



**ESPE**  
UNIVERSIDAD DE LAS FUERZAS ARMADAS  
INNOVACIÓN PARA LA EXCELENCIA



SEDE  
SANTO DOMINGO

## **UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE**

### **SEDE SANTO DOMINGO DE LOS TSÁCHILAS**

#### **DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS**

#### **CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN**



PERIODO : 202351 Noviembre – Marzo 2024

ASIGNATURA : Programación Orientada a objetos

TEMA : Tarea 1

ESTUDIANTE : Chuico Navarrete Edith Liliana, Jimenez Davila Freddy  
Christian, Risco García Kennet Marcelo

NIVEL-PARALELO – NRC : Segundo A - Nrc 16129

DOCENTE : Ing. Verónica Martínez C., Mgs.

FECHA DE ENTREGA : 12 de noviembre de 202

**SANTO DOMINGO – ECUADOR**

# Fundamentos de la Programación

## Índice de contenido

|          |  |    |
|----------|--|----|
| 1        | Introducción .....                               | 4  |
| 2        | Objetivos .....                                  | 4  |
| 2.1      | Objetivo General .....                           | 4  |
| 2.2      | Objetivos Específicos .....                      | 4  |
| 3        | Desarrollo/ Marco teórico/ Practica .....        | 5  |
| 3.1      | Marco teórico .....                              | 5  |
| 3.1.1.   | Sistema de control de versionamiento (VCS) ..... | 5  |
| 3.1.1.1. | Software VCS: Git, Github .....                  | 5  |
| 3.1.2.   | Paradigmas De Programación .....                 | 7  |
| 3.1.2.1. | Transición De Paradigma .....                    | 8  |
| 3.2      | Desarrollo .....                                 | 28 |
| 4        | Conclusiones .....                               | 28 |
| 5        | Recomendaciones .....                            | 29 |
| 6        | Bibliografía .....                               | 29 |
| 7        | Anexos .....                                     | 30 |
| 8        | Legalización del documento .....                 | 30 |

## **Fundamentos de la Programación**

### **Tabla de ilustraciones**

|  |   |
|--|---|
| Ilustración 1/Repositorios de GitHub- Elaboración Propia.....      | 5 |
| Ilustración 2/ Características de GitHub- Elaboración Propia ..... | 6 |
| Ilustración 3/Transición de paradigmas- Elaboración propia.....    | 8 |

# Fundamentos de la Programación

## 1 Introducción

En el presente proyecto se indicarán de modo teórico y práctico el uso de las funciones y características principales de la programación orientada a objetos, esto con el fin de dar a entender de mejor manera el modo de operar y la importancia del uso de los mismos. Los temas que más se buscan resaltar son: clases, objetos, herencias, polimorfismo, abstracción, encapsulamiento, diagramas UML, y manejo de archivos y arreglos, con el objetivo de tener un mejor entendimiento y practica sobre ellos dado que estos serán los temas que mayormente se deberán de poner en práctica a lo largo del semestre.

Se explicará además la definición de los temas vistos en el sílabo y la similitud que tienen estos con los puntos evaluados en el semestre pasado, con el fin de mejorar y facilitar la comprensión de los nuevos conceptos. Cabe mencionar que, de igual modo, se trabajará en un nuevo entorno de desarrollo integrado (IDE) que será NetBeans, para así tener una mayor amplitud de los IDE en los que podremos operar y dominar en un futuro.

Además, se analizará también la instalación del nuevo IDE, anteriormente mencionado, en el que se va a trabajar, indicando el paso a paso de cómo realizarlo y las características del mismo. Es importante recordar además conceptos básicos y principales que se deben de investigar para poder comprender de mejor manera la información planteada, unos de estos conceptos es el de comprender el significado de la Programación Orientada a Objetos que es la continuación de los temas aprendidos en el semestre pasado y comúnmente su nombre se resumen en sus siglas POO, este paradigma de programación se basa principalmente en el concepto de clases y objetos, que como anteriormente se mencionó serán analizados en este documento

## 2 Objetivos

### 2.1 Objetivo General

Desarrollar un programa funcional en base a los temas estudiados en la unidad uno de programación orientada a objetos

### 2.2 Objetivos Específicos

Relacionar la teoría con la práctica, mediante la realización de nuestro programa

Explicar, de manera práctica y teórica, la importancia de cada una de las funciones y estructuras utilizadas.

Exponer los conocimientos obtenidos a lo largo de la realización de este proyecto.

### 3 Desarrollo/ Marco teórico/ Practica

#### 3.1 Marco teórico

##### 3.1.1. Sistema de control de versionamiento (VCS)

###### 3.1.1.1. Software VCS: Git, Github

Un software de tipo VCS es un sistema de control de revisiones o de fuentes, esta herramienta permite monitorizar y gestionar cambios en un sistema de archivos, además también permite compartir y editar cambios a proyectos o programas de otros usuarios.

GitHub: Es un software VCS fundado por Tom Preston-Werner, Chris Wanstrath y PJ Hyett en el año 2008, se creó con el objetivo de mejorar la colaboración en el desarrollo de software ya que permite publicar y compartir archivos entre diversos usuarios, permitiéndoles la colaboración y edición del mismo. Para ello proporciona herramientas para gestionar proyectos y realizar un seguimiento de los cambios en el código fuente.

GitHub se basa en el sistema de control de versiones Git. Los desarrolladores pueden clonar un repositorio (una copia de un proyecto) a sus máquinas locales, realizar cambios, y luego enviar esos cambios de vuelta al repositorio principal en GitHub. Git se encarga de gestionar las diferentes versiones del código, y GitHub proporciona una interfaz web y herramientas adicionales para facilitar la colaboración.

Este software de tipo VCS se creó debido a la necesidad del desarrollo de software colaborativo y del control de versiones, brindando así una mayor accesibilidad para acceder a un proyecto desde cualquier parte del mundo y poder realizar las modificaciones necesarias o algún comentario respecto al código.

#### Otras características

Nos da la opción de modificar un repositorio en público o en privado

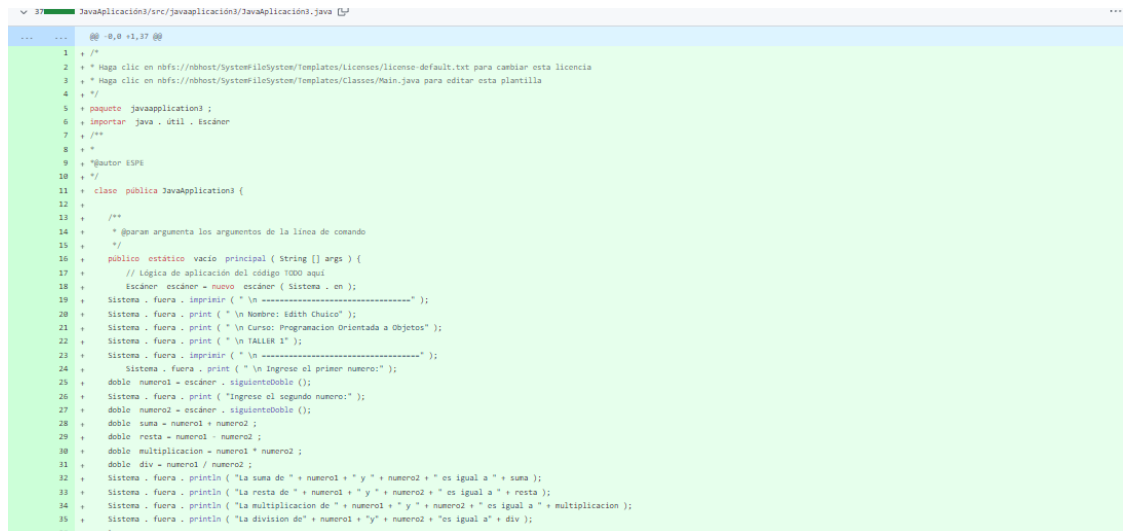


The image shows the GitHub interface for creating a new repository. At the top, it says 'Iniciar un nuevo repositorio'. Below this, a description states: 'Un repositorio contiene todos los archivos de su proyecto, el historial de revisiones y las discusiones de los colaboradores.' There is a text input field with the placeholder 'ponle nombre a tu nuevo repositorio...' and the username 'Edith Chuico /' is visible to the left. Below the input field, there are two radio button options: 'Público' (Public) with a globe icon and 'Privado' (Private) with a lock icon. The 'Privado' option is selected. Below these options, there is a green button labeled 'Crear un nuevo repositorio'.

