



Programació Bàsica

CAS 5

CFGS-ASIX/DAW | 2020-2021

Profesora María Merino | **Grupo 5** Daniela Gallardo | **Fecha** 14/04/2021

ÍNDICE

INTRODUCCIÓN	3
1. ¿QUÉ ES UN OBJETO?	4
1.1 Estructura y Comportamiento	4
2. CLASES	5
2.1 Clases vs. Objetos	5
2.2 Clases: Sintaxis	7
2.3 A Partir de una Clase: ¿Cómo Podemos Obtener Objetos?	8
2.4 Valor Null	8
2.5 Interacción entre Objetos	8
2.6 Modificadores de Acceso	9
2.7 Paquete	9
2.8 ¿Qué Significa Construir un Objeto?	9
2.8.1 Constructores: Qué son, Qué hacen, Cómo se usan	10
2.8.2 Constructores: Sintaxis	10
2.8.3 Constructores: Puntero this	11
2.9 REPASEMOS Y AMPLIEMOS: Métodos y Atributos	12
2.9.1 Estructura Atributos de una Clase	13
2.9.2 Comportamiento Métodos de una Clase	14
2.9.3 Getters/Setters	15
2.9.4 toString()	18
3. ARRAYS BIDIMENSIONALES	19
4. DESCOMPOSICIÓN ESQUELETO ENUNCIADO CAS5	20
4.1 Creación de Ficheros	20
3.2 Personalización Inicial en Cada Ficheros .java	20
3.2.1 GestioTeatreUlldecona.java	20
3.2.2 Clients.java	22
3.2.3 Obra.java	25

3.2.4 Seient.java	27
3.2.5 Teatre.java	27
5. PROGRAMA	30
CONCLUSIÓN	31
WEBGRAFÍA	32

INTRODUCCIÓN

Un paradigma es un marco o cristal por el que podemos guiarnos para entender ciertos contextos, conceptos o cosas en general.

Java se sitúa en el paradigma de programación orientada a objetos y, desde este paradigma todos los elementos que manejamos en nuestro(s) programa(s) son objetos.

Los objetos no son más que instancias/ejemplos contruidos a través de un molde, al que llamamos clase. De modo que si nuestro programa contiene más de una clase, los objetos interaccionan entre ellos mediante el paso de mensaje o instrucciones y, estos mensajes e instrucciones se transmiten mediante el uso de métodos. A continuación se presenta el hilo conductor que da lugar, ejemplo y lógica a lo que acabamos de describir.

1. ¿QUÉ ES UN OBJETO?

Un objeto es cualquier cosa sobre la que podemos emitir un concepto:

- Un pato
- Un camión
- Una bombilla
- Una persona
- Una ventana
- etc.


Un objeto en el lenguaje de Java, no tiene que ser necesariamente tangible, como hemos dicho, puede ser cualquier cosa sobre la que podemos emitir un concepto.

En nuestros programas en Java, podemos construir una representación o representaciones de estos objetos y estas, poseen una **estructura** además de uno o varios **tipos de comportamiento**.

1.1 Estructura y Comportamiento

Por lo general, todos los objetos poseen estructura (se refiere a cómo están conformados) y comportamiento (ejecutan operaciones).

Por ejemplo, un **cochecito de juguete**: es de plástico, tiene 4 ruedas, se puede mover adelante y hacia atrás.

Automóvil de Juguete	
Estructura	Comportamiento
<ul style="list-style-type: none">• plástico• 4 ruedas• ...	<ul style="list-style-type: none">• mover delante• mover detrás• ...
	

2. CLASES

Parejo al concepto de objeto, tenemos el concepto **CLASE**.

Fijémonos en la siguiente imagen: tenemos un grupo de cochecitos de juguete; cada uno de ellos, podemos decir que es un objeto, los hay azul completo, azul con bordes amarillos, amarillo, plateado, púrpura, rojo, cobre, negro (...). Sin embargo, todos ellos tienen algo en común, nos referimos a ellos como **cochecitos de juguete**.



- Una **clase** es una matriz/molde con el que podemos construir objetos de un tipo.
- La **clase** (o “molde”) determina el comportamiento que podrían tener el o los objetos que construya a partir de ella .
- A este molde como tal, no lo consideramos un objeto; una **clase** **no es** un **objeto**.

2.1 Clases vs. Objetos

Una clase sería, por ejemplo, una representación de una persona. A nivel estructural podríamos querer saber su Nombre, Edad, Nacionalidad, Ocupación, Estado Civil y a nivel de comportamiento, podríamos definir si puede hablar, ver, caminar, nacer, morir.

Todo lo que acabamos de mencionar viene representado en este “molde” para fabricar personas y corresponde a la clase Persona.

Clase Client [molde]	
Estructura	Comportamiento/habilidades /cualidades
<ul style="list-style-type: none"> • Nom • Edat • Diners • ... 	<ul style="list-style-type: none"> • pagarEntrada • teDiners • esMajorDedat

Como objetos de la clase persona, podríamos tener los tres que están representados a continuación mediante imágenes (con nombres diferentes, edades que podrían ser diferentes, nacionalidades diferentes, ocupaciones que podrían ser diferentes, etc), todos ellos podrían hablar, ver, caminar, nacer, morir y realizar todas las operaciones que están definidas dentro de la clase.



