El futuro digital es de todos

MinTIC

PROGRAMACIÓN ORIENTADA A OBJETOS Y UML

Hechos QUE CONECTAN

Universidad Industrial de Santander

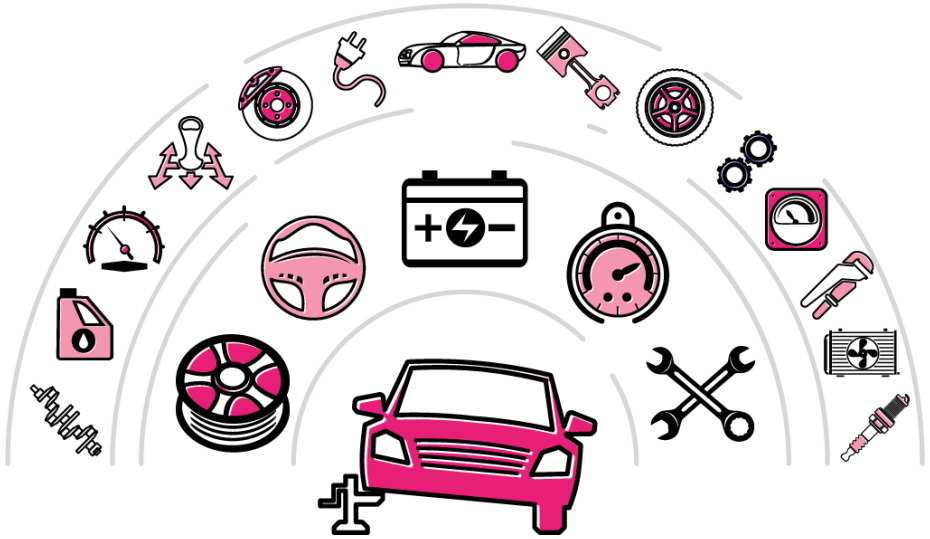
UIS

MisionTIC 2022

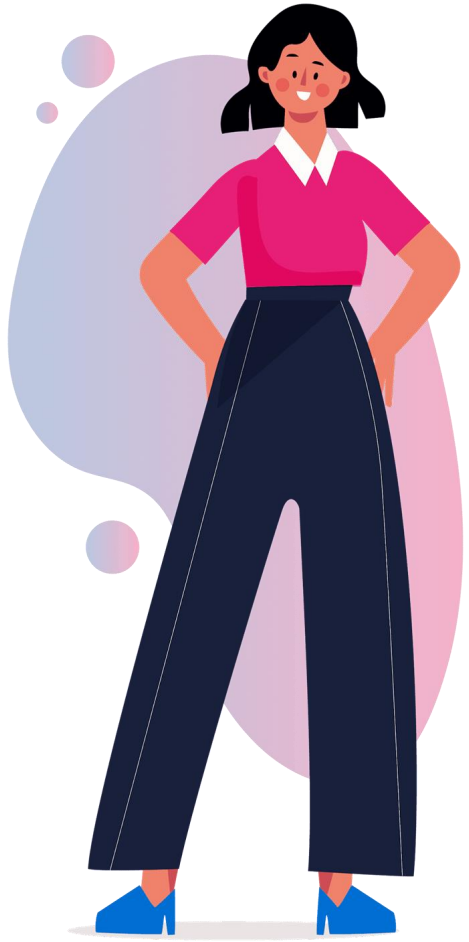
1. Introducción

Para empezar, te sugerimos ver el video titulado: ***¡Hablemos el lenguaje de la programación!***, *que se presenta a continuación.* En él, se hace un acercamiento a conceptos asociados a la Programación orientada a objetos o POO (por sus siglas en español). Veamos de qué se trata.