Ilustración 1/Repositorios de GitHub- Elaboración Propia

## Fundamentos de la Programación

Permite visualizar el código dentro del software, sin necesidad de descargarlo



```
37 JavaAplicación3/src/JavaAplicación3/JavaAplicación3.java
...
1 //
2 * Haga clic en nbfs://nhost/system/temlates/licenses/license-default.txt para cambiar esta licencia
3 * Haga clic en nbfs://nhost/system/temlates/Classes/Main.java para editar esta plantilla
4 */
5 + paquete JavaAplicación3 ;
6 + importar java . util . Escáner
7 + /**
8 + *
9 + * Autor KSP6
10 + */
11 + clase pública JavaAplicación3 {
12 +
13 + /**
14 + * @param argumentos los argumentos de la línea de comando
15 + */
16 + público estático vacío principal ( String [] args ) {
17 + // lógica de aplicación del código irán aquí
18 + Escáner escáner = nuevo escáner ( Sistema . in );
19 + Sistema . fuera . imprimir ( " \n ===== " );
20 + Sistema . fuera . print ( " \n Nombre: Edith Chucó " );
21 + Sistema . fuera . print ( " \n Curso: Programación Orientada a Objetos " );
22 + Sistema . fuera . print ( " \n TALLER 1 " );
23 + Sistema . fuera . imprimir ( " \n ===== " );
24 + Sistema . fuera . print ( " \n Ingrese el primer número: " );
25 + doble número1 = escáner . siguienteDoble ();
26 + Sistema . fuera . print ( " Ingrese el segundo número: " );
27 + doble número2 = escáner . siguienteDoble ();
28 + doble suma = número1 + número2 ;
29 + doble resta = número1 - número2 ;
30 + doble multiplicación = número1 * número2 ;
31 + doble div = número1 / número2 ;
32 + Sistema . fuera . println ( " La suma de " + número1 + " y " + número2 + " es igual a " + suma );
33 + Sistema . fuera . println ( " La resta de " + número1 + " y " + número2 + " es igual a " + resta );
34 + Sistema . fuera . println ( " La multiplicación de " + número1 + " y " + número2 + " es igual a " + multiplicación );
35 + Sistema . fuera . println ( " La división de " + número1 + " y " + número2 + " es igual a " + div );
36 + }
```

*Ilustración 2/ Características de GitHub- Elaboración Propia*

### ¿Se puede usar fuera de la nube?

Si, ya que GitHub permite a los desarrolladores clonar los repositorios y así tener una copia local del repositorio. O por medio de GitHub Enterprise que permite hospedar repositorios de Git en nuestra propia infraestructura, ya sea por medio de servidores locales o de la nube.

### Tipos de archivos que permite GitHub

Si bien es cierto GitHub tiene un enfoque principal en el control de versiones de código fuente, por lo cual se indica que permite archivos que contengan estos códigos ya sea en diversos lenguajes de programación como Java, Python, C++, entre otras. Sin embargo, además de archivos de código, GitHub también admite una amplia variedad de distintos tipos de archivos, como, por ejemplo: Archivos de texto, Formatos de imágenes como (.png) o (.jpeg), Archivos de bases de datos como SQL (lenguaje de consulta estructurado), Archivos de presentaciones (.pptx), entre otros.

Además de GitHub, también es importante mencionar otras opciones de VCS como lo son:

**GitLab:** Esta plataforma además de ofrecer alojamiento de repositorios Git, cuenta con herramientas que ayudan a gestionar de manera correcta el ciclo de vida del desarrollo de software.

**GitKraken:** Se puede utilizar para interactuar con repositorios de distintas plataformas VCS, como GitHub, GitLab o Bitbucket. Además de ello, cuenta con una interfaz gráfica de usuario muy amigable.

**SourceForge:** Ofrece alojamiento de repositorios, seguimiento de problemas, foros y otras herramientas colaborativas.

Es necesario mencionar que cada herramienta cuenta con sus propias características y ventajas, el uso dependerá de las funciones que se deseen utilizar en el proyecto

### 3.1.2. Paradigmas De Programación

Un paradigma de programación es un modelo, estilo o plantilla de desarrollo de programas, para resolver problemas computacionales. Existen dos tipos de paradigmas que a su vez se dividen en otros tipos dentro de ellos, del siguiente modo:

Paradigma de programación: Modelo o patrón que se debe seguir

**Paradigma Imperativo:** Se indica la forma de llegar a la solución. Explica el ¿cómo resolver el problema? Y a su vez se divide en otros paradigmas, como:

1. Estructurado: Se centra en el control de flujo y la división de problemas en partes más pequeñas o en (subprogramas)
2. Orientado a Objetos: Se centra en el uso de herramienta como clases y objetos, con el objetivo de aproximarse lo más posible al modo de actuar del hombre
3. Reactivo: Observa flujos de datos asincrónicos y reacciona frente a su cambio

**Paradigma Declarativo:** Indica los pasos para llegar a la solución. Explica el ¿Qué hacer para resolver el problema? Y se divide en otros paradigmas, como:

1. Lógico: Se basa en la lógica matemática y utiliza el razonamiento deductivo para resolver problemas
2. Funcional: Se centra en el uso de variables y funciones, utiliza datos inmutables, es decir, que no pueden ser modificados.

### 3.1.2.1. Transición De Paradigma

Indica los distintos tipos de paradigmas que existen, y las funciones de cada uno, así como también el por qué es importante estudiarlos.



Ilustración 3/Transición de paradigmas- Elaboración propia

### 3.1.3. Entorno de Desarrollo

#### 3.1.3.1. Características e instalación CARACTERÍSTICAS

**NetBeans** es una herramienta poderosa y versátil que ha sido utilizada en una variedad de proyectos de desarrollo de software a lo largo de los años. Su amplio conjunto de características lo convierte en una opción atractiva para desarrolladores en diferentes dominios y tecnologías. Algunas de sus características más importantes son:

1. Amplia compatibilidad con varias plataformas como Windows, macOS y Linux.
2. Dispone de un soporte apto para diversos lenguajes de programación como Java, HTML, JavaScript, C++, entre otros.
3. Mayor facilidad al momento de la creación y gestión de proyectos debido a las diversas plantillas que ofrece.



## Fundamentos de la Programación

4. Incluye un diseñador de interfaces gráficas para aplicaciones Java.

**Visual Studio Code** es un entorno de desarrollo ligero y altamente configurable desarrollado por Microsoft. Aquí hay algunas de sus características clave:

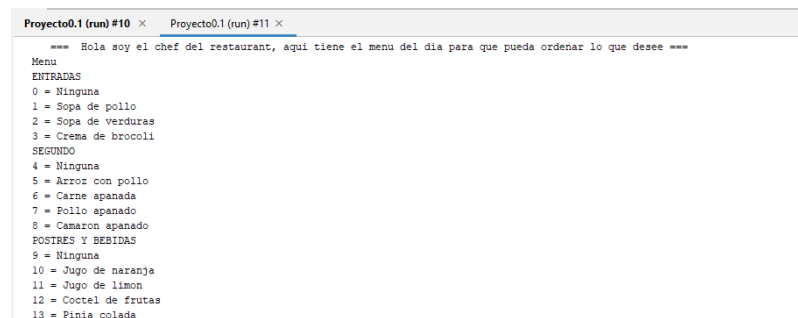
1. Proporciona un editor de texto avanzado con resaltado de sintaxis, autocompletado, sugerencias inteligentes y características de edición potente.
2. Es compatible con Windows, macOS y Linux, lo que permite a los desarrolladores utilizar la misma herramienta en diferentes sistemas operativos.
3. Amplias opciones de extensiones que permiten personalizar y ampliar las capacidades del editor.
4. Integración con Git para el control de versiones, lo que facilita el seguimiento y la gestión de cambios en el código fuente.

**Eclipse** es un entorno de desarrollo integrado (IDE) de código abierto que se utiliza principalmente para la programación en Java, aunque también admite otros lenguajes de programación a través de complementos. Aquí tienes algunas características clave de Eclipse:

1. Es una plataforma extensible que permite la integración de una amplia variedad de complementos y herramientas de desarrollo.
2. Eclipse es multiplataforma y está disponible para Windows, macOS y Linux, lo que permite a los desarrolladores utilizar la misma herramienta en diferentes sistemas operativos.
3. Proporciona un editor de código con funciones avanzadas como resaltado de sintaxis, autocompletado, navegación rápida por el código, etc.
4. Incluye un administrador de proyectos que facilita la creación, organización y gestión de proyectos de software. Los desarrolladores pueden importar proyectos existentes y trabajar con sistemas de control de versiones como Git.

