[Resumen Examen 1 - Programación Avanzada – JAVA 2](#_Toc197894553)

[Clase 2](#_Toc197894554)

[Objeto 2](#_Toc197894555)

[Clase vs Objeto 2](#_Toc197894556)

[Tipo de datos 3](#_Toc197894557)

[Atributos 4](#_Toc197894558)

[Tipo de atributos 4](#_Toc197894559)

[Control de Flujo 5](#_Toc197894560)

[Condicionales 5](#_Toc197894561)

[IF, ELSE IF, ELSE 5](#_Toc197894562)

[Switch 5](#_Toc197894563)

[Bucle condicional 6](#_Toc197894564)

[While 6](#_Toc197894565)

[Do – While 6](#_Toc197894566)

[Bucle controlado 6](#_Toc197894567)

[For 6](#_Toc197894568)

[For – each 6](#_Toc197894569)

[Control de flujo del bucle 7](#_Toc197894570)

[Overload y Override 7](#_Toc197894571)

[Override 7](#_Toc197894572)

[Overload 8](#_Toc197894573)

[Pilares de la Programación orientada a Objetos (POO) 8](#_Toc197894574)

[Encapsulamiento 8](#_Toc197894575)

[Herencia 9](#_Toc197894576)

[Super 9](#_Toc197894577)

[Abstracción 10](#_Toc197894578)

[Polimorfismo 10](#_Toc197894579)

[Tipo de polimorfismos 11](#_Toc197894580)

[Excepciones 12](#_Toc197894581)

[Métodos comunes 14](#_Toc197894582)

[De Objetos 14](#_Toc197894583)

[De Strings 14](#_Toc197894584)

[De Arrays 15](#_Toc197894585)

[De listas (List) 16](#_Toc197894586)

[De Exepciones 17](#_Toc197894587)

# Resumen Examen 1 - Programación Avanzada – JAVA

Recomendable mas practicar que lo teórico

## Clase

Es un **modelo o plantilla** que tiene **propiedades que la definen**, tiene **comportamientos llamado métodos**. Es importante en POO

### Objeto

Es la **instancia de una clase**, se puede hacer las veces que se quiera, cada objeto es único

* **Estado:** Representado por sus **atributos (variables)**
* **Comportamiento:** Representado por sus **métodos (funciones)**
* **Identidad:** Una referencia única en la memoria

**Se crea usando la palabra clave new para instanciar una clase**

Ej.: New Persona

#### Clase vs Objeto

* La **clase** es el modelo o plano
* El **objeto** es una instancia concreta de ese modelo
* Clase: Plano de una casa
* Objeto: Una casa construida a partir de ese plano

| **Aspecto** | **Clase** | **Objeto** |
| --- | --- | --- |
| ¿Qué es? | Un plano, plantilla o modelo. | Una instancia concreta de la clase. |
| ¿Cuántos? | Solo uno (el modelo). | Múltiples (cada instancia creada). |
| Memoria | No ocupa memoria hasta que se crean objetos. | Ocupa memoria cada vez que se crea. |
| Ejemplo | Clase: Persona. | Objeto: Juan, María. |

## Tipo de datos

Existen los primitivos y NO primitivos

**Primitivos**: Son los mas básicos, directos del lenguaje. No son objetos

| **Tipo** | **Tamaño** | **Rango** | **Ejemplo** |
| --- | --- | --- | --- |
| **byte** | 8 bits | -128 a 127 | byte edad = 25; |
| **short** | 16 bits | -32,768 a 32,767 | short año = 2023; |
| **int** | 32 bits | -2,147,483,648 a 2,147,483,647 | int numero = 100; |
| **long** | 64 bits | -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 | long distancia = 10000000000L; |
| **float** | 32 bits | ±1.4E-45 a ±3.4E+38 (con decimales) | float precio = 19.99F; |
| **double** | 64 bits | ±4.9E-324 a ±1.7E+308 (con más precisión) | double pi = 3.14159265359; |
| **char** | 16 bits | Un solo carácter Unicode | char letra = 'A'; |
| **boolean** | 1 bit | true o false | boolean activo = true; |