La programación orientada a objetos es un paradigma de programación, en donde se hace uso de objetos, los cuales son datos abstractos que contienen características inherentes de estos. Se puede decir que un objeto en programación, es una representación de un objeto real, que posee características propias. Este paradigma o modelo permite, mediante cierta analogía, poder representar objetos reales, que podemos encontrar en nuestro diario vivir, por medio de los objetos en determinado lenguaje de programación. Para un mejor entendimiento, pensemos en un vehículo, el cual está conformado por un conjunto de partes que se requieren para su buen funcionamiento (ver Fig. 1).



Cada una de estas partes puede pertenecer ya sea al sistema de frenado, al sistema de suspensión, al sistema de eléctrico-electrónico o al sistema de refrigeración, etc. Estos sistemas, a su vez, cumplen una función específica, que gracias a cada parte que lo compone, permite el buen funcionamiento del vehículo. Por ejemplo, el sistema de frenado permite disminuir de manera progresiva la velocidad del vehículo o detenerlo. Como podemos observar, en la vida real tenemos objetos, y estos se encuentran bien organizados por ciertos componentes, los cuales poseen determinadas funciones.



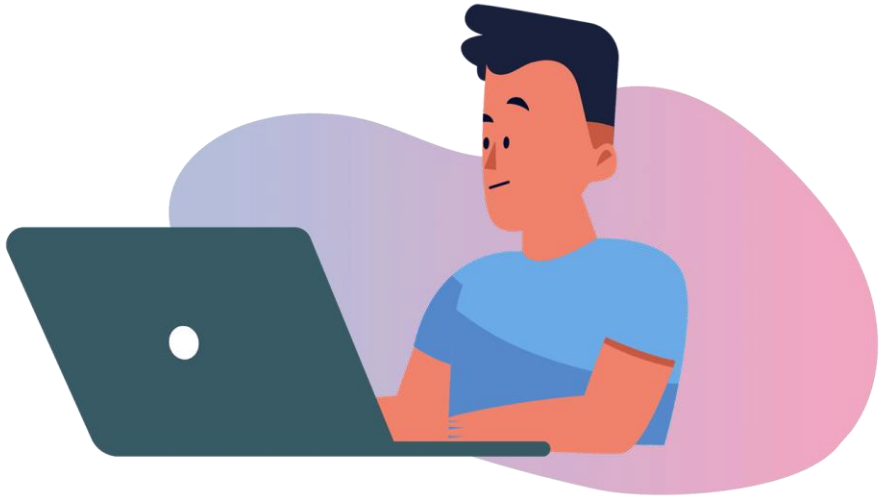
Dentro de todo este concepto de programación, existe además el paradigma de programación orientada a eventos. Dentro de este paradigma, la estructura del programa y su ejecución, los determina el usuario y no el programador al momento de desarrollar el programa. Como su nombre lo indica, este tipo de paradigma se basa en la construcción de programas a partir de una serie de eventos, los cuales tienen lugar dentro de la ejecución del programa en el momento en que el usuario decida hacer uso de ellos.

En un programa secuencial, el programador decide qué se va a ejecutar y cómo se va a ejecutar para que la tarea que se desea resolver se resuelva y se obtenga el resultado esperado. Este tipo de programación es la base del desarrollo de interfaces de usuario

Podemos imaginarnos, como ejemplo, el momento de usar un programa de edición de texto o un programa de edición de imágenes, donde cada uno de ellos tiene una interfaz de usuario y se encuentra programado de tal manera que cada botón que compone esa interfaz viene con una función específica o evento. La programación orientada a eventos, permite interactuar con el usuario a lo largo de todo el tiempo en que se esté ejecutando el programa; por su parte, el comportamiento del programa va a depender del uso que le del usuario y la manera en que este haga uso de cada uno de los eventos que lo compone.