### 3.1.3.3. Líneas de comando

Son instrucciones que se ingresan en una interfaz de línea de comandos (CLI) para realizar tareas en un sistema operativo o software.



```
Proyecto0.1 (run) #10 × Proyecto0.1 (run) #11 ×
=== Hola soy el chef del restaurant, aqui tiene el menu del dia para que pueda ordenar lo que desee ===
Menu
ENTRADAS
0 = Ninguna
1 = Sopa de pollo
2 = Sopa de verduras
3 = Crema de brocoli
SEGUNDO
4 = Ninguna
5 = Arroz con pollo
6 = Carne asada
7 = Pollo asado
8 = Camaron asado
POSTRES Y BEBIDAS
9 = Ninguna
10 = Jugo de naranja
11 = Jugo de limon
12 = Coctel de frutas
13 = Pina colada
```

*Ilustración 4/Líneas de comando- Elaboración propia*

### Otros tipos de interfaces de usuarios:

1. Interfaz de Usuario Gráfica (GUI): Utiliza elementos visuales como ventanas, iconos y botones para permitir la interacción del usuario con el sistema. Es la forma más común de interfaz de usuario en sistemas operativos y aplicaciones de software.
2. Interfaz de Usuario Natural (NUI): Busca imitar la forma en que las personas interactúan naturalmente con el mundo, utilizando gestos, movimientos corporales y voz en lugar de dispositivos de entrada tradicionales como el teclado y el mouse.

### 3.1.4. Revisión de Conceptos Generales de la POO

#### 3.1.4.1. Principios Generales de la Programación Orientada a Objetos

La programación orientada a objetos es un paradigma de programación enfocado mayormente en cómo se expresarían o realizarían acciones en la vida real, el objetivo de la Poo es tratar de aproximarse al modo de actuar del hombre. Este paradigma, cuenta con cuatro pilares fundamentales que son:

Abstracción: Define los atributos y métodos de una clase (De acuerdo a las características necesarias para la resolución de nuestro problema).

Herencia: Heredar atributos y métodos de las clases madres a las clases hijas.

Polimorfismo: Otorga la misma orden a distintos objetos y se espera que cada uno responda de manera distinta.

Encapsulamiento: Da la opción de hacer privada cierta información de un objeto.

#### 3.1.4.2. Definición de clases, objetos, atributos y métodos.

Clases: Es un molde, modelo o plantilla por el cual se crean objetos, y define sus características y comportamientos.

Objetos: Es el resultado de una clase, o el resultado del molde, estos objetos tienen atributos (datos) y métodos (funcionalidades), que pueden ser propios o comunes

Atributos: Son datos o variables que pertenecen a una clase y que sirven para representar las características de un objeto creado a partir de la misma clase

Métodos: Son funciones que pertenecen a una clase y que sirven para representar los comportamientos de un objeto creado a partir de la misma clase

### 3.1.5. Modelamientos de Clases y Objetos

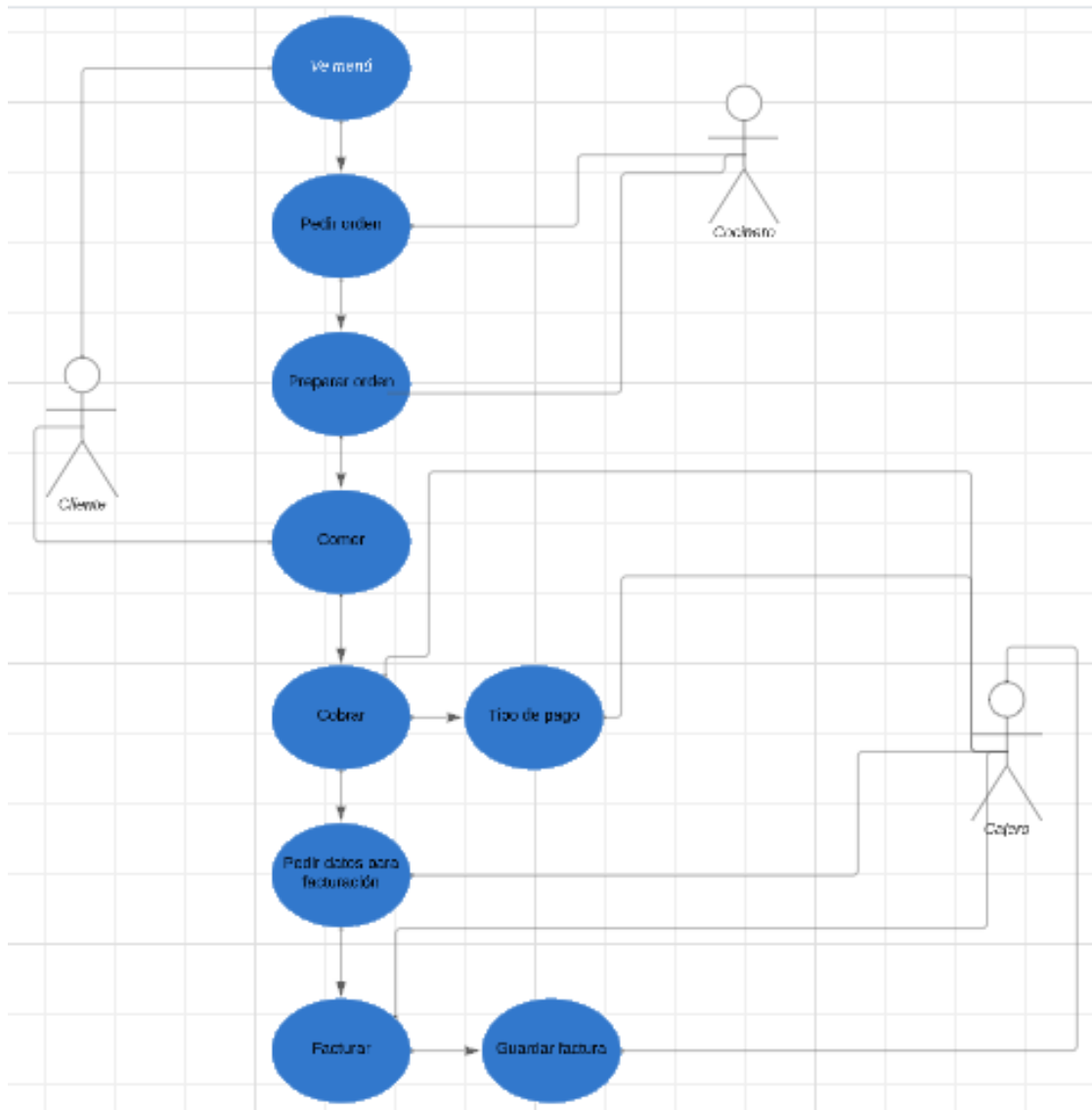
El modelamiento de clases se basa en la representación gráfica del programa que se va a codificar, esta acción es de gran importancia y apoyo al momento de programar, ya que nos ayuda a tener una idea clara de lo que se espera realizar y obtener.

#### 3.1.5.1. UML: Diagramas De Casos De Uso

Se utilizan para representar el cómo interactúan los usuarios de sistema

(actores) con el sistema desarrollado además de la forma, tipo y orden en que lo

hacen. Por ejemplo:



[https://lucid.app/lucidchart/f09e7a9c-b40b-43f5-8704-5b677aaef6ef/edit?viewport\\_loc=-2405%2C-430%2C2863%2C1302%2C0\\_0&invitationId=inv\\_d0e38daa-a491-44eb-83a6-c792095c4c1c](https://lucid.app/lucidchart/f09e7a9c-b40b-43f5-8704-5b677aaef6ef/edit?viewport_loc=-2405%2C-430%2C2863%2C1302%2C0_0&invitationId=inv_d0e38daa-a491-44eb-83a6-c792095c4c1c)

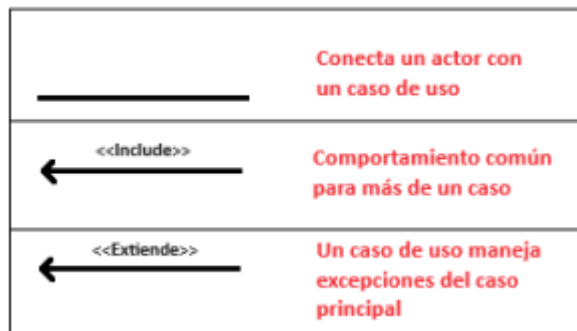
Como se puede visualizar en la ilustración, el diagrama de casos de uso cuenta con algunos elementos, los más relevantes y necesarios para comprender este tipo de UML son los siguientes:

**Actores:** Son los usuarios o entidades externas que se van a encargar de interactuar con el programa realizado, se representan con una figura humano.

**Casos de Uso:** Se puede definir como el lenguaje natural que se utilizaba en el paradigma estructurado, en donde, se explica de forma clara y precisa las acciones y las secuencias de relación que van a existir entre el sistema programado y el actor.