* Son **almacenados directamente en la memoria** (pila o stack)
* No son objetos, por lo que no tienen métodos ni propiedades
* Siempre tienen un **valor predeterminado** si se declaran como atributos de clase

**NO primitivos:** Objetos creados a partir de clases o interfaces

| **Tipo** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| **String** | Cadena de texto. | String nombre = "Juan"; |
| **Array** | Colección de valores del mismo tipo. | int[] numeros = {1, 2, 3}; |
| **Class** | Plantilla para crear objetos. | Persona persona = new Persona(); |
| **Interface** | Plantilla de métodos sin implementación. | interface Volador {} |
| **Collection** | Listas, conjuntos y mapas. | List<Integer> lista = new ArrayList<>(); |

* Son **objetos**, lo que significa que tienen métodos y propiedades
* Se almacenan en la **memoria heap** y su referencia se guarda en el stack
* Su valor predeterminado es null (no apuntan a nada)
* Pueden ser personalizados (clases propias)

## Atributos

También llamados **campos o variables** son las **características** del objeto. Suelen ser variables que pertenecen a una clase, representando un estado

### Tipo de atributos

* De **instancia**: **Pertenece a cada objeto creado a partir de la clase.** Significa que si creo otro objeto de esa clase, tendrá su propia información

Texto

El contenido generado por IA puede ser incorrecto.

Ejemplo: Cada vez que se crea una New Persona, tendrá cada una su nombre, pudiendo ser diferente

* **Estático**: Pertenece a la clase y no al objeto, **todos comparten el mismo valor**, definiéndose con la clave **static**

Texto

El contenido generado por IA puede ser incorrecto.

Ejemplo: Aunque cree varios objetos con New Configuracion, el campo “sistemaOperativo” será el mismo para cada objeto creado (En este caso todos tendrán como “Windows”). Y si se cambia el valor, se cambiará para todos los objetos creados

* Constantes (final): **Su valor no se puede modificar luego de ser asignado**. Ese valor será permanente, se usa la palabra clave **final** (En general va también con static). Para respetar nomenclaturas, se **escriben en mayúscula**

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Ejemplo: PI siempre será ese valor, no se podrá cambiar

## Control de Flujo

### Condicionales

#### IF, ELSE IF, ELSE

Permiten ejecutar bloques según alguna condición

Texto

El contenido generado por IA puede ser incorrecto.

#### Switch

Permite elegir entre múltiples opciones de manera clara

Texto

El contenido generado por IA puede ser incorrecto.

### Bucle condicional

#### While

Repite un bloque de código mientras la condición sea verdadera

Texto

El contenido generado por IA puede ser incorrecto.

#### Do – While

Es como el while pero repite un bloque de código **al menos una vez**

### Bucle controlado

#### For

Repite un bloque de código un numero especifico de veces

Texto

El contenido generado por IA puede ser incorrecto.

#### For – each

Bucle para correr arrays y colecciones

Texto

El contenido generado por IA puede ser incorrecto.

### Control de flujo del bucle

* **Break**: Detiene el bucle completamente
* **Continue**: Salta a la siguiente iteración

Texto

El contenido generado por IA puede ser incorrecto.

## Overload y Override

### Override

Override o sobreescritura, **es cuando se redefine un método, pero manteniendo el mismo nombre, parámetro y tipo de retorno**

Existe este:

Texto

El contenido generado por IA puede ser incorrecto.

Pero lo pisa así en otro lado

Texto

El contenido generado por IA puede ser incorrecto.

### Overload

Overload o sobrecarga es cuando tienen el **mismo nombre de método, pero con diferentes parámetros**, haciendo que lo sobrecargue

Texto

El contenido generado por IA puede ser incorrecto.