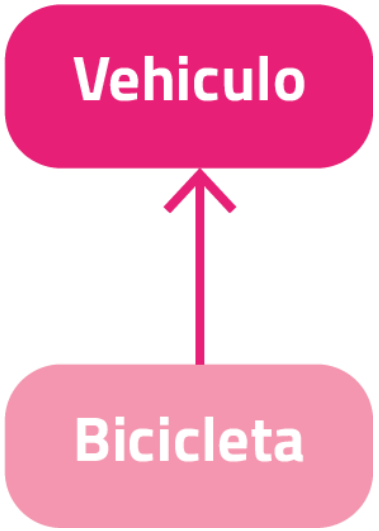
Quizá en el uso del programa para edición de imágenes, un usuario quiera abrir una imagen, aplicar un filtro, recortar la imagen y guardarla, pero otro usuario simplemente quiera abrir una imagen, modificar un poco los niveles de colores y guardarla. Los programas desarrollados bajo este paradigma, están constituidos por un bucle exterior, cuya función es la de recolectar los eventos solicitados por el usuario. Este bucle, que por lo general está oculto al programador, tiene la siguiente forma:

```
while(true) {
    switch(event) {
        case ctrl_s:
        case key_up:
        case mouse_click:
        case default:
    }
}
```



Como se puede observar, hay diferentes “interruptores” dentro de un bucle while, donde cada uno de estos “interruptores” es activado por el usuario mediante una orden que puede ser pasada usando cualquier periférico de una computadora o dispositivo electrónico. Estos “interruptores” pueden contener uno o varios eventos, y el usuario es quien decide el orden en que son ejecutados y cuándo deben ser ejecutados. Como ejemplo, si tenemos un editor de texto y queremos guardar la información que hemos editado, lo que se puede hacer es llamar al evento guardar mediante la combinación de teclas **CTRL + S.** el cua**l** inmediatamente envía la orden de guardar al programa. El programa toma el objeto que representa el documento como tal y almacena la información en un disco en determinado formato.

2. Lenguaje Unificado de Modelado – UML



El lenguaje unificado de modelado o UML, es un lenguaje de modelado de desarrollo que permite estandarizar la manera de visualizar el diseño de un sistema de software o aplicativo. El UML fue desarrollado por Rational Software entre 1994 y 1995, y hoy en día, es manejado y adoptado como un estándar por el Object Management Group (OMG). El UML puede ser usado en los siguientes niveles o etapas del desarrollo:

Conceptual: dentro del diseño en la etapa del problema, donde no existe una conexión fuerte con la codificación. En esta etapa, los diagramas de UML están más relacionados con el lenguaje humano.

Especificación: dentro de un diseño de software propuesto, donde existe una conexión fuerte con la codificación, ya que se pretende que los diagramas se conviertan en código.

Implementación: dentro de una implementación de software ya desarrollada, donde existe una conexión fuerte con la codificación. En esta etapa, los diagramas deben seguir ciertas reglas y semántica.

Podemos representar por medio de diagramas cualquier cosa, como por ejemplo, si consideramos la siguiente oración, "Una bicicleta es un vehículo", esta se puede representar como se muestra en la Fig. 4.

En el diagrama de la figura se indica que la bicicleta pertenece al conjunto de los vehículos, lo que quiere decir, que una bicicleta es un caso particular de vehículos.

Las características de un UML son:

Visualizando: Quizá alguno de nosotros haya necesitado antes de resolver un problema hacer una especie de garabato en una hoja de papel para tener una guía de cómo resolver el problema. De la misma manera, los desarrolladores requieren guías visuales para la solución de problemas complejos o para tener una trazabilidad del proceso de desarrollo de una aplicación. Lo primero que se debe es generar una ficha gráfica de la situación a la cual nos enfrentamos, ya que esto facilita la comprensión y la solución del problema. Un UML es un lenguaje formal y estandarizado que permite llevar a otro nivel estas ayudas gráficas, de tal manera que, cualquiera que sepa este lenguaje, pueda entender lo que se quiere transmitir.

