

POC REPASO

1. Introducción: Tabla de Sintaxis.

- Java hace distinción entre mayúsculas y minúsculas.
- Se puede declarar variables con cualquier carácter como _
- No pueden ser palabras reservadas
- No puede ser igual a otro identificador en ese mismo ámbito.

2. Ampliación de Sintaxis Básica.

• Operadores	» Desplazamiento izq.
	» Desplazamiento derecha signo
	8. AND OR
	A XOR

- Operadores lógicos a nivel de bit.
- Break y Continue. (Se puede usar con etiquetas)
 - ↳ Break: Se sale de las sentencias cuando es invocada.
 - ↳ Continue: Solo funciona para bucles, saltar las líneas que existan hasta el final del bucle y forzar la siguiente iteración.
- Break y Switch.

3. Características de Java.

- Sencillo, Robusto y Neutro en Arquitectura.
- Portable
- Dedicado a Multitareas → Multihilo (Threads)
- Segura (Control de applets)
- Otros aspectos → JISL (Librerías de Java)

4. Introducción Básica.

- Mayor grado de comprensión de los programas → Avantages de la legibilidad y mantenimiento.
- Clase (Elemento Básico)
 - ↳ Moldé de infinitos objetos con características
- Objeto
 - ↳ Ejemplar concreto de una clase, que responde al comportamiento definido por las operaciones de la clase.

- Atributo: Cada uno de los datos de una clase. (color: Tipo gestión, color...)
- Estado: Conjunto de los valores de los atributos que tiene un objeto para pertenecer a una clase.
- M\'etodos: Definici\'on de una operaci\'on de una clase.
- Mensaje: Es la invocaci\'on de una operaci\'on sobre un objeto

• Referencia

- Clase: Descripci\'on de los atributos y de los m\'etodos que describen el comportamiento de un cierto conjunto de objetos hom\'ogeneos
- Objeto: Ejemplar concreto de una clase, que responde al comportamiento, definido por los m\'etodos de la clase.

5. M\'etodos Get y Set.

- Declaramos los atributos como private y ofrecemos m\'etodos, get y set que devuelven su valor y permitan su modificaci\'on.
Se hace así por la encapsulaci\'on.

public void Set () → Consulta variables
 public int Get () → Modificaci\'on.

- Temas de seguridad → Podemos limitar que el numerador nunca sea 0.

6. Vectores - Arrays.

- Los arrays en Java → Se tratan como una clase predefinida.
Son objetos con algunas caracter\'isticas propias.
 - Referencial al Array : double [] x
 - Creaci\'on: Con el operador new

$$\text{int } v[] = \text{new int[10];}$$
 - Acceder al n\'o de elementos → $\text{int longitud} = v.length;$
 - Acceder a un elemento → corchete []
 - Se puede crear arrays con cualquier tipo.
 - Se puede inicializar con valores entre llaves.

String días[] = {"Lunes", "Martes", ...};

- También se pueden inicializar con varias llamadas a new dentro de llaves.
Círculo círculos[] = {new Círculo(), new Círculo()};
- Los operadores de incremento y decremento se pueden usar.
- Se inician al valor correspondiente a su tipo.
- Como todos los objetos, los arrays pasan como argumentos a los métodos por referencia.

7. Arrays Bidimensionales.

- Una matriz es un vector de vectores fila.

int [][] mat = new int [3][4]

8. Usos del This.

- Clasificar los atributos dentro de la clase. (También clasifica los objetos)
- Clasificar los atributos dentro de la misma clase cuando su método se llama a sí mismo. (Del mismo modo)
- Devolver un argumento implícito.
- Llamada entre constructores.

9. Paquetes en Java.

- Cláusula package: Agrupación de clases afines

↳ El nombre de la clase debe ser único dentro del paquete
(Con el mismo nombre en paquetes distintos → da)

↳ Declarar package nombre_package;

↳ Si no se declara un package, no pertenece a ninguno, o sea están en todos

- Cláusula import: Importar paquetes, junto con sus clases o otro paquete.

↳ Se puede importar solo una clase import Ge.Básica. Punto,
↳ se coloca después de la sentencia package.