2.2 Clases: Sintaxis

Un concepto fundamental en la programación orientada a objetos son las Clases. Para definir una clase debemos tener claro que:

- La palabra reservada para su declaración es **class**.
- La declaración e implementación de una clase estará en un mismo fichero.
- Los nombres de una clase usan la notación **UpperCamelCase**.

Para declarar una clase, tendríamos un código con la sintaxis que se muestra a continuación:

- un modificador de acceso (explicado en detalle más adelante), que permitirán a otra clase acceder a otra o a interaccionar con otra.
- La palabra reservada **class** y el nombre de la clase.
- Y encerrado entre llaves:
 - Su estructura: a través de una serie de propiedades.
 - Su comportamiento: a través de una serie de métodos.
- Las propiedades que puede tener:
 - puede ser de tipo primitivo: puede ser de tipo entero, string y/o float.
- Y su comportamiento, con el uso de métodos que podrán ser de distinto tipo, podrán tener o no un tipo de retorno, que será el tipo de valor que retorne su ejecución.

```
<modificador> class NombreDeLaClase {  
  
    //propiedades  
    int propiedad1;  
    String propiedad2;  
    float propiedad3;  
    //...  
  
    //metodos  
    void metodo1() {  
        //...  
    }  
  
    //...  
}
```


2.3 A Partir de una Clase: ¿Cómo Podemos Obtener Objetos?

Lo hacemos mediante una operación que se llama **Instanciación** o **Instancia** (proviene del inglés **instance**) y, que en programación quiere decir “en este contexto, ejemplo o caso”.

- A la construcción de un objeto a partir de un “molde”, que en este caso, sería a partir de una **clase** lo llamaremos **instanciación** o **instancia**.
- Su sintaxis es muy parecida a la de una variable de un tipo primitivo.
- Utilizamos el operador **new**.

Esta sentencia en la que instanciamos un objeto de tipo Client, tiene varias partes, aunque la sentencia es sencilla; de izquierda a derecha tendría:

- El tipo de dato estructurado **Client**, que es la clase del objeto a declarar.
- El nombre del objeto (también conocido como referencia): **client**.
- El operador de instancia: **new**.
- El constructor, que corresponde al mismo nombre de la clase y a continuación unos paréntesis.
- La finalización de la sentencia con un punto y coma.

Tipo(clase)	Nombre del objeto (referencia)	Operador de instancia	constructor
Client client = new Client();			

Esto nos permitiría poder construir un objeto de Client; instanciar un client.

2.4 Valor Null

Si declaramos una referencia, pero no construimos ningún objeto en ese momento, obtenemos una referencia nula, o con valor **null**. Además, es una palabra reservada que podemos usar para comparar.

Ejemplo
Client client1;

2.5 Interacción entre Objetos

Como mencionamos anteriormente, los objetos interactúan entre sí a través del paso de mensajes. Este paso de mensajes se realiza llamando a los métodos de un objeto mediante el uso de otro método.

En la práctica sería de la siguiente manera:

- Utilizamos el nombre de la referencia de este objeto.
- añadimos un punto

- escribimos el nombre del método

```
client.pagarEntrada();  
client.teDiners();
```

2.6 Modificadores de Acceso

Tanto a nivel de clase, como de atributos y métodos, Java pone a nuestra disposición una serie de modificadores de acceso, que harán que no cualquier clase pueda acceder a otra.

public: Es el menos restrictivo. Cualquiera puede acceder a esa clase o método.

protected: Sólo puede acceder a ese método o atributo la propia clase, o una que herede de ella.

private: Sólo puede acceder a ese método o atributo desde la propia clase.

Por defecto (es decir, sin modificador de acceso): pueden acceder a esa clase todas las del mismo paquete.

2.7 Paquete

Es una agrupación de clases. Indicamos que una clase pertenece a un paquete mediante la palabra reservada **package** en la primera línea de código, antes de declarar la clase:

Ejemplo

```
package pkgTeatregrama;  
  
public class Client{  
    //...  
}
```

2.8 ¿Qué Significa Construir un Objeto?

Si rescatamos la sentencia de instancia de la construcción de un objeto de tipo **Client** como vemos a continuación. Lo que vemos es una operación compleja en la memoria de nuestro ordenador porque:

1. Almacena en dos sitios diferentes:
 - a. Por un lado **client** (en minúscula), que es el nombre del objeto es una referencia hacia el objeto en sí, es decir, esa referencia no es el objeto en sí, sino que una manera de acceder al objeto.
 - b. El objeto se crea en otra zona de memoria diferente conocida como **heap**, donde encontraremos objetos almacenados y se construye a través del **operador de instanciación** y del **constructor**.

Tipo(clase)	Nombre del objeto (referencia)	Operador de instancia	constructor
<code>Client client = new Client();</code>			

2.8.1 Constructores: Qué son, Qué hacen, Cómo se usan

- Un constructor es **como** un **método especial**, posee el mismo nombre que su clase y es sintácticamente similar a un método.
- Un constructor inicializa un objeto cuando se crea; utilizaremos un constructor para **dar valores iniciales a las variables** de instancia definidas por la clase, o para realizar cualquier otro procedimiento de inicio requerido para crear un objeto completamente formado.
- Se invoca con el operador **new**.
- Un constructor de clase, por ejemplo, nos permitirá crear instancias a **Client**; con un constructor de clase podemos especificar datos de **Client**.
- Los constructores no tienen un tipo de devolución explícito.
- Normalmente, **Todas las clases tienen constructores**, ya sea que definamos uno o no, porque Java proporciona automáticamente un constructor predeterminado que nos permite usar esta clase, pero no nos permite definir una información inicial determinada.
- **Pueden haber varios constructores en una misma clase.**

2.8.2 Constructores: Sintaxis

- Normalmente son **public**.
- Un constructor se llama **siempre** igual que el nombre de la clase a la cual construye.
- Pueden tener cero o más argumentos.