**Relaciones entre Casos de Uso:** Se expresan mediante flechas en forma de triángulo y puede contar con una etiqueta con la cual se puede definir el tipo de relación que puede darse. Existen dos tipos de etiquetas, las cuales son:

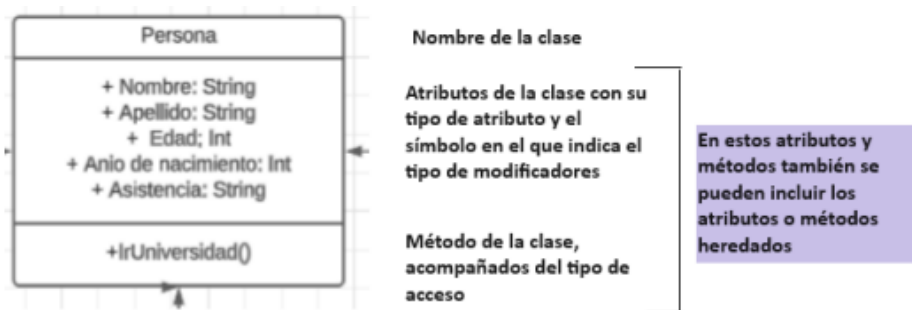
1. **Extends:** Un caso de uso brinda información a otro mediante la extensión de su funcionalidad, sin embargo, no es información relevante para el funcionamiento del programa como tal.
2. **Include:** Son funciones obligatorias que deben de cumplirse para el correcto funcionamiento del programa.



### 3.1.5.2. UML: Diagramas de Clases

Se utilizan para representar las estructuras estáticas de un sistema orientado a objetos, también se encargan de indicar las relaciones y propiedades existentes entre las clases del sistema.

Un diagrama de clases comúnmente se encuentra compuesto por tres partes, que son las siguientes:



En 25/Composición de un diagrama de clases. Elaboración propia

### Modificadores de acceso a miembros

Existen distintos niveles de acceso en función al modificador de visibilidad, esto en base al uso de encapsulamiento de los atributos o métodos a utilizar, este acceso se puede modificar de acuerdo a la conveniencia del programador, unos de estos niveles más utilizados son los siguientes:

Público (+)

## Fundamentos de la Programación

Privado (-)

Protegido (#)

Paquete (~)

Derivado (/)

Estático (subrayado)

### **Multiplicidad**

Define la cantidad de instancias de un objeto que participan en una relación, comúnmente se representa con dígitos bajo los conectores, unas de ellas son:

0 = La clase no desarrolla instancias (poco frecuente)

0..1 = Una o ninguna instancia

1 o 1..1 = Solo una instancia

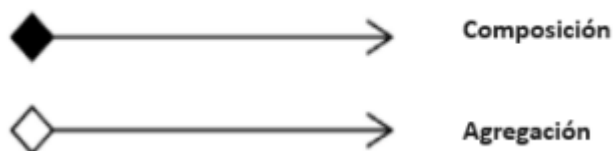
0..\* o \* = Ninguna instancia o varias con un valor máximo indefinido

1..\* = Una instancia o más con valor máximo indefinido

1..\*n = De uno a varios

\* \* = De varios a varios

### **Tipos de conectores**

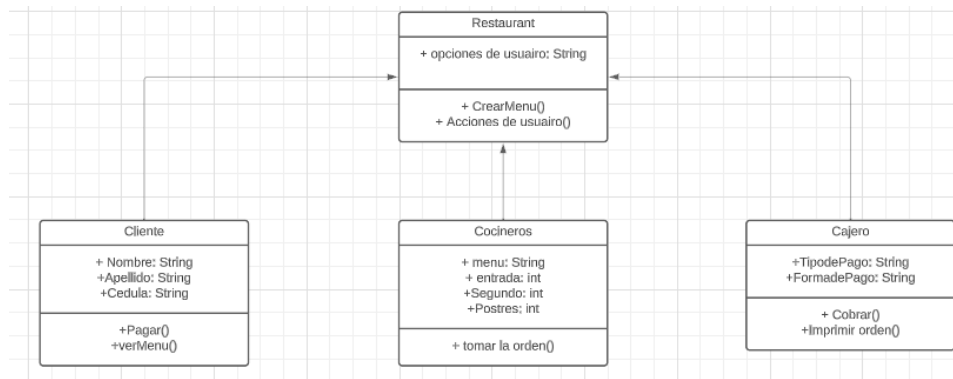


Composición: Un objeto no puede ser ese objeto sin otros objetos, todos dependen entre si

Agregación: Indica que un objeto forma parte, o le pertenece a otro objeto

Ejemplo de un diagrama de clases básico:

## Fundamentos de la Programación



### 3.1.5.3. Identificación de Clases de un Sistema, Uso Correcto de Identificadores

Los identificadores de clases son los tipos o estilos que se pueden y deben de utilizar para los nombres con los cuales se van a identificar y referenciar las clases en un código.

El identificador de clases **PascalCase**: Se trata de que cada palabra en el identificador comienza con mayúscula, por ejemplo: "ProgramaciónOrientadaAObjetos", también es muy importante utilizar nombres descriptivos al momento de nombrar una clase, para poder comprender las funciones y los objetos que va a realizar.

### 3.1.5.4. Modificadores de Acceso

Tal como su nombre lo indica, nos sirve para controlar los niveles de acceso a las clases métodos y variables de un programa. Los modificadores de acceso más comunes son:

**Public**: La clase o el método pueden ser accesibles desde cualquier otra parte

**Private**: No se puede acceder desde clases externas

**Protected**: Se puede acceder solo desde la misma clase, clases del mismo paquete o clases primas

Si no se especifica ningún modificador de acceso, se tiene como predeterminado el paquete público, en el que las clases, sus métodos y variables son accesibles solo dentro del mismo paquete

### 3.1.5.5. Implementación de Clases

Trata de la creación de clases que describen la estructura y el comportamiento de los objetos. Estos objetos, a su vez, representan instancias específicas de esas plantillas y se utilizan para modelar y resolver problemas en el desarrollo de software.

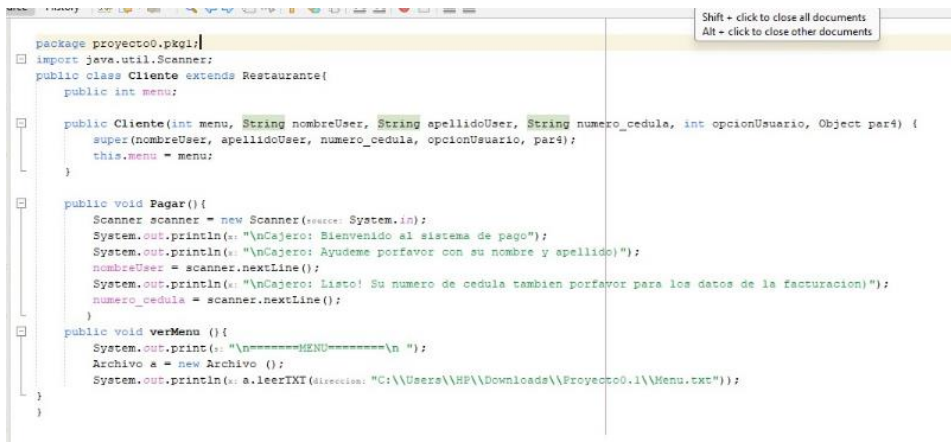
### Ejemplo de una clase y sus objetos, métodos y atributos

## Fundamentos de la Programación



ción 28/ Creación de una clase-Elaboración Propia

### 4.6 Código Limpio



### 3.1.6. Código limpio

Se basa en tener una correcta sintaxis y semántica sobre el código que se espera ejecutar, esto con el fin de hacerlo más entendible tanto para el programador, como para el usuario del programa

#### 3.1.6.1. Estándares de implementación, buenas prácticas de programación

Son normas que se deben de implementar al momento de programar con el fin de garantizar el buen entendimiento, la coherencia y la facilidad de mantenimiento del programa, algunas de estas normas son:

- Llevar a cabo la correcta escritura de variables, con el fin de hacer más entendible el código, para ello se pueden utilizar los estilos **camelCase** o **Snake\_case**,
- Tener un buen orden de nuestro código, ayudándonos de la correcta indentación del mismo en base a las estructuras de control que se vayan ingresando y las clases que se estén creando.

Estos estándares y normas empleados nos ayudan a tener un código limpio para de este modo hacer nuestro programa más eficiente y con una menor probabilidad de errores de sintaxis o de semántica.