↳ Paquete Java.lang (Automáticamente)

- **Membre de los paquetes:** Se pueden tener paquetes ordenados por jerarquía
 - mis Paquetes, Figurativo, Geometrica,
- **Ubicación de los paquetes en el sistema.** → Cada una OBL en el equipo
- **Accesibilidad de las clases de un paquete**
 - ↳ Para que una clase pueda ser usada fuera del paquete con import, deba ser necesariamente declarado public
 - ↳ Si no se declara public, solo podrá usarse por clases del mismo paquete.
- ④ **Se puede llamar a un constructor, sin tener ningún constructor escrito, Java nos regala el constructor args (Cuando haya 1 este ya no está)**

10. Igualdad de Objetos.

$a == b$ compara referencias, no valores.
 $a != b$

- a.equals(b) Método pensado para comparar la igualdad de los valores.
 Este método está para todas las clases
 No compara valores salvo que lo redefinamos (Si no se compara con $= =$)

④ Con objetos usar siempre equals no $= =$

11. Estático.

- Tanto métodos y atributos estarán vinculados al objeto que los contiene.
- Es posible crear, métodos y atributos en Java que no pertenezcan a un objeto en particular

Los Métodos o atributos de clase o ESTÁTICOS

Se usa para variables también

```
static int nucirculos = 0
public static int GetCirculos()
```

- | | |
|-------------|------------------------|
| • Atributos | Iniciar Static |
| | Iniciados con un valor |
- Dato: Atributos que no dependen de ningún objeto en particular
- | | |
|-----------|------------------------------------|
| • Métodos | En la cabecera poner <u>static</u> |
| | En su implementación no usar this. |
- Dato: Métodos que no dependen de ningún objeto en particular y sí de toda clase.

12. Modificadores de Acceso en Java.

- Son elementos que se colocan delante de variables, atributos, métodos o clases y las alteran como final, static, public.
- Preceden a la declaración, ya sea de clase, atributo o método

1. Clases Public → Todo el mundo

Sin modificar → Solo clases de dentro paquete

Public: Todo el mundo accede (ver) × recordar, get/set

2. Métodos, Atributos Private: Atributos solo de la misma clase, constructores = y Métodos de otros métodos

Sin modificar: Solo a esta dentro paquete, donde cualquiera

Protected: Relacionado con herencia.

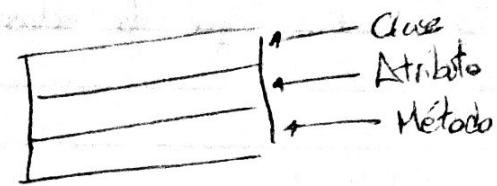
13. Variables.

- Variables de tipos primitivos y Variables de Referencia
 - Variable miembros de una clase
 - Variables locales (Generan dato del {})
- Atributos: Se inicializan siempre de modo automática.
↳ Debe ser primitivos (int, boolean, double...) o referencias a objetos
- Métodos: Se aplican a un objeto de la clase por el operador ()
- Constructores: Tienen características especiales
 - El nombre del constructor tiene que ser igual que el de la clase
 - No devuelve ningún valor → Pero no se declara void
 - Solo invocar con new.
 - Si hacemos varios constructores (Y como se tienen que llamar del mismo nombre) hay que hacer que se diferencien en tipo de argumentos o nº de argumentos.
 - Constructor no (arg) sino hay ~~un~~ ningún constructor más

14. Cadenas de Caracteres.

- La clase String Builder permite al programador, combinar la cadena insertando borrando...
- Operador de concatenación (+) y el método append()
- Los métodos de String se pueden usar sobre literales
`String Str1 = "Hola". Método Crear.`

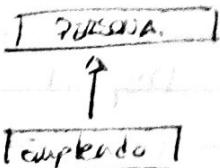
15. Diagrama Clases UML



- Asociaciones: 1, 0..1, N..M. Se muestra en los extremos de la linea
- Agregación \leftrightarrow (Objetos que pueden compartirse)
- Composición \blacktriangleleft (Objetos que no pueden compartirse / o vivir sin el otro)



- Reflexivas.
- Generalización



- Dependencia de tipo \dashrightarrow

* Otros diagramas UML.

1. Diagramas de Objetos (Todas cajas) solas

2. Diagramas de Secuencia

3. Diagramas de Colaboración.

4. Diagramas de Estados.

16. Herencia.

- Creación de otras clase, con una ya existente. (Se puede heredar características)
- Se basa en la reutilización del Código (Hereda funcionalidad de la clase base y le añade otros comportamientos)
 - Aprovechar cosas ya hechas.

• La Clase Base o Superclase.

- Sus atributos son protected

• La Clase derivada, hereda atributos y métodos, más sus propios atributos (declarados protected también)

public class VentanaTitulo extends Ventana

- Extends es la palabra reservada que indica, que la clase derivada, deriva de la otra clase.
- La primera llamada de la clase derivada es una llamada al constructor de la clase base mediante la palabra Super → Super(x, y, z) inicializa los atributos de la clase derivada.
- Para redifinir un método de la C.Base, basta con poner super.mostrar() y añadir lo que le falta.

Después de redifinirla no hace falta llamar a super para usarla. ventana.mostrar()

- Para mismo paquete usar ~~super~~ protected, para distintos paquetes usar público.

• Operador InstanceOF

- Tiene dos operando, un objeto (lado izq) y una clase en el lado derech.
- Devuelve TRUE o FALSE de que el objeto instancie a la clase.