Cabecera del constructor de la clase Client que acepta 3 argumentos:

- nom
- edat
- diners

```
public Client (int String, int edat, double diners) {
    ...
}
```

2.8.3 Constructores: Puntero *this*

- **this** es una palabra reservada.
- Sirve para hacer referencia para un objeto desde dentro de sí mismo.
- Ayuda a la encapsulación: variables, métodos, constructores.

Por ejemplo en este código añadimos unos modificadores de acceso:

1. La clase es pública; cualquiera puede acceder a ella.
2. Sin embargo su estructura pasa a ser privada (desde este punto es evidente que estamos aplicando encapsulación), esto quiere decir, que no cualquier objeto puede acceder a la estructura de un objeto Client, pero él mismo sí y para esto utilizamos el puntero **this**.

```
public class Client {  
    // Estructura de la clase: atributos para cada cliente  
    private String nom;// nombre del cliente  
    private int edat;// edad del cliente  
    private double diners;// cant. de dinero del cliente  
  
    //constructor con 3 argumentos (nom, edat i diners)  
    public Client (String nom, int edat, double diners) {  
        this.nom=nom;  
        this.edat=edat;  
        this.diners=diners;  
    }  
}
```

2.9 REPASEMOS Y AMPLIEMOS: Métodos y Atributos

Como mencionamos anteriormente, los **métodos** corresponden al **comportamiento de una clase** y los **atributos** (o propiedades) con la **estructura de la misma**.

CLASE		
ESTRUCTURA ATRIBUTOS	nom edat diners	<pre> public class Client{ private String nom;// nombre del cliente private int edat;// edad del cliente private double diners;// dinero del cliente </pre>
	majorEdat	<pre> public boolean majorEdat() { return edat >= 18; } </pre>
	teDiners	<pre> public boolean teDiners(Teatre teatre) { return (diners >= teatre.getPreu()); } </pre>
	pagarEntrada	<pre> public double pagarEntrada(Client client, Theatre teatre) { if (client.teDiners(teatre) == true &&(client.getDiners() > teatre.getPreu())) { diners = diners - teatre.getPreu(); } else { System.out.println(client.getNom() + " no té prou diners per pagar l'entrada"); } return diners;// actualizamos el valor de diners o no } </pre>
	toString	<pre> public String toString() { return "Client nom: " + nom + " Edat: " + edat + " Diners: " + diners + "€"; } } </pre>

Como mencionamos antes, un objeto interactúa con otro objeto a través de métodos; invocándolos, esto se conoce como **encapsulación** y, dado que la estructura interna de un objeto solamente la conoce él mismo, **lo normal es que los atributos de un objeto se oculten** (haciéndolos privados) y **se consulten o editen siempre a través de una serie de métodos**.

2.9.1 Estructura Atributos de una Clase

Conforman la estructura de una clase y la sintaxis normal de la declaración de un atributo de una clase tendrá siempre para cada atributo tres partes:

1 Modificador de Acceso	2 Tipo de Dato del Atributo	3 Nombre del Atributo
<code>modificadorDeAtributo</code>	<code>tipoAtributo</code>	<code>nombreAtributo;</code>
<ul style="list-style-type: none">• private• protected• public• ...	<ul style="list-style-type: none">• char• int• float• double• String• Otra clase	<ul style="list-style-type: none">• Notación camelCase• Autodescriptivo

2.9.2 Comportamiento Métodos de una Clase

Conforman el comportamiento de la clase. También poseen:

- 1. Modificadores de acceso** del método y, adicionalmente podemos utilizar algunas de las palabras reservadas (static, abstract, final, native, synchronized) y que darán un carácter especial u otro al método (por ahora no la utilizaremos pero debemos saber que pueden haberlas).
- 2. Tipo de retorno:** los métodos los podemos hacer para que no devuelvan ningún valor, o podemos hacer que, el resultado de su ejecución devuelva algún tipo de valor.