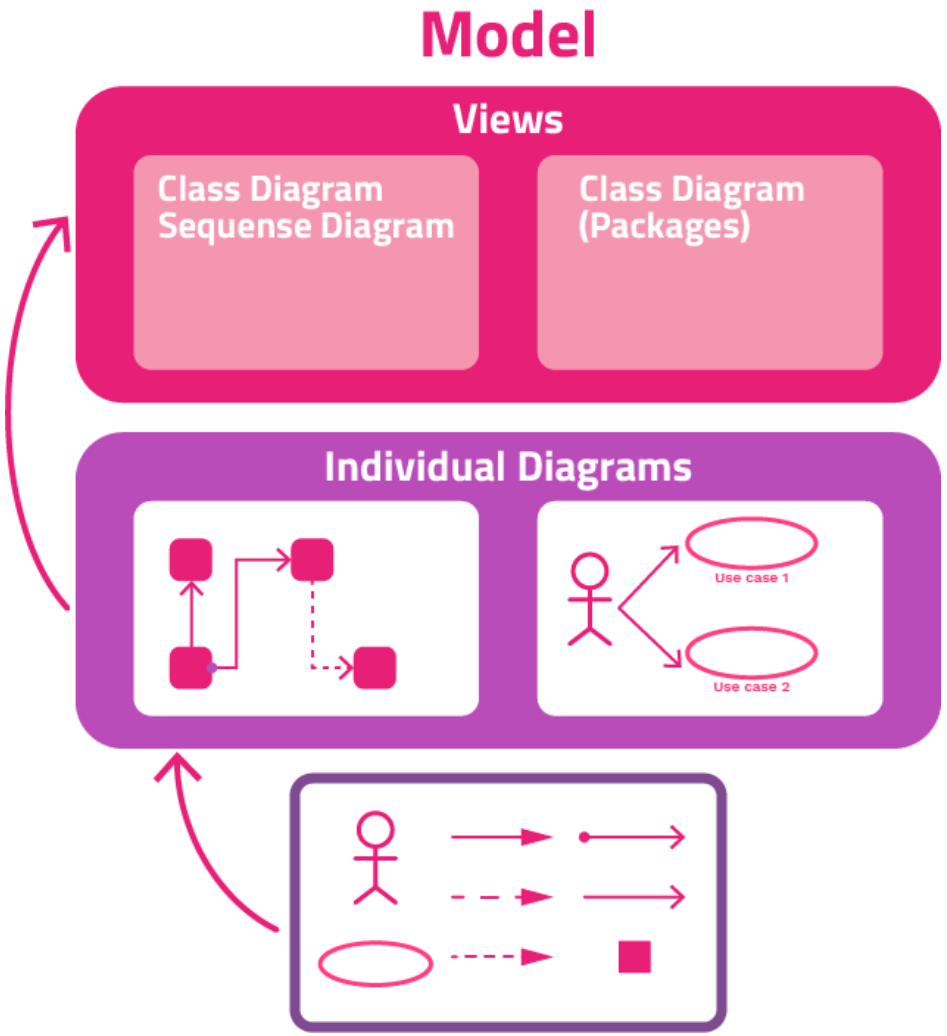
Especificando: El mecanismo de comunicación para transmitir las ideas debe ser precisa y común. Un UML permite, de manera fácil y correcta, transmitir las especificaciones requeridas.

Construyendo: Por la simple razón de que el UML es un lenguaje formal compuesto por determinadas reglas y una sintaxis fija, se pueden construir herramientas que interpreten y mapean nuestros modelos a cualquier lenguaje de programación.

Documentando: El uso correcto del UML produce como resultado una serie de documentación, la cual puede ser útil como modelo de nuestro sistema de software.