• Herencia II

- Sep pueden heredar atributos Super(x,y)
- Clases abstractas → Clases que solo se usan como base para otros (No se pueden usar objetos pertenecientes a una clase abstracta)
 - + Deben tener un método (o varios) abstractos
- Clases o métodos finales
 - + Se declara con final cuando no nos interesa de derivar nro. de esa clase.
 - + final en método, entra que lo redifinan.

17 Interfaz y Polimorfismo.

- Interfaz: Colección de declaraciones de métodos. (sin definir)
Definen el comportamiento de las clases.
 - ↳ Si una clase implementa una interfaz, está obligado a implementar todos sus métodos (Serializable/Comparable)
- Diferencias entre Clases y Interfaces.
 - ↳ Una interfaz es solo una lista de métodos, sin implementar.
 - ↳ Una clase puede衍生 de otras (extends)
 - ↳ Los nombres de las interfaces se colocan con la palabra implements (sus métodos son siempre public y abstract)
 - ↳ Cada interfaz debe ser definido en un fichero con el mismo nombre que la interfaz.
- Se relacionan entre Clases y Interfaces
 - ↳ Las interfaces se pueden heredar (extender) entre ellas
 - ↳ Se pueden redefinir las constantes y los métodos.

18. Excepciones.

- Java incorpora su propio lenguaje de gestión de errores.
- Excepción: Certo tipo de error o una condición anormal que se ha producido durante la ejecución de un programa.
- Por heredar los Throwable, se pueden usar
 - String getMessage() → Extrae mensaje asociado a la excepción
 - String toString() → String que describe la excepción
 - Void printStackTrace() → Indica la secuencia de métodos por los que ha pasado la excepción.
- Lanzar una Excepción
 - Crear un objeto Exception de la clase adecuada
 - Se lanza → Throw seguida del objeto Exception creado.
 - Al lanzar una excepción el método termina.
 - ↳ Todo método que lanza 1 o más excepciones debe declararla en el encabezamiento con throws (separadas por comas)
- ↳ Se puede poner un superclass de excepciones y ya.
- Capturar una excepción
 - Para capturar habrá que gestionar la excepción con una construcción del tipo try f.h catch f.h y re-lanzar la excepción hacia un método anterior en el stack, declarando que su método.

19. Colecciones.

- Vamos a estudiar un conjunto de clases e interfaces de Java que formaran la Java Collections Framework (proporcionan la capacidad de trabajar con estructura de datos)
- Interfaces de la JCF
 - Collection: Define métodos para tratar una colección genérica de elementos
 - Set: Colección que no admite elementos repetidos.
 - SortedSet: Set cuyos elementos se mantienen ordenados según criterio
 - List: Admite elementos repetidos y mantiene un orden inicial.
 - Map: Conjunto de pares clave/valor sin repeticion claves
 - SortedMap: map cuyos elementos se mantienen repetidos según criterio.

• Interfaces de soporte (Diapos.)

- Comparaciones: Dado que ciertas colecciones trabajan ordenando sus elementos, necesitamos forma de Comparar.
 - Interfaz Comparable → public int compareTo
 - Interfaz Comparator → public int compare(Object o1, Object o2)
- Genéricos: Definir clases y parámetros (var) genéricos?
 - Utilizar Generics de Java
(Apéndice sobre Generics → 11ta ejemplos)

• El bucle For-Each.

20. Ficheros en Java.

- Los streams pueden ser de entrada o de salida.
- Los ficheros pueden ser, de texto (Almacenan datos caracteres) o binario (Almacenan bytes)

↳ Se accede a un fichero, creando un objeto asociado al stream de ese tipo.

- Para trabajar con ficheros, hay que importar java.io.File.

↳ Streams de Entrada → File InputStream

↳ Streams de Salida → File OutputStream

- Lectores y Escritores. (Ficheros de Texto)

- Subclase Reader → File Reader (Lectura caracteres de fichero texto)

- Subclase Writer → File Writer

- La serialización de un objeto consiste en generar una secuencia de bytes lista para su almacenamiento o transmisión.

↳ Para que un objeto sea serializable debe implementar la interfaz java.io.Serializable

↳ Todas las tipos primitivos son serializables

- Otras clase de java.io - StreamTokenizer.

↳ Permite Partir el flujo a Tokens.

TT-EOL Fin linea

TT-EOL Fin Fichero

- Ficheros de Acceso Aleatorio → La clase RandomAccessFile nos permite representar un fichero de Acceso Aleatorio.