Tenemos que indicar cuando definimos un método es el tipo de dato que va a devolver:

- Si no devuelve nada, indicamos de vuelve vacío; void.
- Si devuelve, puede devolver:
 - Un Tipo Primitivo: char, int, float, double, String.
 - Otra Clase

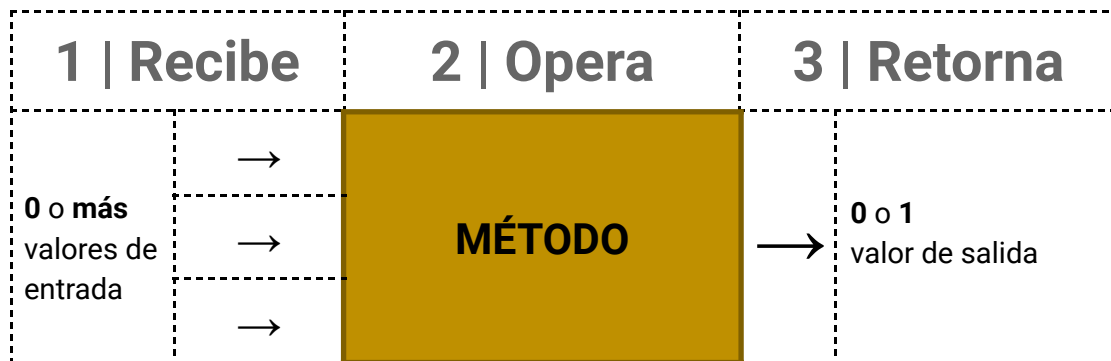
- 3. Nombre del método.**

- 4.** Entre paréntesis una lista separada por comas de los **parámetro(s) que recibe ese método**, que también puede ser sólo uno y que se define como una variable de toda la vida **tipoDato** nombreParámetro (seguido de una coma si hay más de uno).

- 5.** Seguido del paréntesis, entre llaves {}, vendría el **"cuerpo del método"**.

1 Modificadores de Acceso	2 Tipo de Dato del Método	3 Nombre del Método	4 Parámetro(s) que recibe	5 Cuerpo del Método
modificadorDeMétodo	tipoMétodo	nombreMétodo;	(tipoDato nombreParámetro, ...)	{...}
<div><ul style="list-style-type: none">• private• protected• public• ...</div> <div><ul style="list-style-type: none">staticabstractfinalnative...</div>	<ul style="list-style-type: none">• char• int• float• double• String• Otra clase	<ul style="list-style-type: none">• Notación camelCase• Autodescriptivo	<div><ul style="list-style-type: none">• char• int• float• double• String• Otra clase</div> <div><ul style="list-style-type: none">• Notación camelCase• Autodescriptivo</div>	aquí va qué ejecuta el método

Podemos ver los métodos como una caja que **1)** Recibe una serie de valores de entrada, **2)** dentro suceden una serie de operaciones y que **3)** retorna 0 o 1 valor de salida.



2.9.3 Getters/Setters

Son unos métodos especiales, pero sencillos, porque nos permiten implementar la encapsulación.

1. Controlan el acceso a los atributos de una clase.
2. Nos permiten establecer condiciones para acceder o modificar los atributos de una clase.

Los métodos getter (get) nos servirán para para obtener el valor actual de un atributo determinado.

Los métodos setter (set) nos servirán para establecer/modificar el valor de ese atributo.

Pero ¿cómo funcionan exactamente?

Si volvemos al ejemplo que utilizamos para explicar el uso del puntero **this**, mencionamos:

- La palabra reservada **this** sirve para hacer referencia para un objeto desde dentro de sí mismo.
- Forma parte de la encapsulación: variables, métodos, constructores.

Por ejemplo al siguiente código añadimos unos modificadores de acceso:

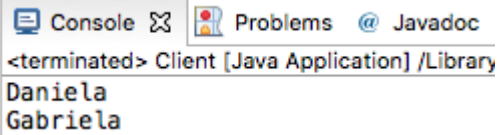
1. **La clase es pública**; cualquiera puede acceder a ella.
2. Sin embargo, **su estructura pasa a ser privada** (desde este punto estamos aplicando encapsulación), esto quiere decir, que no cualquier objeto puede acceder a los atributos de un objeto de la clase Client, sólo se puede acceder a sus atributos desde ella misma y, para esto, utilizamos el puntero **this**.
3. El puntero **this** lo utilizaremos en el cuerpo de un constructor de la clase Client, que nos ayudará a inicializar los valores para los objetos que queramos construir.

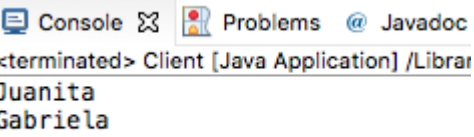
En otras palabras: Desde otro archivo java no podemos acceder directamente a los atributos de la clase Client; necesitamos un método.

4. Si queremos obtener el valor actual de un atributo, necesitamos un método get.

5. Si queremos cambiar un valor de un atributo Client, necesitamos un método set.	
public class Client {	1
<i>// atributos</i> private String nom; private int edat; private double diners;	2
<i>//constructor</i> public Client (String nom, int edat, double diners) { this .nom=nom; this .edat=edat; this .diners=diners; }	3
<i>/**</i> <i>* Método getNom. Obtiene el nombre del cliente</i> <i>* @return the nom</i> <i>*/</i> public String getNom() { return nom; }	4
<i>/**</i> <i>*Método setNom. Establece el nombre del cliente.</i> <i>* @param nom the nom to set</i> <i>*/</i> public void setNom(String nom) { this .nom = nom; }	5
} <i>//fin de la clase Client</i>	1

Comprobamos el funcionamiento del método getNom():	
1. Creamos dos objetos de la clase Client 2. Probamos de imprimir por pantalla	
public static void main(String[] args) {	
<i>// objetos cliente</i> Client client1 = new Client("Daniela",30,500); Client client2 = new Client("Gabriela",28,1000);	1