## Pilares de la Programación orientada a Objetos (POO)

### Encapsulamiento

Es básicamente **proteger al dato de un objeto haciéndolo accesible solo con lugares específicos** (O sea funciones/métodos puntuales)

Existen los modificadores:

* **Private:** Acceso solo dentro de la **misma** **clase**
* **Protected:** Acceso solo dentro del **mismo** **paquete** (**También** en **subclases**)
* **Public:** Acceso en cualquier lugar

**Default** es si no se le pone ningún tipo de encapsulamiento, es de acceso solo hasta el **mismo paquete, pero no subclases**

Y con **Getters** y **Setters** se controla el acceso

| **Modificador** | **Clase** | **Paquete** | **Subclase (herencia)** | **Mundo (cualquier clase)** |
| --- | --- | --- | --- | --- |
| private | ✅ | ❌ | ❌ | ❌ |
| default | ✅ | ✅ | ❌ | ❌ |
| protected | ✅ | ✅ | ✅ | ❌ |
| public | ✅ | ✅ | ✅ | ✅ |

### Herencia

Pilar de la **POO**. Permite que **una clase herede atributos y métodos de otra clase**: Una **subclase/hija** hereda de **superclase/padre.** Con esto se puede reutilizar código y crear jerarquías entre clases. Su palabra reservada es **Extends**

Un dato importante es que en JAVA solo **se puede heredar de una clase**

Mientras mas para las clases hijas, es cada vez mas especifico. Caso contrario al revés, mientras más vas para la clase padre es más general cada vez

SerVivo 🡪 Animal 🡪 Perro 🡪 FoxTerrier

#### Super

**Super** es una palabra clave en la herencia, se refiere a los **atributos y funciones de su clase padre**

Si estoy en la clase padre *Animal* y tiene un atributo llamado “nombre” …

Texto

El contenido generado por IA puede ser incorrecto.

Cuando lo herede su clase hijo *Gato* y quiere usar un atributo de su padre tendrá que llamarlo con “super.nombre”

Texto

El contenido generado por IA puede ser incorrecto.

Esto también podría pasar con funciones al poner “**super.”,** ejecutando la función del padre

### Abstracción

**Oculta los detalles complejos y muestra solamente lo esencial de la función (Su tipo de retorno, su nombre, que elementos obtiene, etc)**. Se logra usando **clases abstractas e interfaces**. Cuando se quiera usar en otra parte la función abstracta, **se tiene que sobrescribir** (**Override**)

Una **clase abstracta no se puede instanciar directamente**. Se debe heredar por otra clase. Esta **puede tener métodos abstractos y no abstractos**

Ejemplo: Tengo la clase abstracta Vehículo

Texto

El contenido generado por IA puede ser incorrecto.

Y Auto será su hijo, entonces hereda de Vehículo, pero como la función encender es abstracta, debe darte una lógica y ser objetiva

Texto

El contenido generado por IA puede ser incorrecto.

### Polimorfismo

**Permite que objetos de una clase hija sea tratado como un objeto de su clase padre**. Hace que tenga varios comportamientos dependiendo del objeto que lo usa

#### Tipo de polimorfismos

**En tiempo de compilación (Sobrecarga/Overloading)**

* **Tener varias funciones con el mismo nombre**, pero con diferentes parámetros
* No es una herencia, pero es una forma de polimorfismo

Texto

El contenido generado por IA puede ser incorrecto.

*Mismos nombres, pero diferentes tipos*

**En tiempo de ejecución (Sobreescritura/Override)**

* **Usa la herencia y el sobrescribir funciones**
* La función llamada depende del tipo de objeto en la ejecución

Ejemplo:

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

## Excepciones

Es un evento que ocurre cuando se ejecuta un programa y se interrumpe de forma anormal, este maneja errores o condiciones de manera controlada. **Entonces se devuelve el objeto por medio de throw y se guarda en esta la información del error ocurrido**

* Es una **clase** y su clase base es **Throwable**
* Se manejan explícitamente con **try – catch o declarando** con **throws**
* Se lanza un **objeto**

Texto

El contenido generado por IA puede ser incorrecto.

Tambien se puede usar el **throw** para lanzar manualmente una excepción desde el código

Texto

El contenido generado por IA puede ser incorrecto.

El **thows** declara que un método puede lanzar excepciones que debe ser manejada por quien lo llama, es mas para especificar que excepción lo va a manejar

Ej.: En este IOException lo va a manejar

Texto

El contenido generado por IA puede ser incorrecto.

**Finally**: Un bloque que se ejecuta siempre, haya excepción o no

Texto

El contenido generado por IA puede ser incorrecto.

Las excepciones se pueden personalizar, se puede hacer para ser más descriptivos o específicos

Se crea una clase que herede de Exception, se definen constructores adecuados y se lanza donde sea necesario

Texto

El contenido generado por IA puede ser incorrecto.