## Fundamentos de la Programación

### 3.1.6.2. Atributos de calidad de código

Se trata de las características y propiedades que indican un buen funcionamiento del código de nuestro programa, estos atributos son necesarios para tener un buen entendimiento y eficiencia de nuestro código, para ello debe de ser legible y llevan un mantenimiento cada cierto periodo para verificar así su correcta ejecución.

```
public class Restaurante {
    String nombreUser;
    public String apellidoUser;
    public String numero_cedula;
    public int opcionUsuario;

    public Restaurante(String nombreUser, String apellidoUser, String numero_cedula, int opcionUsuario, Object pará) {
        this.nombreUser = nombreUser;
        this.apellidoUser = apellidoUser;
        this.numero_cedula = numero_cedula;
        this.opcionUsuario = opcionUsuario;
    }

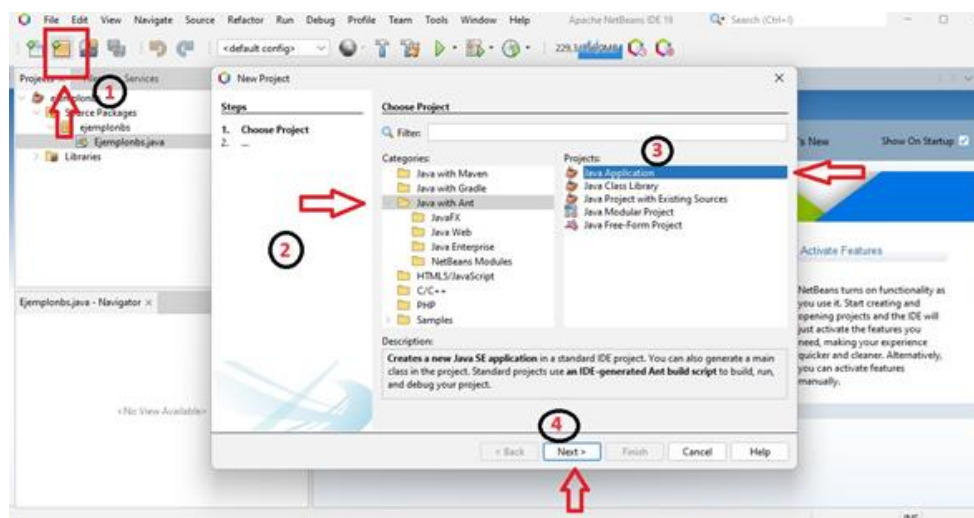
    public void Acciones () {
        Scanner scanner= new Scanner(System.in);
        Cliente cliente= new Cliente(scanner.nextLine(), scanner.nextLine(), scanner.nextLine(), scanner.nextLine());
        do {
            System.out.print("=====Usuario===== ");
            System.out.print("¿Que desea hacer?");
            System.out.print("\n1. Observar el menu de comida" + "\n2. Hacer la orden y pagar" + "\n3. Salir del restaurante");
            System.out.println("\nEscriba una opcion: ");
            opcionUsuario = scanner.nextInt();
            while (opcionUsuario != 1 && opcionUsuario != 2 && opcionUsuario != 3) {
                System.out.println("Ingrese solo una opcion valida: ");
                opcionUsuario = scanner.nextInt();
            }
            switch (opcionUsuario) {
                case 1:
                    cliente.verMenu();
            }
        }
    }
}
```

### 3.1.7. Estructura general de un programa

#### 3.1.7.1. Creación de un programa básico O.O

La estructura general de un programa basado en la orientación a objetos está centrada principalmente en sus cuatro pilares fundamentales, los cuales son: Herencia, Encapsulamiento, Polimorfismo y Abstracción, a continuación, se indicarán los pasos a seguir para la correcta creación de un programa orientado a objetos:

#### Paso 1: Creación de Nueva Carpeta



Elaboración propia

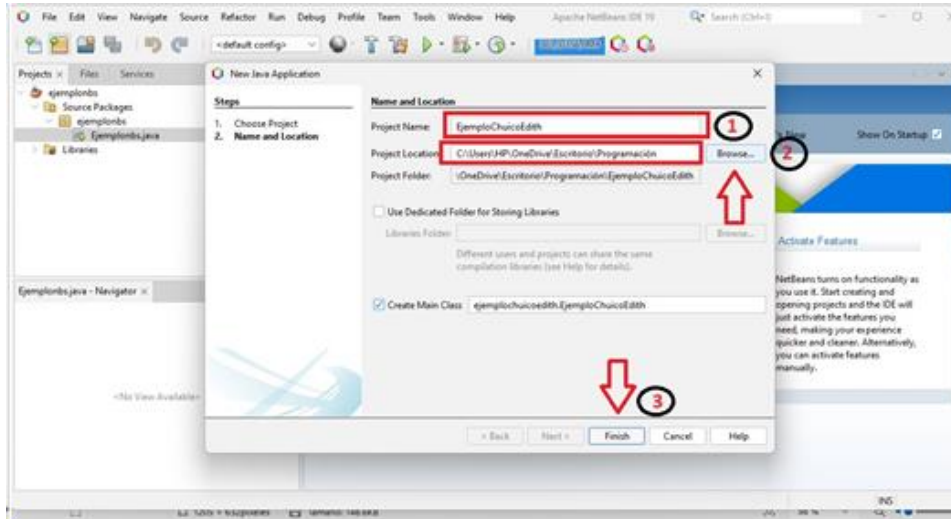
#### Paso 2:

1. Indicar el nombre de la carpeta

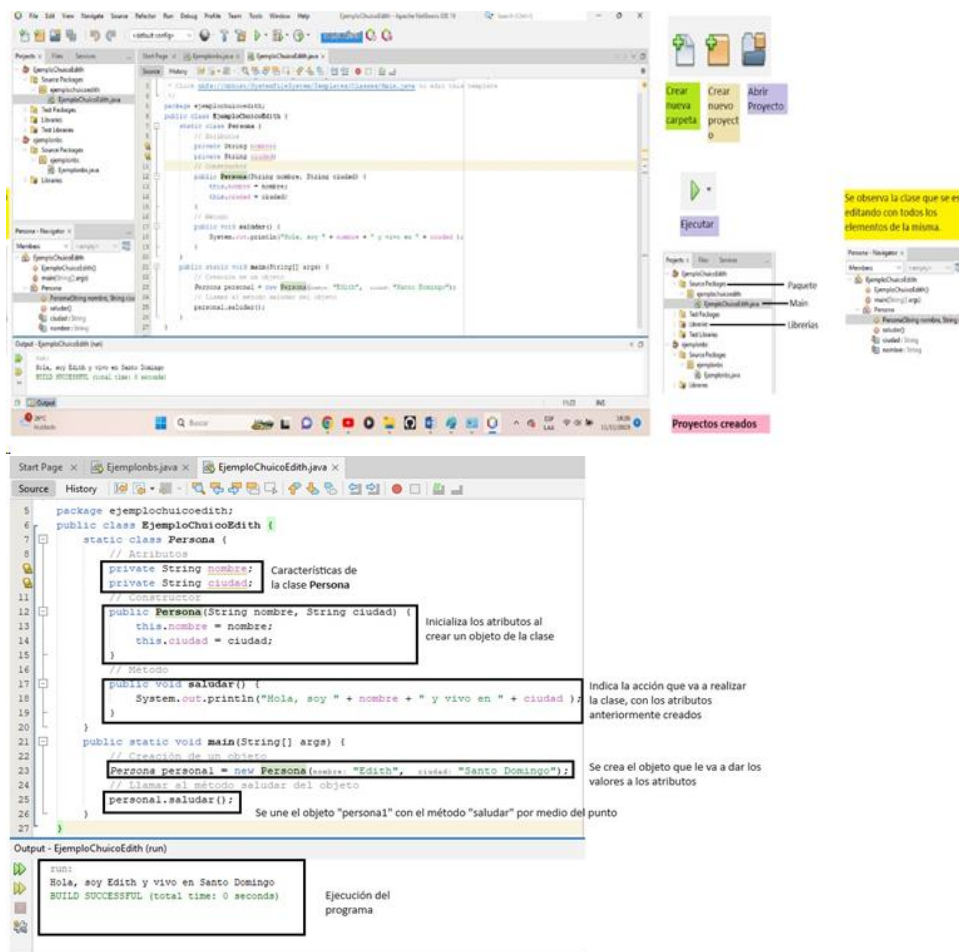


## Fundamentos de la Programación

2. Verificar el lugar en donde se desea que se guarde la carpeta
3. Dar click en “Finish”



### Paso 3: Código de NetBeans



### 3.1.7.2. Tipos de Datos, Primitivos y Referenciados

## Fundamentos de la Programación

Los tipos de datos que existen en la programación es la clasificación en la se organizan los valores en los cuales se van a ingresar los datos deseados, ya sean de tipo primitivo, referenciado o definido por el usuario, conocer el funcionamiento de cada uno de estos tipos de datos es de gran importancia, sin embargo, en este documento nos enfocaremos principalmente en lo que es el dato primitivo y el dato referenciado.

**Datos primitivos:** Son los datos comunes con los que se han ido trabajando en el último semestre, estos son los de tipo numérico (int o float), lógico (Boolean) y de carácter ( Char o String), se caracterizan por permitir almacenar o representar un único valor.