<pre>// Prueba imprimir atributos método getNom System.out.println(client1.getNom()); System.out.println(client2.getNom());</pre>	2
<pre>}// fin main</pre>	
3. Ejecutamos	
 <pre><terminated> Client [Java Application] /Library Daniela Gabriela</pre>	

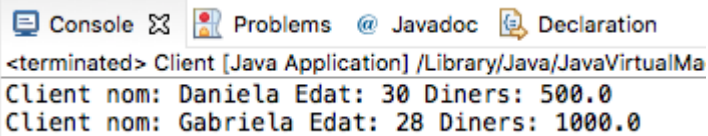
Comprobamos el funcionamiento del método setNom(): <ol style="list-style-type: none"> 1. Probamos de cambiar el nombre de uno de los objetos que ya creamos 2. Probamos de imprimir por pantalla 	
<pre>public static void main(String[] args) { // objetos cliente que ya tenemos Client client1 = new Client("Daniela",30,500); Client client2 = new Client("Gabriela",28,1000);</pre>	
<pre>// modificar valor nom con método setNom client1.setNom("Juanita");</pre>	1
<pre>// imprimir nom con método getNom System.out.println(client1.getNom()); System.out.println(client2.getNom());</pre>	2
<pre>}</pre>	
3. Ejecutamos	
 <pre><terminated> Client [Java Application] /Librar Juanita Gabriela</pre>	

2.9.4 toString()

Corresponde a otro método especial. toString() Sirve para representar un objeto como una cadena de caracteres, es decir, en un sólo valor de tipo String para que podamos imprimirlo más tarde.

Esto quiere decir que transforma un objeto en una cadena de caracteres a través de los valores de sus atributos.

En la clase Client encontramos: 1. Declaración de sus atributos. 2. Declaración de Constructor/es de objetos de la clase Client. 3. Declaración de Getters 4. Declaración de Setters 5. Declaración de Método toString	
public class Client{	
<i>// atributos</i> private String nom; private int edat; private double diners;	1
[Constuctor/es]...	2
[Getters]...	3
[Setters]...	4
<i>/**</i> <i>* Método toString. Devuelve Los atributos un obj.</i> <i>* Client en una cadena de caracteres.</i> <i>* @param atributos Client.</i> <i>* @return String*/</i> public String toString() { return "Client nom:" + nom + " Edat:" + edat + " Diners:" + diners; }	5
}// fin class Client	

Probamos su implementación	
<code>public static void main(String[] args) {</code>	
<pre> // 1.creamos objetos cliente Client client1 = new Client("Daniela",30,500); Client client2 = new Client("Gabriela",28,1000); </pre>	1
<pre> //2.Probamos imprimir atributos con met. toString() System.out.println(client1.toString()); System.out.println(client2.toString()); </pre>	2
<code>}// fin main</code>	
3. Ejecutamos	
 <pre> <terminated> Client [Java Application] /Library/Java/JavaVirtualMa Client nom: Daniela Edat: 30 Diners: 500.0 Client nom: Gabriela Edat: 28 Diners: 1000.0 </pre>	

3. ARRAYS BIDIMENSIONALES

Ya hemos trabajado en el Cas 3 y el 4 con **arrays**. Recordemos que un array es un contenedor de elementos de un mismo tipo, con un tamaño fijo que hay que conocer a la hora de crearlo. Y también que a la hora de recorrerlo, con distintos tipos de bucles, las posiciones comienzan a contar en cero.

Podemos crear un array que tenga más de una dimensión. Tan solo tenemos que añadir otra pareja de corchetes. En el Cas5, necesitamos hacer un array bidimensional en la clase Teatre.java en un atributo de tipo Seient al que llamaremos sessió, esto nos permitirá crear una sala de teatro.

3.1 Declarando el array

Seguiremos los siguientes pasos para llevarlo a cabo:

Declaramos el array y añadimos tantos corchetes como dimensiones necesitemos;

1. Necesitamos dos índices;
 - a. uno para la dimensión de filas de asientos.
 - b. y otro para dimensionar la cantidad de asientos por fila.
2. Como podemos asignar distintas cantidades de filas a cada objeto de la case teatro creamos un atributo fila
3. También podemos asignar distintas cantidades de asientos por fila creamos un atributo num,

```
public class Teatre {  
    ...  
    // 1. butacas disponibles c/nro de fila y nro de asiento  
    private Seient[][] sessio;  
    //2. para dimensionar la cant. de filas  
    private int fila;  
    //3. dimensionar el nro de butacas/seients por fila  
    private int num;  
    ...  
    //universo constructor//  
    ...  
} //fin clase teatre
```

3.2 Inicializando su construcción

Como dijimos anteriormente, un constructor inicializa un objeto cuando se crea; utilizaremos un constructor para dar valores iniciales a las variables de instancia definidas por la clase, o para realizar cualquier otro procedimiento de inicio requerido para crear un objeto completamente formado.

Entonces para crear un constructor de un objeto de la clase teatre:

Pensamos en cómo debemos darle un valor inicial a un objeto de la clase Teatre.
Pensamos en que estamos creando una sala de teatro

```
...
//universo constructor//
// constructor que acepta 4 argumentos (obra, preu, fila, num)
    public Teatre(Obra obra, double preu, int fila, int num) {
        Teatre.obra = obra;
        this.preu = preu;

        // asignar una llargada al array de asientos para filas y
        nro de asientos
        this.sessio = new Seient[fila][num];

        // iniciar los valores de los índices
        this.fila = fila;
        this.num = num;

        // iniciar el conteo que recorra ambos índices y damos
        valores a cada fila y cada num de asiento y a client
        for (int i = 0; i < fila; i++) {
            for (int j = 0; j < num; j++) {
                sessio[i][j] = new Seient(i + 1, j + 1, null);
            }
        }
    }
...