① Archivos de texto LECTURA.

```
try {
    FileReader fr = new FileReader("archivo.txt");
    BufferedReader entrada = new BufferedReader(fr);
```

```
String s
while (s = entrada.readLine() != null)
    texto += s
entrada.close
```

```
} // Lectura
catch → Excepción
```

② Archivos de texto ESCRITURA.

```
try {
    FileWriter fw = new FileWriter("Escríbeme.txt");
    BufferedWriter bw = new BufferedWriter(fw);
    PrintWriter salida = new PrintWriter(bw);
```

```
Salida.println ("Soy la 1st linea");
Salida.close
```

```
} // Escritura
catch → Excepción.
```

③ Archivos Binarios LECTURA Y ESCRITURA.

```
try { // Escritura }
```

```
FileOutPutStream f = new FileOutputStream("prueba.dat");
BufferedOutPutStream b = new BufferedOutPutStream(f);
DataOutPutStream dos = new DataOutPutStream(b);
```

```
dos.write (variable)
```

```
dos.close()
```

```
- try { lectura }
```

```
FileInput Stream f = new FileInput Stream ("prueba.dat");
BufferedInput Stream b = new BufferedInput Stream (f);
DataInput Stream dis = new DataInput Stream (b)
```

```
double d2 =
dis.read Double()
```

④ Serialización ESCRITURA Y LECTURA.

try {

```
FileOutputStream f1 = new FileOutputStream ("archivo.ser");
ObjectOutputStream objat = new ObjectOutputStream (f1);
```

```
String s = new String ("Me van a serializar");
objat.writeObject(s);
```

```
FileInputStream f2 = new FileInputStream ("archivo.ser");
ObjectInputStream objin = new ObjectInputStream (f2);
```

```
String s2 = (String) objin.readObject();
}
```

Estructura de Colecciones en Java.

- ArrayList: Almacena los objetos en un array (List)
- LinkedList: List. Almacena en una lista enlazada (List)
- TreeSet: Árbol binario Ordenado (SortedSet)
- Hashmap: Tabla hash → Colección formada por pares de claves y valor que se almacena en una tabla (Map)
- TreeMap: Árbol binario (Map)

1. Interfaces de JCF

- Collection: Colección Genérica
- Set: Colección que no admite elementos repetidos
- SortedSet: Con elementos ordenados según un criterio.
- List: Admite Elementos repetidos y mantiene orden
- Map: Conjunto de pares clave/valor sin repetición de claves
- SortedMap: Map con elementos Ordenados.

2. Interfaces de Soporte.

- Iterator: Recorre colección y borra
 - ↳ ListIterator: Permite recorrer la lista en ambos sentidos.
- Comparable: Declara el método compareTo()
- Comparator: Método compare()

- Método añadir → boolean add(Objeto) "Devuelve true si añadido"
- Método Eliminar → boolean remove(Objeto) "Devuelve true si eliminado"
- Método de Consulta → int size()
- boolean isEmpty() "Devuelve true si vacía"
- Método para Recorrer → iterator() Permite recorrer con next()
 - * Más adelante interfaz iterator
- Método equals → equals(object) "True si son iguales"

- Interfaz Comparable y Comparator
 - Comparable → public int compareTo(Object obj)
 - Comparator → compare() Devuelve -1, 0, 1 según su argumento sea anterior, igual o posterior al segundo.

import java.util.Comparator

- Interfaz Iterator.
 - Permiten recorrer colecciones. Dispone de un conjunto de métodos
 1. HasNext(): Devuelve cierto si el elemento actual tiene siguiente en la colección.
 2. Next(): Devuelve una referencia al siguiente elemento de la colección. Coloca el iterador en el elemento siguiente.
 3. Remove(): Elimina de la colección el último elemento retornado por next().
 - La listaIterator deriva de iterator, pero con métodos para movernos de aante a atrás.
 - hasSet(), next(), hasPrevious(), previous(), nextIndex(), previousIndex(), remove()
 - add(), set() (Reemplaza el último elemento usado)

- Interfaz List
 - Constructores
 - ArrayList () (collection)
 - Métodos
 - add (Object)
 - contains (Object)
 - Object get (int index) → Devuelve elemento de la lista que ocupa ese lugar
 - Object set (int index, Object o) → Sustituye

- Genericos

```
List<String> ls = new ArrayList<String>();  
ls.add(new Integer(0));  
Integer x = ls.iterator().next();
```

< > → Especifican de que es la lista.

(1) Crea (2) Añade (3) saca elem.

- Interfaz Comparable.

compareTo (Object o)

compare (Object o1, Object o2)

```
public int compareTo (Objeto otro) {
```

Fraccion otro2 = (Fraccion) otro
if ...

```
public int compare (Object o1, Object o2) {
```

:

EJEMPLOS EN ARRAYLIST.

1. Inicializar List lista = new ArrayList()

2. Añadir lista.add (new Integer(i))
lista.add ("Alex")

3. Eliminar objetos con Iterator

```
Iterator i = lista.iterator()  
while (i.hasNext())
```

4. Convertir a array

```
Object[] objArray = lista.toArray()
```

5. Eliminar lista.clear()