## Métodos comunes

### De Objetos

| **Método** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| **.toString()** | *Devuelve una representación en texto del objeto.* | obj.toString(); |
| **.equals(Object o)** | *Compara si dos objetos son iguales.* | obj1.equals(obj2); |
| **.hashCode()** | *Devuelve un código hash para el objeto.* | obj.hashCode(); |
| **.getClass()** | *Devuelve la clase del objeto.* | obj.getClass(); |
| **.clone()** | *Crea una copia del objeto. (Debe implementar Cloneable)* | obj.clone(); |

### De Strings

| **Método** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| **.length()** | *Devuelve la longitud de la cadena.* | "Hola".length(); // 4 |
| **.charAt(int index)** | *Devuelve el carácter en una posición específica.* | "Hola".charAt(1); // 'o' |
| **.toUpperCase()** | *Convierte la cadena a mayúsculas.* | "hola".toUpperCase(); // "HOLA" |
| **.toLowerCase()** | *Convierte la cadena a minúsculas.* | "HOLA".toLowerCase(); // "hola" |
| **.trim()** | *Elimina espacios en blanco al inicio y al final.* | " Hola ".trim(); // "Hola" |
| **.substring(int start)** | *Extrae una subcadena desde un índice.* | "Hola".substring(1); // "ola" |
| **.replace(char, char)** | *Reemplaza caracteres específicos.* | "Hola".replace('o', 'a'); // "Hala" |
| **.contains(String)** | *Verifica si contiene una subcadena.* | "Hola".contains("la"); // true |
| **.startsWith(String)** | *Verifica si inicia con una subcadena.* | "Hola".startsWith("Ho"); // true |
| **.endsWith(String)** | *Verifica si termina con una subcadena.* | "Hola".endsWith("la"); // true |
| **.split(String)** | *Divide la cadena en un array por un delimitador.* | "a,b,c".split(","); |
| **.isEmpty()** | *Verifica si la cadena está vacía.* | "".isEmpty(); // true |

### De Arrays

| **Método** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| **Arrays.toString(array)** | *Convierte el array a texto.* | System.out.println(Arrays.toString(arr)); |
| **Arrays.sort(array)** | *Ordena el array.* | Arrays.sort(arr); |
| **Arrays.fill(array, valor)** | *Rellena el array con un valor.* | Arrays.fill(arr, 0); |
| **Arrays.copyOf(arr, length)** | *Copia el array con un nuevo tamaño.* | int[] copia = Arrays.copyOf(arr, 5); |
| **Arrays.asList(array)** | *Convierte el array a lista.* | List<String> lista = Arrays.asList(arr); |

### De listas (List)

| **Método** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| **.add(E elemento)** | *Añade un elemento al final de la lista.* | lista.add("Hola"); |
| **.get(int index)** | *Obtiene el elemento en una posición.* | lista.get(0); |
| **.size()** | *Devuelve el tamaño de la lista.* | lista.size(); |
| **.contains(Object o)** | *Verifica si contiene un elemento.* | lista.contains("Hola"); |
| **.remove(Object o)** | *Elimina el primer elemento encontrado.* | lista.remove("Hola"); |
| **.clear()** | *Elimina todos los elementos.* | lista.clear(); |
| **.isEmpty()** | *Verifica si la lista está vacía.* | lista.isEmpty(); |
| **.indexOf(Object o)** | *Devuelve el índice del primer elemento encontrado.* | lista.indexOf("Hola"); |
| **.forEach(Consumer)** | *Aplica una acción a cada elemento.* | lista.forEach(System.out::println); |
| **.sort(Comparator)** | *Ordena la lista según el criterio dado.* | lista.sort(Comparator.naturalOrder()); |

### De Exepciones

| **Método** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| **.getMessage()** | *Devuelve el mensaje de la excepción.* | e.getMessage(); |
| **.printStackTrace()** | *Imprime el seguimiento del error.* | e.printStackTrace(); |
| **.getCause()** | *Devuelve la causa de la excepción (otra excepción).* | e.getCause(); |
| **.toString()** | *Muestra la excepción como texto.* | e.toString(); |