**Datos Referenciados:** Cumplen de mismo modo con el objetivo de almacenar valores, sin embargo, a diferencia de los datos primitivos, estos cuentan con la capacidad de representar a uno o más valores, con la única restricción de que estos valores deben de ser todos del mismo tipo, por ejemplo, **String [10] vehículos**, nos indica, que la variable vehículos va a tener la capacidad de almacenar 10 valores de tipo String

### 3.1.8. Lectura y escritura de datos por consola

#### 3.1.8.1. Entrada

La entrada de datos consiste en la opción que permite al programador pedir o crear información para una variable o atributo y trabajar con los valores asignados, considerando siempre una secuencia lógica y delimitando aspectos requeridos como la cantidad de datos o el tamaño del dato, etc.

#### 3.1.8.2. Salida

Por su parte la salida de datos como se indica es el resultado de un método en el cual se revelan o se imprimen los datos contenidos dentro de un atributo en la consola, estos pueden ser resultado de los datos puros del atributo o de la operación u otros métodos que hayan interactuado con ese atributo.

### 3.1.9. Excepciones

#### 3.1.9.1. Definición

Las excepciones son aquellas situaciones en que se compila el código y este no se ejecuta como se debería, para esto el programador toma medidas que ayudan a controlar estos errores y evitar que el programa deje de funcionar de manera normal, para esto se crean instancias que no buscan corregir, más bien buscan crear alternativas que puedan ofrecer una solución alternativa del problema mientras se puede corregir el error principal.

#### 3.1.9.2. Excepciones y errores

Las excepciones y errores no son un término semejante en la POO, ya que las excepciones son casos en los que se presenta una ejecución no prevista y con

alternativas de solución, mientras que los errores directamente no permiten la ejecución del programa, estos pueden provocarse por el código o directamente fallos externos relacionados al compilador u otros factores fuera del alcance del programador, en términos simples las excepciones son aquellas alternativas frente a casos de fallos en el programa que lo protegen de estos, mientras que los errores directamente no son corregibles fácilmente o directamente no tienen una solución.

### **3.1.9.3. Clases de excepción**

Las excepciones son clases dentro del propio lenguaje de programación y son llamados throwable, y existen 2 clases de excepciones principales en las cuales se presentan determinados casos:

Java.lang.exception: Estas excepciones son aquellas que deben declararse explícitamente en el programa ya que requieren de un método de captura para prever el comportamiento frente a un escenario determinado.

Java.lang.RuntimeException: Estas excepciones por su parte son un tipo que no debe declararse explícitamente, y son en general escenarios en los que se presentaría una falla de lógica por parte del usuario y no representar un gran problema para la ejecución y podrían solo ser informados y saltados al siguiente proceso, por lo general son provistos por el propio lenguaje.

### **3.1.9.4. Tipos de excepciones**

Existen 2 tipos de excepciones

Excepciones marcadas: Son aquellas excepciones en las cuales el programa ofrece la oportunidad de brindar información sobre un proceso incorrecto y enviar un mensaje o respuesta frente al problema por consola, esto se hace mediante un intento de captura lo que se define como un caso en el que ocurra este problema y como debe reaccionar el programa ante este, a continuación, la estructura que sigue este método.

```
try {  
  
    //proceso que provoque la excepción  
  
} catch () {  
  
    //ejecución alternativa  
  
}
```

Excepciones no marcadas: Tipos de excepciones en las cuales son de carácter implícito, es decir no requieren declaración y son comúnmente utilizados en procesos en los cuales las fallas son por errores lógicos y evitan que el programa finalice de manera imprevista.

### **3.1.9.5. Excepciones personalizadas**

Como su nombre lo indica son aquellas excepciones que pueden ser creadas por el programador esto sucede cuando las excepciones de java no pueden representar o quedan muy limitadas frente a una situación específica o tan solo para informar al usuario de un error preciso, para crear una de estas se realiza el siguiente proceso.

1. Crear una clase con la excepción.
2. Informar sobre la excepción con las condiciones que la provocaron
3. Manejar la excepción capturada en la ejecución.

### **3.1.10. Encapsulamiento**

#### **3.1.10.1. Definición**

Se define como la accesibilidad que se tiene para manipular o utilizar un objeto o una clase y todo lo que este contiene, es decir limitar quienes, y como se pueden utilizar objetos dentro y fuera de una clase, esto se hace con el fin de resguardar y preservar el código de posibles copias o eliminación del código.

#### **3.1.10.2. Clases**

Actualmente existen 3 tipos de encapsulamiento, estos tienen como fin el acceso completo, parcial y total de la clase y son:

**Público:** Permite el acceso total a una clase tanto a sus objetos como los atributos y métodos de cada uno, normalmente se escribe al inicio de cada proceso que sigue el programa, si no se escribe automáticamente la clase pasa a ser pública.

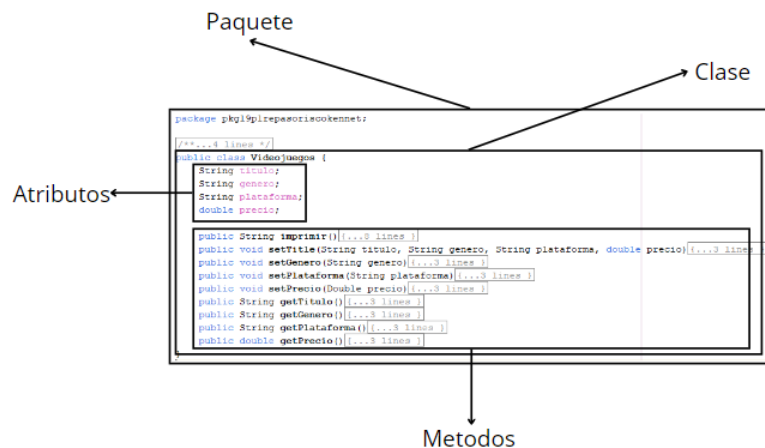
**Privado:** Restringe completamente el acceso y manipulación de una clase y todo lo que esta contenga a menos que se tenga autorización de uso de esta para ser utilizado en otros paquetes, clases e incluso dentro de su propio paquete, y se representa con un `private` al inicio de cada proceso, para utilizar a un atributo o método de esa clase privada se requiere de los métodos `setter` y `getter` explicados más adelante.

**Protegido:** Brinda un acceso y modificación limitado a ciertos aspectos de la clase en otra clase y restringe el acceso en otros paquetes.

#### **3.1.10.3. Paquetes**

Un paquete es por decirlo de alguna manera el contenedor de un programa, este contiene las clases que hacen que este funcione, estos sirven para un mejor control del código y la correcta identificación de los procesos que este contiene y puede realizar ya que se requiere de un `"package NombrePaquete;"` que indica que es un paquete y la función que puede realizar.

## Fundamentos de la Programación



### 3.1.10.4. Librerías/Bibliotecas, métodos static.

Las librerías son códigos prefabricados que ayudan al momento de programar, ya que estos pueden evitar el trabajo de tener que programar manualmente ciertas acciones dentro del código, agilizando el trabajo y en algunas ocasiones mejorando el código, para utilizar una librería se toma en cuenta la siguiente sintaxis:

Import paquete.clase;

Después de esta sentencia bajo de la línea del paquete donde será utilizada, se puede empezar a trabajar con los atributos y métodos de esta librería.

Los métodos static son funciones que permanecen de manera común dentro de una clase, es decir no son alterados por otras funciones que no las requieran, y estas no tienen la necesidad de utilizar instancias u objetos para desarrollarse.

### 3.1.11. Constructores

#### 3.1.11.1. Tipos de constructores

Un constructor en Java es aquel método que sirve para crear o iniciar la asignación de datos o valores a un atributo, estas cumplen su función mediante una entrada o asignación de datos, mas no son capaces de devolver datos, existen 2 tipos:

**Default:** Este constructor es creado por el propio lenguaje inicializando los valores establecidos en la programación del lenguaje, y puede ser modificado según se requiera.

**Parametrizados:** Utilizan parámetros o reglas que se deben cumplir para establecer datos o valores a un atributo y se pueda trabajar con estos valores posteriormente.