```

4. DESCOMPOSICIÓN ESQUELETO ENUNCIADO CAS5

Lo primero que realizaremos para descomponer el enunciado del Cas 5 será crear el esqueleto de nuestro programa de la siguiente manera:

1. Crear los ficheros .java
2. Personalización inicial en cada fichero .java, pondremos:

- a. los atributos de la classe
- b. cabecera del/los constructores
- c. cabecera de los métodos

4.1 Creación de Ficheros

Los ficheros que crearemos deberán tener la siguiente estructura:

```

├── GestioTeatreUlldecona
│   └── src
│       ├── pkgGestioTeatreUlldecona
│       │   └── GestioTeatreUlldecona.java
│       └── pkgTeatregrama
│           ├── Public.java
│           ├── Obra.java
│           ├── Seient.java
│           └── Teatre.java

```

3.2 Personalización Inicial en Cada Ficheros .java

3.2.1 GestioTeatreUlldecona.java

1. Creamos el programa principal e importamos el paquete pkgTeatregrama que contiene las clases Public.java, Obra.java, Seient.java y Teatre.java
2. También importamos java.util.Scanner para leer por teclado

```

package pkgGestioTeatreUlldecona;
import pkgTeatregrama.*;
import java.util.*;

public class GestioTeatreUlldecona {
    // método main
    public static void main(String[] args) {

        // 1. Declarar array con opciones del menú fuera de la
función
        String[] opciones = new String[4];
        Scanner sc = new Scanner(System.in);
        opciones[0] = "[o]Crear Obra de teatre";
        opciones[1] = "[s]Crear Sala de teatre";
        opciones[2] = "[p]Crear Persona del público";
        opciones[3] = "[x]sortir";

        // 2. variable para salir del programa
        boolean sortir = false;

```

```

do {
    //var para guardarnos la opcion seleccionada
    char opcio;
    //llamamos al metodo para imprimir el menu
    opcio=menu(opcions);

    switch (opcio) {

        case 'o':
            System.out.println("[O]Crear Obra de
teatre");
            break;

        case 's':
            System.out.println("[S]Crear Sala de
teatre");
            break;

        case 'p':
            System.out.println("[P]Crear Persona del
público");
            break;

        case 'x':
            System.out.println("[X]sortir");
            System.out.println("Fin del programa.
¡Adiós!");
            sortir = true;
            break;

        default:

            System.out.println("\n-----");
            System.out.println("| OJO! debes ingresar
una opción válida!");
            System.out.println("-----\n");
            sortir=false;
        }

    } while (sortir == false);
}

/**
 * Función Menú. Imprime el menú de opciones por pantalla y
 * pide ingresar una por teclado
 * @param opcions

```



```

    * @return opció seleccionada tipo char
    */
    public static char menu(String[] opciones) {
        System.out.println("-----");
        System.out.println("|          MENÚ TEATRE          |");
        System.out.println("-----");
        for (int i = 0; i < opciones.length; i++) {
            System.out.println(opciones[i]);
        }
        // Preguntamos qué opción seleccionarán
        System.out.println("-----");
        System.out.println("|          QUÉ VOLS FER?          |");
        System.out.println("-----");
        System.out.println("    [ingressa una opció]    ");

        // entrada por teclado
        Scanner sc = new Scanner(System.in);
        // guardamos la entrada en opcio
        char opcio1 = sc.next().charAt(0);
        //retornamos opcio al main
        return opcio1;
    }
}

} // fin clase GestioTeatreUlldecona

```

3.2.2 Clients.java

1. Crearemos una clase Public para representar a cada uno de los clientes del teatro.

```

//1. Creamos una clase public para los clientes
public class Client {
    //2. Estructura de la clase: Creamos los atributos para cada cliente
    private String nom;// nombre del cliente
    private int edat;// edad del cliente
    private double diners;// cantidad de dinero del cliente
    private double valorEntrada=20;//valor entrada

    //3. métodos//

    /**
     * @return the nom
     */
}

```

```

public String getNom() {
    return nom;
}

/**
 * @return the edat
 */
public int getEdat() {
    return edat;
}

/**
 * @return the diners
 */
public double getDiners(){
    return diners;
}

public double getValorEntrada() {
    return valorEntrada;
}

/**
 * @param nom the nom to set
 */
public void setNom(String nom){
    this.nom = nom;
}

/**
 * @param edat the edat to set
 */
public void setEdat(int edat){
    this.edat = edat;
}

/**
 * @param diners the diners to set
 */
public void setDiners(double diners){
    this.diners = diners;
}

public void setValorEntrada(double valorEntrada) {