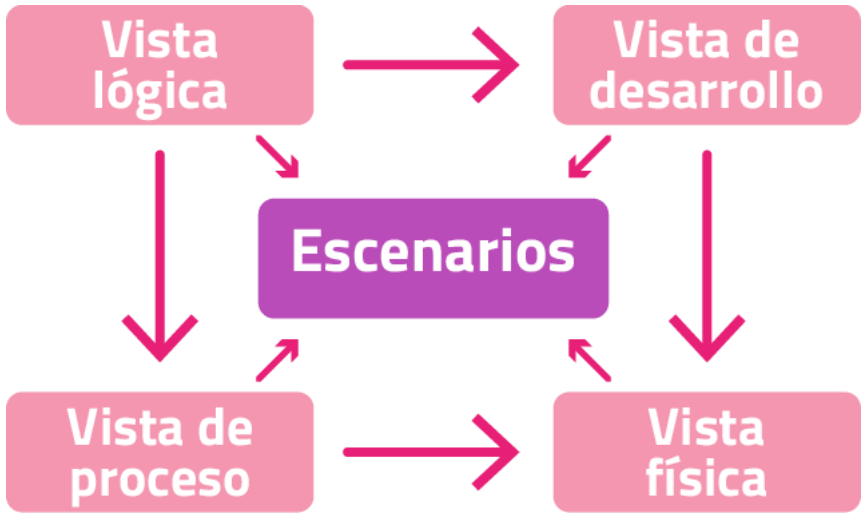
Para poder implementar el UML en un proyecto de software, primeramente se debe saber que existe un esquema jerárquico para la construcción de los modelos. Los modelos se encuentran compuestos de las vistas de nuestro sistema de software, las cuales están conformadas por diagramas individuales, que a su vez están compuestos de ciertos elementos fundamentales.

Los diagramas más comunes son los diagramas de clase, los cuales describen la relación estructural entre las clases que componen nuestro sistema. La combinación de un diagrama de clase y otro diagrama que comunique la dinámica del sistema, es a lo que se conoce como una vista, las cuales describen de manera particular el sistema.



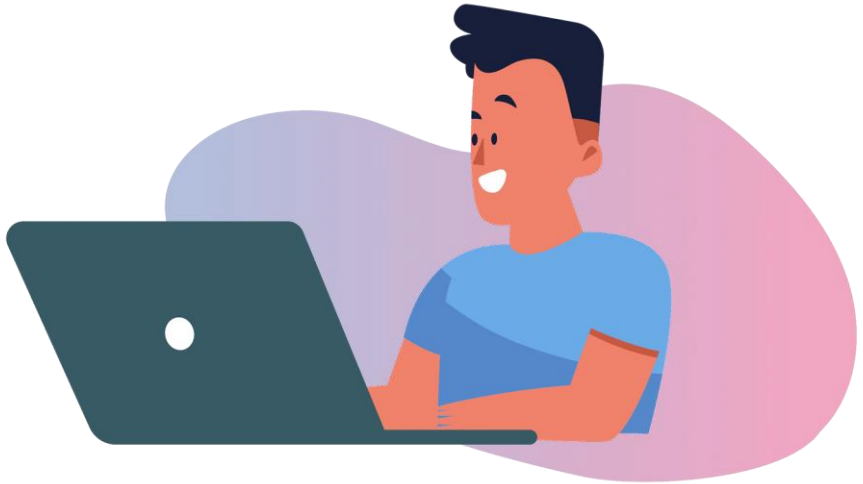
Un modelo para describir la arquitectura del sistema de software es el *modelo de vistas 4+1*, diseñado por Philippe Kruchten. Se pueden crear modelos completos de nuestro sistema o aplicación, los cuales están compuestos por múltiples vistas, las cuales representan el sistema desde perspectivas distintas, cada una de ellas relacionadas con diferentes individuos asociados con la iniciativa de desarrollo.

La vista lógica tiene que ver con la funcionalidad de la aplicación y está relacionada con los diseñadores y el usuario final o consumidor. La vista de desarrollo está asociada con la gestión de software y está relacionada con los desarrolladores. La vista del proceso está relacionada con el rendimiento, la escalabilidad y el desempeño del sistema , así como con los integradores del sistema. La vista física se relaciona con la topología del sistema, entrega, instalación y telecomunicación y está relacionada con los ingenieros del sistema.