#### 3.1.11.2. Instanciación

## Fundamentos de la Programación

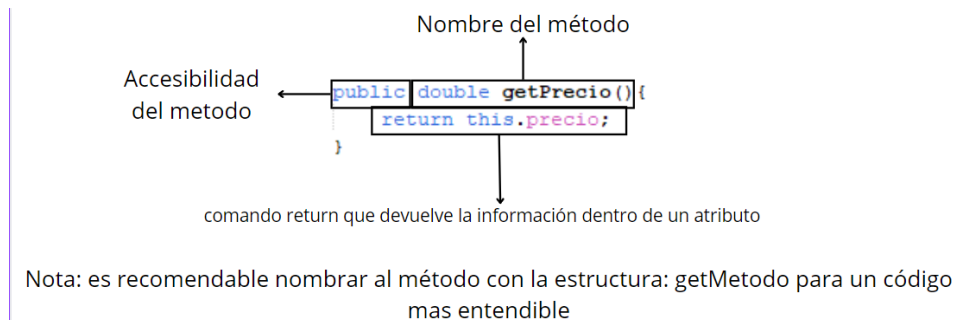
Las denominadas instancias no son más que los famosos objetos dentro de las clases y estos son quienes portan sus cualidades y funciones propias o compartidas (atributos y métodos), estas están muy ligadas a 2 pilares de la POO, la encapsulación porque responden a la modificación de sus atributos mediante un método de la clase, y el polimorfismo porque tienen una alternativa de solución a las demás instancias, ajustándose al funcionamiento y necesidades del programa.

### 3.1.12. Métodos getters y setters

#### 3.1.12.1. Definición e implementación

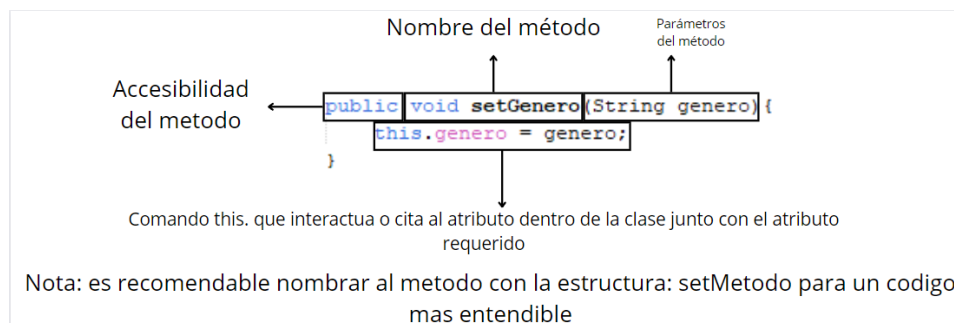
La estructura para utilizar este método es denominada como setter (set/obtener o mostrar), su sintaxis es muy parecida al de su contraparte getter, a diferencia de este es que no incluye dentro de sus parámetros al atributo únicamente devuelve a este, y su forma de escritura es la siguiente:

#### ILUSTRACIÓN estructura getter



En el caso de la programación orientada a objetos estas entradas son almacenadas como atributos de la clase, y se logran mediante un método llamado setter (set/ asignar o establecer) la cual permite al programa pedir datos al usuario mediante la consola y almacenarlos, la estructura de este método es la siguiente:

#### ILUSTRACIÓN estructura setter



### 3.1.13. Persistencia de Datos

## Fundamentos de la Programación

En la Programación Orientada a Objetos la persistencia de datos hace énfasis a la capacidad de poder guardar información más allá de la ejecución de un programa, es decir de tener almacenamiento externo, y que implica la posibilidad de recuperar esta información con sus objetos y características para ser nuevamente utilizados. Algunas de las formas más comunes de implementar la persistencia de datos son a través de archivos de texto o de una base de datos.

### 3.1.13.1. Manipulación de archivos

Conlleva el uso de objetos y términos de la Programación Orientada a Objetos para tener la posibilidad de interactuar con archivos sin la necesidad de depender de procesos o funciones, lo que permite encapsular de manera más eficaz y organizadas las clases y objetos creados.

Para crear un archivo se debe de llevar la siguiente sintaxis:

```
package com.mycompany.proyecto1;

import java.io.*;

public class Archivo {
    public String leerTXT(String direccion){//direccion del archivo
        String texto = "";

        try {
            BufferedReader bf = new BufferedReader (new FileReader (fileName: direccion));
            String linea;
            while ((linea = bf.readLine()) != null) {
                System.out.println(x: linea);
            }
        } catch (IOException e){
            System.err.println(x: "No se encontro el archivo");
        }

        return texto;
    }
}
```

### 3.1.13.2. Lectura y escritura de datos

Estructura para la lectura de datos:

```
public class LeerArchivo {

    public static void main(String[] args) {

        String nombreArchivo = "miArchivo.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(nombreArchivo))) {
            String linea;
            while ((linea = br.readLine()) != null) {
                System.out.println(linea);
            }
        } catch (IOException e) {
```

## Fundamentos de la Programación

```
        System.err.println("Error de lectura: " + e.getMessage());
    }
}
}
```

Estructura para la escritura:

```
public class EscribirArchivo {
    public static void main(String[] args) {
        String nombreArchivo = "miArchivo.txt";

        try (BufferedWriter bw = new BufferedWriter(new FileWriter(nombreArchivo))) {
            bw.write("Hola, esto es un ejemplo.");
            bw.newLine(); // Agrega una nueva línea
            bw.write("Segunda línea de texto.");
        } catch (IOException e) {
            System.err.println("Error de escritura: " + e.getMessage());
        }
    }
}
```

### **3.1.13.3. Lectura y escritura de objetos**

Lectura de objetos:

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class EscrituraObjetos {
    public static void main(String[] args) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream("personas.dat"))) {
            Persona persona1 = new Persona("Juan", 30);
            Persona persona2 = new Persona("María", 25);
```



## Fundamentos de la Programación

```
        oos.writeObject(persona1);
        oos.writeObject(persona2);
        System.out.println("Objetos escritos en el archivo.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

Ecritura de objetos:

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
```

```
public class LecturaObjetos {
    public static void main(String[] args) {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("personas.dat")))
        {
            while (true) {
                try {
                    Persona persona = (Persona) ois.readObject();
                    System.out.println("Objeto leído: " + persona);
                } catch (ClassNotFoundException e) {
                    e.printStackTrace();
                    break;
                }
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
}  
}  
}
```

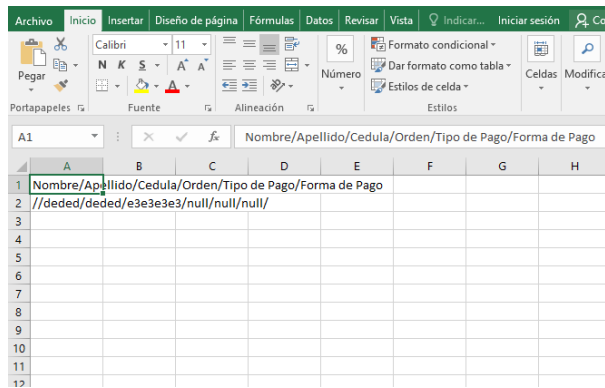
### 3.1.13.4. Formato de datos: CSV y JSON

#### CSV

Es un formato de archivo de texto el cual se puede manipular en cualquier entorno de programación que admita operaciones de lectura y escritura de archivos, siendo el CSV un tipo de fichero de texto plano que consta de varios valores separados por comas (o punto y coma), para que java lo pueda interpretar línea por línea.

```
package usuario;  
  
import java.io.FileWriter;  
import java.io.IOException;  
  
public class ArchivoCSV{  
    public static void crearArchivoCSV(String delim, String... info) {  
        final String NEXT_LINE = "\n";  
        try {  
            // Ruta relativa al directorio del proyecto  
            String file = "DatosCliente.csv";  
            FileWriter fw = new FileWriter(fileName: file, append: true);  
  
            // Encabezado del archivo CSV  
            fw.append(esq: "Nombre").append(esq: delim);  
            fw.append(esq: "Apellido").append(esq: delim);  
            fw.append(esq: "Cedula").append(esq: delim);  
            fw.append(esq: "Orden").append(esq: delim);  
            fw.append(esq: "Tipo de Pago").append(esq: delim);  
            fw.append(esq: "Forma de Pago").append(esq: NEXT_LINE);  
  
            // Información del usuario  
            for (String value : info) {  
                fw.append(esq: value).append(esq: delim);  
            }  
            fw.append(esq: NEXT_LINE);  
  
            fw.flush();  
            fw.close();  
            System.out.println("Archivo CSV creado con éxito en: " + file);  
        } catch (IOException e) {  
            public void crearInfoCSV() {  
                Scanner scanner = new Scanner(source: System.in);  
  
                System.out.println(s: "===== Creación de Info CSV =====");  
                System.out.print(s: "Ingrese su nombre: ");  
                nombreUser=scanner.nextLine();  
                System.out.print(s: "Ingrese su apellido: ");  
                apellidoUser=scanner.nextLine();  
                System.out.print(s: "Ingrese su n de cedula: ");  
                numero_cedula=scanner.nextLine();  
  
                // Resto de las operaciones para obtener la información...  
  