```

```

        this.valorEntrada = valorEntrada;
    }

    /**
     * Método teDiners. Mira si el cliente tiene suficiente dinero
    para pagar la
     * entrada al teatro. Utiliza el paso de valores por valor.
     * @param double valorEntrada (para comparar con el total de
    diners que tiene el cliente)
     * @return true or false
     */
    public boolean teDiners(double valorEntrada){
        boolean resultado;

        return resultado;
    }

    /**
     * Método pagarEntrada. Resta el dinero de la entrada al dinero
    del cliente.
     * Utiliza el paso de valores por valor y actualiza el atributo
    diners.
     * @param double valorEntrada
     * @return double diners actualizado (cambio después de pagar
    entrada).
     */
    public double pagarEntrada(double valorEntrada) {

    }

    /**
     * Método majorEdat. Mira si el cliente es mayor de edad o no.
     * @param nada.
     * @return true or false.
     */
    public boolean majorEdat(){
        boolean resultado;

        return resultado;
    }

    /**
     * Método toString.
     * Devuelve los atributos del cliente en forma de frase.
     * @param atributos Client.

```

```

    * @return Cadena de caracteres con los datos del cliente.
    */

    public String toString() {

    }

    // Constructor/es//

    //constructor que acepta 3 argumentos (nom, edat i diners)
    public Client (String nom, int edat, double diners) {

    }

    //constructor que acepta 2 argumentos (edat i diners), el nom
    Lo pondremos como «anonim».
    public Client(int edat, double diners) {

    }

    // constructor que acepta 1 argumento (diners), el nombre lo
    pondremos como «anonim» y edat 99
    public Client(double diners){

    }

```

3.2.3 Obra.java

2. Crearemos una clase Public para generar el molde que me permitirá crear un objeto del tipo obra

```

public class Obra {
    /**
     * Els atributs seran el titol,
     * La durada,
     * L'autor
     * i si és per a majors d'edat.
     */
    private String titol;
    private int durada;
    private String autor;
    private boolean EsParaMajorsdEdat;

    /**

```

```

    * mètodes (getters i setters i toString).
    */

/**
 * @return the titol
 */
public String getTitol() {
    return titol;
}

/**
 * @return the durada
 */
public double getDurada() {
    return durada;
}

/**
 * @return the autor
 */
public String getAutor() {
    return autor;
}

/**
 * @return the EsParaMajorsdEdat
 */
public boolean EsParaMajorsdEdat() {
}

/**
 * @param titol the titol to set
 */
public void setTitol(String titol) {
}

/**
 * @param durada the durada to set
 */
public void setDurada(int durada) {
}

/**
 * @param autor the autor to set
 */

```

```

    public void setAutor(String autor) {
    }

    /**
     * @param majorEdat the majorEdat to set
     */
    public void setEsParaMajorsdEdat(boolean EsParaMajorsdEdat) {
    }

    public String toString() {

    }

    //constructor que acepta 4 argumentos (titol, durada, autor,
    EsParaMajorsdEdat)
    public Obra(String titol, int durada, String autor, boolean
    EsParaMajorsdEdat){

        }
    }
}

```

3.2.4 Seient.java

Esqueleto clase Seient

```

public class Seient {

    /**
     * Els atributs seran
     * la fila,
     * el numero
     * i public (la persona que l'ocupa o null).
     */
    private String fila;
    private int num;
    private String publico;

    // Métodos//

    /**
     * @return the fila
     */
    public String getFila() {
        return fila;
    }
}

```

```

}

/**
 * @return the num
 */
public int getNum() {
    return num;
}

/**
 * @return the publico
 */
public String getPublico() {
    return publico;
}

/**
 * @param fila the fila to set
 */
public void setFila(String fila) {
    this.fila = fila;
}

/**
 * @param num the num to set
 */
public void setNum(int num) {
    this.num = num;
}

/**
 * @param publico the publico to set
 */
public void setPublico(String publico) {
    this.publico = publico;
}

public String toString() {
    return "Seient fila=" + fila + ", num=" + num + ",

```

```

publico=" + publico;
    }

    /**
     * Método seientDisponible.
     * Confirma si un asiento está disponible o no.
     * @param
     * @return true or false
     */
    public boolean seientDisponible(){
        boolean resultado=true;//por decir algo

        return resultado;
    }

    //
    -----//
    // ----- Constructor/es
    -----//
    //
    -----//
    //constructor que acepta 3 argumentos (fila, num, publico)
    public Seient(String fila, int num, String publico){
        this.fila=fila;
        this.num=num;
        this.publico=publico;
    }
}

```

3.2.5 Teatre.java

1. Clase Teatre para representar el teatro
2. Los atributos de la clase
 - a. obra
 - b. preu
 - c. un array bidimensional de tipus Seient per als seients anomenat **sessio**.
 - d. dos atributos de tipo int: fila y num, se utilizarán para asignar los índices del array bidimensional.
3. Métodos
 - a. getters i setters,