3. Objetos y clases: Conceptos, diseño (notaciones), implementación

Los objetos y las clases son la base de la programación orientada a objetos. Los objetos son instancias de una clase, y las clases son prototipos dentro de los cuales se definen atributos y métodos, por medio de los cuales se construyen los objetos. Los atributos son variables o datos definidos dentro de la clase, los cuales la caracterizan. Los métodos son funciones o subrutinas que representan acciones que operan sobre los atributos o datos de una clase, y es a través de estos métodos, por medio del cual, los objetos se comunican entre sí. Los métodos permiten a los objetos enviar y recibir mensajes hacia o desde otros objetos.



Clases

Las clases en Java son almacenadas en ficheros con extensión *.java*, y tienen una estructura bien definida. Las clases en Java están compuestas por una declaración del paquete al cual pertenece la clase. Seguido de la declaración del paquete, va el importe de paquetes externos, seguido de los comentarios. Luego de los comentarios, viene la definición de la clase que se hace mediante la palabra reservada *class*. Posterior a la definición de la clase, son ubicadas las variables o atributos de la clase. Y seguido de esto, van los constructores de la clase, donde van ubicadas las características iniciales que se deseen para los objetos que se deriven de la clase. Por último, van los métodos o funciones. A continuación, se muestra un esquema de las componentes de una clase:

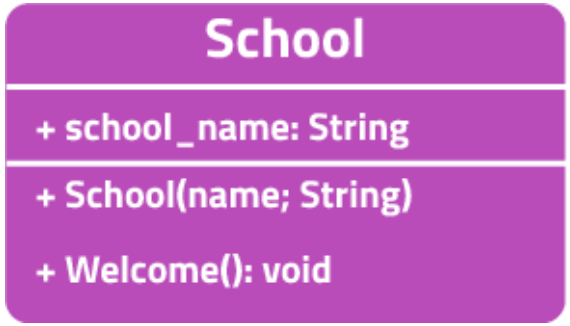
```
paquetes
importe de paquetes externos
comentarios
definición de la clase
    constantes
    atributos
    constructores
    métodos
```

Un ejemplo de una clase en Java, llamada School, es el siguiente

package school;

```
/**
*
* @author
*/
public class School {
    public String school_name;
    public School(String name) {
        this.school_name = name;
    }
    public void Welcome() {
        System.out.print("Welcome: " + this.school_name + "\n");
   }
}
```

donde su representación por medio de UML es de la siguiente manera:



A partir del ejemplo anterior, podemos identificar cada una de las componentes. Al inicio de la clase podemos observar el paquete al cual pertenecen, el paquete school. Seguido se puede observar un bloque de líneas de comentarios. Luego se define la clase, llamada School, usando la palabra reservada class.

Dentro de la clase, al inicio, se encuentran sus atributos. Podemos observar un atributo llamado school_name de tipo *String.* Se encuentra también el constructor, que lleva como nombre el mismo nombre de la clase, y que puede o no recibir parámetros de entrada. En este caso, el constructor recibe un parámetro de entrada llamado name de tipo *String*. Los constructores son usados para la creación de objetos, y su función es la de inicializar los atributos para dichos objetos, en este ejemplo el atributo school_name.

Adicionalmente, una clase puede tener métodos, los cuales son funciones que contienen ciertas acciones que operan sobre los objetos que se crean a partir de ella. En este ejemplo, hay un solo método llamado Welcome de tipo void, cuya función es la de visualizar un mensaje a la salida. Dentro de una clase, se puede definir un método llamado *main*, el cual es el punto inicial de todo programa en Java, su sintaxis es siempre public static void(String[] args), y recibe como parámetros de entradas argumentos almacenados en la variable args, que es de tipo arreglo de cadena de caracteres.