                // Llamada al método para crear el archivo CSV en el directorio del proyecto  
                ArchivoCSV.crearArchivoCSV(delim: "/", info: "/", info: nombreUser, info: apellidoUser, info: numero_cedula, info: orc
```

# Fundamentos de la Programación



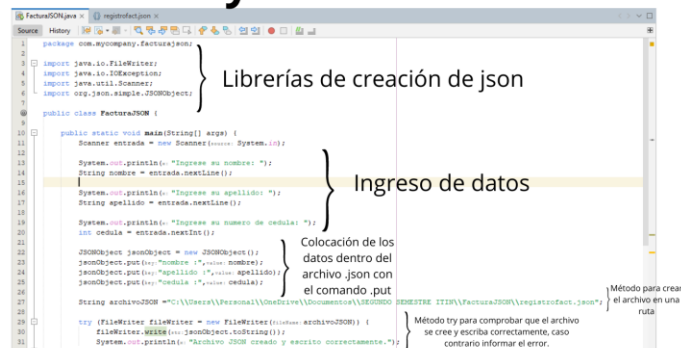
## JSON

Es un formato ligero de intercambio de datos que se utiliza para representar datos estructurados siendo fácil de leer, escribir, analizar y generar para las máquinas. Aunque la "J" en JSON proviene de JavaScript, JSON es un formato de datos independiente del lenguaje de programación y se utiliza en una amplia variedad de aplicaciones y entornos. Algunas características son:

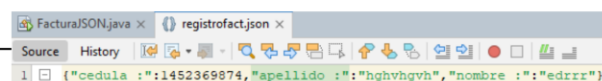
- Sintaxis simple.
- Soporta objetos, arrays, números, cadenas de texto, booleanos y valores nulos.
- Estructura anidada
- Independencia del lenguaje.

En la siguiente figura, se demostrará el proceso para crear y escribir un archivo .json.

## Creacion y escritura de un archivo .json



Resultado de la creación del  
archivo .json



### 3.1.13.5. Colecciones/Arraylist

En Java, las colecciones son estructuras de datos que permiten almacenar y manipular grupos de elementos. La interfaz principal para las colecciones en Java es la interfaz Collection, que extiende la interfaz Iterable y proporciona métodos para operaciones

comunes en colecciones, como agregar elementos, eliminar elementos y recorrer la colección.

Una de las implementaciones más utilizadas de la interfaz List es ArrayList, la cual es una clase que implementa una lista de arreglos dinámicos, lo que significa que puede cambiar de tamaño durante la ejecución del programa.

### 3.1.14. Arreglos y Colecciones

#### 3.1.14.1. Arreglos de datos primitivos

Un arreglo es un conjunto ordenado de variables del mismo tipo. Los arreglos pueden contener elementos de tipos de datos primitivos o referencias a objetos.

#### 3.1.14.2. Arreglos de Objetos

Es una estructura de datos que permite almacenar múltiples objetos del mismo tipo en una sola variable. En el caso de un arreglo de objetos, los elementos del arreglo son referencias a objetos en lugar de valores primitivos.

### 3.1.15. Arreglos y Colecciones

#### Asociación de agregación/composición: modelado e implementación.

##### Agregación

La asociación es un concepto fundamental que simplemente indica que hay una relación entre dos clases. Puede ser una relación débil o fuerte y puede o no tener una dirección. En términos sencillos, una asociación representa una conexión entre dos clases.

##### Composición

La composición es un tipo más fuerte de relación "todo a parte" en comparación con la agregación. En una composición, la parte (objeto) no puede existir independientemente del todo (objeto principal). Si el todo se destruye, las partes también se destruyen.

##### Dependencia

En Programación Orientada a Objetos (POO), la dependencia se refiere a la relación entre dos clases cuando un objeto de una clase utiliza servicios proporcionados por otro objeto de otra clase. En otras palabras, si una clase A depende de la clase B, significa que los objetos de la clase A utilizan o interactúan con los objetos de la clase B.

## 3.2 Desarrollo

## 4 Conclusiones

Se puede concluir en que la Programación Orientada a Objetos es un paradigma de programación que cuenta con mayor cantidad de funciones y elementos a diferencia de la programación estructurada que fue la que se estudió el semestre pasado, la mayor diferencia es que la POO se centra mayormente en realizar las resoluciones de los problemas identificados de un modo más aproximado a la forma de actuar del hombre,

## **Fundamentos de la Programación**

mediante encapsulación de datos y funciones similares, mientras que la Programación Estructurada se basa más en seguir procedimientos paso a paso.

Las clases son una manera más organizada de realizar un algoritmo, porque estas brindan una mejor gestión de las funciones que se pueden realizar, al ser el lenguaje utilizado diseñado para un paradigma implícito, ofrece otras ventajas al momento de desarrollar el algoritmo, opción de ejecutar tareas al mismo tiempo y en síntesis es muy similar al anterior lenguaje estudiado únicamente variando en su sintaxis, pero siendo más abstracto que el anterior.

El manejo de archivos es una estrategia que debe de ser muy tomada encuentra dentro del desarrollo de códigos ya que permite agilizar la organizar de datos, brindando la posibilidad de guardar la información y poder ser clasificada dentro de archivos, como lo son las clases y los objetos, los cuales podrán ser llamados en cualquier parte del proyecto, leídos y modificados en base a las necesidades del programador.

### **5 Recomendaciones**

Es muy importante, al momento de programar tomar en cuenta las fases del desarrollo de un programa, desde la identificación del problema que se desea resolver, hasta el mantenimiento y documentación del software, para evitar así algún error en nuestro programa y tener una noción clara de lo que se busca realizar con el código a crear. Además, es de gran ayuda la realización de los diagramas de clases o de los diagramas de casos de uso, para con ello saber de que como se deben de relacionar las clases que vayamos a crear y con qué fin de realizarán.

Es recomendable indagar o investigar sobre las excepciones ya que abordan una parte muy importante al momento de informar o incluso ayudar a tomar medidas ante casos específicos que se pueden presentar al momento de ejecutar al problema y así evitar fallos que desfavorezcan la experiencia de usuario.

Es aconsejable siempre tener entendido los principios de programación para poder desenvolverse de manera más eficaz cuando se trabaje en uno nuevo, debido a que muchas veces el único cambio que hay es la sintaxis, por lo demás los lenguajes de programación manejan códigos muy similares., además de tener que llevar siempre una organización dentro del desarrollo del código para que este sea más estructurado, como lo sería la agrupación de clases en un paquete en base a su relación.

### **6 Bibliografía**

Antón, C., & Alarcón, J. (2018). Introducción a Java. Babahoyo: CIDEPRO.

Barnes, D. (2007). Programación orientada a objetos con Java. Madrid, España: PEARSON EDUCACIÓN, S.A.

Dean, J., & Dean, R. (2009). Introducción a la Programación con Java. Ciudad de México: McGraw-Hill.

## Fundamentos de la Programación

Joyanes, L. (2013). Fundamentos Generales de Programación. Salamanca: McGraw-Hill.

Luis, J. (2008). Fundamentos de programación: Algoritmos, estructuras de datos y objetos. Madrid: McGraw-Hill.

Bugayenko, Y. (2014). Getters/setters. Evil. Period. Yegor Bugayenko.  
<https://www.yegor256.com/2014/09/16/getters-and-setters-are-evil.html>.

Belmonte. (s/f). Excepciones. Universitat Jaume I.  
<https://www3.uji.es/~belfern/Docencia/Presentaciones/ProgramacionAvanzada/Tema1/excepciones.html#1>

Uso de excepciones en Java. (2019, mayo 14). IfgeekthenNTTdata.  
<https://ifgeekthen.nttdata.com/es/uso-de-excepciones-en-java>.

Excepción personalizada en Java - Parzibyte's blog. (2021, abril 14). Parzibyte's blog; parzibyte. <https://parzibyte.me/blog/2021/04/14/excepcion-personalizada-java/>

Encapsulación: definición e importancia. (2022, abril 7). Formation Data Science | Datascientest.com. <https://datascientest.com/es/encapsulacion-definicion-e-importancia>

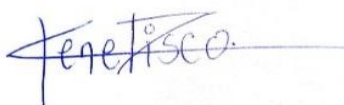
Coppola, M. (2023, marzo 16). Qué es un constructor en Java, sus tipos y cómo implementarlo. Hubspot.es. <https://blog.hubspot.es/webiste/que-es-constructor-java>

## 7 Anexos

### 8 Legalización del documento

Nombres y Apellidos: Kennet Marcelo Risco García

CI: 1729161297



Firma:

Nombres y Apellidos: Edith Liliana Chuico Navarrete

CI: 2350370967



Firma:

Nombres y Apellidos: Jiménez Dávila Freddy Cristhian.

CI: 2300153604.

Firma:

## Fundamentos de la Programación

Proceder