b. constructores c. métodos reservar asiento, ver asientos ocupados, listar público y método toString de la clase Teatre	
public class Teatre {	1
private static Obra <i>obra</i> ;// <u>nom de la obra</u>	2 . a
private double <i>preu</i> ;// <u>valor de la entrada</u>	2 . b
private Seient[][] <i>sessio</i> ;// <u>butacas c/nro de fila y nro de asiento</u>	2 . c
private int <i>fila</i> ;// <u>para asignar el índice de filas al array sessio</u> private int <i>num</i> ;// <u>para asignar el índice de num de butaca</u>	2 . d
/**Getter <u>de los datos de la obra</u> * <u>@return the obra</u> */ public Obra getObra() { return <i>obra</i> ; } /**getter <u>del precio de la entrada</u> * <u>@return the preu</u> */ public double getPreu() { return <i>preu</i> ; } /**getter del nro de filas y butacas por filas * <u>@return the sessió</u> */ public Seient[][] getSessio() { return <i>sessio</i> ; } /**Definir la obra * <u>@param obra the obra to set</u> */ public void setObra(Obra <i>obra</i>) { Teatre. <i>obra</i> = <i>obra</i> ; } /**Definir el precio de la obra * <u>@param preu the preu to set</u> */ public void setPreu(double <i>preu</i>) { this . <i>preu</i> = <i>preu</i> ; } /**Definir la sesión del teatro:nro de filas y butacas por fila * <u>@param sessio the sessio to set</u> */	3 . a

<pre> public void setSessio(Seient[][] sessio) { this.sessio = sessio; } </pre>	
<pre> // constructor <u>que acepta 4 argumentos (obra, preu, fila, num)</u> public Teatre(Obra obra, double preu, int fila, int num) { } </pre>	3. b
<pre> //constructor <u>de obra</u> public Teatre(String titol,int durada,String autor,boolean EsParaMajorsdEdat){ obra = new Obra(titol, durada, autor, EsParaMajorsdEdat); } </pre>	
<pre> /** * <u>Método reservaSeient()</u>. * <u>Para asignar una butaca a un client.</u> * <u>Si no está disponible avisa.</u> * @param sessio * @return <u>nada</u> */ public void reservaSeient(Seient seient) { } /** * <u>Método voreSeientsOcupats()</u>. * <u>Muestra por pantalla visualmente cómo queda</u> * <u>el patio de butacas(seients)en forma de tabla.</u> * <u>Siendo Ocupado "X" y Disponible "O"</u> * @param nada * @return <u>nada</u> */ public void voreSeientsOcupats() { } /** * <u>Método que lista al público.</u> * @param nada * @return <u>nada</u> */ public void llistarPublico() { } /** * <u>Método toString.</u> </pre>	3c

```
* @param nada  
* @return nada  
*/  
public String toString() {  
  
}
```

5. PROGRAMA

El programa se puede visualizar y descargar desde [este repositorio en Github](#).

CONCLUSIÓN

- El principio de análisis y descomposición de requerimientos específicos de lo que queremos lograr (objetivo), sigue siendo útil y la pieza clave para entender la lógica de java.
- El objetivo es importante, pero una vez que sabemos cuál es, debemos saber identificar y definir con qué información contamos o con qué información debemos contar para definir un orden de tareas a ejecutar.
- Una vez que sepamos o podamos medianamente tener claridad sobre los puntos anteriores seremos capaces o estaremos más cerca de poder construir diferentes piezas tipo lego que formarán parte de programas java.
- En el programa existen métodos y su implementación para mejorar, pero voy por un buen camino.

WEBGRAFÍA

ASIX1_M03 - UF2 CAS 5

<https://ciclesfp.gitlab.io/asix/m03prog/uf2/cas5/#intro>

[Consulta: 10 de marzo 2021]

Curso Open Webinars: Java 8 desde Cero

<https://openwebinars.net/academia/aprende/java/>

[Consulta: todo el tiempo de desarrollo de todo el cas5]

Cómo cambiar el nombre de un paquete en Eclipse - Artículos - 2021

https://es.laermfeuer.org/mudar-pacote-eclipse-como_46414-8284

[Consulta: 16 de marzo 2021]

Paquetes en Java: qué son, para qué se utilizan, y cómo se usan (con vídeo)

<https://www.campusmvp.es/recursos/post/paquetes-en-java-que-son-para-que-se-utilizan-y-como-se-usan.aspx>

[Consulta: 16 de marzo 2021]

Java toString overriding y Eclipse

<https://www.arquitecturajava.com/java-tostring-overriding-y-eclipse/>

[Consulta: 17 de marzo 2021]

Aprende a utilizar el método toString() en Java

[https://javautodidacta.es/metodo-tostring-java/#:~:text=El%20m%C3%A9todo%20toString\(\)%20en%20Java%2C%20como%20su%20propio%20nombre,de%20texto](https://javautodidacta.es/metodo-tostring-java/#:~:text=El%20m%C3%A9todo%20toString()%20en%20Java%2C%20como%20su%20propio%20nombre,de%20texto)

[Consulta: 17 de marzo 2021]

▷ Java Intermedio | Aprender Java desde Cero

<https://javadesdecero.es/intermedio/>

[Consulta: 17 de marzo 2021]

Printing an array in multiple lines

<https://stackoverflow.com/questions/22179557/printing-an-array-in-multiple-lines>

[Consulta: 11 Abril 2021]

Print an array elements with 10 elements on each line

<https://stackoverflow.com/questions/2661489/print-an-array-elements-with-10-elements-on-each-line>

[Consulta: 11 Abril 2021]

Java.util.Scanner.close() Method

https://www.tutorialspoint.com/java/util/scanner_close.htm

[Consulta: 12 Abril 2021]

Métodos en Java con Ejemplos | Java desde Cero
<https://javadesdecero.es/poo/metodos-con-ejemplos/>
[Consulta: 12 Abril 2021]