, `imprimirArrayList` es una función genérica que toma un `ArrayList<String>` como parámetro y luego imprime cada elemento de la lista. En el `main`, se crean dos `ArrayList` de libros prestados y no prestados, se añaden algunos libros de ejemplo, y luego se llama a la función `imprimirArrayList` para imprimir cada lista.

▼ Genérica

Para hacer una función genérica que funcione con `ArrayList` de cualquier tipo de dato, puedes usar la programación genérica en Java. Aquí te muestro cómo puedes hacer una función que imprima un `ArrayList` de cualquier tipo:

```
import java.util.ArrayList;

public class Biblioteca {
    // Función genérica para imprimir un ArrayList
    de cualquier tipo
    public static <T> void imprimirArrayList(ArrayL
ist<T> lista) {
        for (T elemento : lista) {
            System.out.println(elemento);
        }
    }

    public static void main(String[] args) {
        ArrayList<String> librosPrestados = new Arr
ayList<>();
        ArrayList<Integer> numeros = new ArrayList<
>();
        ArrayList<Double> precios = new ArrayList<>
();

        // Agregando algunos libros de ejemplo
        librosPrestados.add("El_viejo_y_el_mar");
        librosPrestados.add("Cien_anios_de_soleda
d");
    }
}
```

```

        // Agregando algunos números de ejemplo
        numeros.add(42);
        numeros.add(7);
        numeros.add(13);

        // Agregando algunos precios de ejemplo
        precios.add(19.99);
        precios.add(9.99);
        precios.add(5.49);

        // Imprimiendo los libros prestados
        System.out.println("Libros prestados:");
        imprimirArrayList(librosPrestados);

        // Imprimiendo los números
        System.out.println("Números:");
        imprimirArrayList(numeros);

        // Imprimiendo los precios
        System.out.println("Precios:");
        imprimirArrayList(precios);
    }
}

```

En este ejemplo, la función `imprimirArrayList` es genérica y utiliza `<T>` para representar cualquier tipo de dato. Esto permite que la función imprima `ArrayList` de cualquier tipo, ya sea `String`, `Integer`, `Double`, etc.

La implementación del `main` muestra cómo usar esta función genérica para imprimir diferentes tipos de `ArrayList`.

▼ Constructor por referencia

```

abstract public class Alumno{ // En este caso el "abstract" es opcional
    private int codigo;
    private String nombre;

    public Alumno(){

    }

    public Alumno(int codigo, String nombre){
        this.codigo = codigo;
        this.nombre = nombre;
    }
}

```

▼ Clase del tipo abstract en polimorfismo

Existen diversos motivos por los que una clase se puede definir como abstracta

- Cuando tenemos un método puro (solo existe el método en las clases hijas) , la clase padre se debe definir como abstract

En el ejemplo de Alumnos de la clase de Guanira...

- Alumno → no es abstract ya que puede existir por si solo (distinto al caso de figura ya que figura sí o sí es una de las derivas) → si hay un metodo puro → si se coloca abstract
- Si alumno es abstract → no se puede poner **new Alumno();**

▼ Apuntes clase guanira - Herencia y polimorfismo (no revisar)

EN EL LAB

- entrar al cmd de java

POLIMORFISMOS

Herencia, sobrescritura (virtual) , punteros

- en java todos son punteros

Si tenemos una clase Base :

```

Bsase objt = new Derivada();

```

de bajo hacia arriba → ver en donde se queda

en c++

```
for(Figura &f : figuras){  
    //cualquier modificacion de aca se guarda gracias a &  
}
```

en java:

no se necesita & ya que todos son punteros

```
del *.class ----> comando para borrar todas las clases y cc  
//luego compilar el principal
```

ULTIMA CLASE

- primeros problemas en tp
- Alumno Regular o de intercambio
- cuantos hay → para leer un for → (final)
- o al final un numero (jemeple de clase)
- Alumno → no es abstract ya que puede existir por si solo (distinto al caso de figura ya que figura sí o sí es una de las derivas) → si hay un metodo puro → si se coloca abstract
- Si alumno es abstract → no se puede poner ***new Alumno();***

```
---> ALUMNO.JAVA  
public Alumno(){  
    //constructor vacio xd  
}
```

```

public void leerDatos(Scanner arch){
    codigo = arch.nextInt();
    nombre = arch.next();
    // ojo -> el final se verifica cuando se lee le
}
//ojito : falta anotar imprimeLinea
system.out.print(c);
system.out.println();//al final del for

```

----> ALUMNOREGULAR.JAVA

```

public AlumnoRegular(){

}

```

@Override

```

public void leerDatos(Scanner arch){
    super.leerDatos(arch);
    especialidad= arch.next();
    facultad= arch.next();
}

```

@Override //no es obligatorio -> es solo sobreindic

```

public void imprimeDatos(Scanner arch){
    super.imprimeDatos();
    System.out.println("Alumno regular: ");
    System.out.println("Especialidad: "+especialidad);
    System.out.println("Facultad"+facultad);
}

```

---- >ALUMNOINTERCAMBIO.JAVA

```

public AlumnoIntercambio(){
    //cuando tenemos un atributo de tipo ArrayList
    idiomas = new ArrayList<String>();
}

```

```

@Override
public void leerDatos(Scanner arch){
    String idioma;
    super.leerDatos(arch);

    paisDeOrigen = arch.next();
    //lee los idiomas, para cuando llegue al numero
    while(!arch.hasNextInt()){ //solamente lo mira,
        idioma = arch.next();
        idiomas.add(idioma);
    }
    //lee el numero de semestres
    numeroDeSemestres = arch.NextInt();
}

@Override //no es obligatorio -> es solo sobreindicar
public void imprimeDatos(Scanner arch){
    super.imprimeDatos();
    System.out.println("Alumno de Intercambio: ");
    System.out.println("Pais de origen: "+paisDeOri);
    System.out.print("Idiomas");
    for(String id:idiomas){
        System.out.print(" "+id);
    }
    System.out.println();
    System.out.println("Numero de sem: "+numeroDeSe
}

```

---- >LISTADEALUMNOS.JAVA

// compareTo -> devuelve un int

// boolean equals -> devuelve si son iguales

```

public ListaDeAlumnos(){
    listaDeAlumnos = new ArrayList<Alumno>();
}

public void leerAlumno(){
    //basta con string , a menos que nos pidan un a

```

```

Scanner arch = new Scanner();
String tipoAlumno;

Alumno alumno;
while(arch.hasNext()){
    tipoAlumno = arch.next();
    if(tipoAlumno.compareTo("R") ==0){
        alumno = new AlumnoRegular();
    }else{
        alumno = new Alumno();
    }
    alumno.leerDatos(arch);
    listaDeAlumnos.add(alumno); //agrega al alu
}
}

public void listarAlumno(){
    for(Alumno al: listaDeAlumnos){
        al.imprimeDatos();
    }
}

---- >PRINCIPAL.JAVA
class ProyectoFinal(){
    public static void main(String[] args){
        ListaDeAlumnos = new ListaDeAlumnos ();
    }
}

```