Para usar los atributos y métodos dentro de la misma clase a la cual pertenecen, se hace a través de la palabra reservada this, la cual hace referencia al objeto actual. Si se desea usar los atributos y métodos en otras clases, por lo general, se hace mediante la creación de objetos o instancias de la clase a la cual pertenecen esos atributos y métodos.

Objetos

Los objetos son instancias de una clase que se pueden ver como una abstracción de datos asociados a una dicha clase. Los objetos son creados usando la siguiente sintaxis:

```
Clase objeto = new Clase();
```

donde Clase es el nombre de la clase, a partir de la cual se quiere crear el objeto y el nombre que se le quiere dar al objeto. Para definir los objetos, es usada la palabra reservada *new* seguida del nombre de la clase, indicando que se va a crear un nuevo objeto que pertenece a esa clase. Por ejemplo, para definir un objeto de la clase anterior se puede hacer de la siguiente manera:

```
School school = new School("School A");
```

Donde se puede ver que se crea el objeto school que pertenece a la clase School. En este caso, se indica que se crea un nuevo objeto a partir de la clase School, la cual recibe como parámetro de entrada un dato tipo String. Si se desea hacer uso de un atributo o un método definido dentro de esa misma clase, se puede hacer mediante el operador "." a través del objeto creado, esto es:

```
objeto.metodo();
```

Como por ejemplo, para el caso de la clase School, si se desea implementar el método Welcome() se procede de la siguiente manera:

```
school.Welcome();
```

lo que resultaría en la impresión del mensaje "Welcome: School A". De esa manera y con esta sintaxis, son generados y usados los objetos en Java.

Paquetes

Los códigos en Java están contenidos en paquetes, los cuales pueden verse como directorios. Y estos paquetes pueden estar organizados de manera jerárquica en nuestro sistema de software. Un ejemplo de la estructura de paquetes en un programa en Java es el siguiente:

```
School
Source   Packages
     school
          School.java
     user
          User.java
```

Donde se observa que el programa **School** tiene dos paquetes, uno llamado **school** y otro llamado **user**. Cada clase o un conjunto de estas, perteneciente a determinado paquete, son importadas mediante el uso de la palabra reservada import de la siguiente manera:

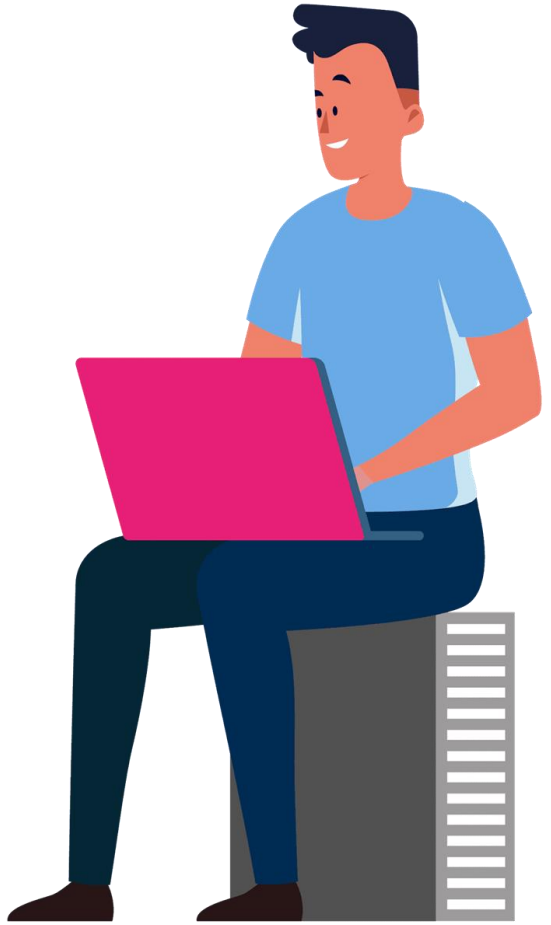
```
import path.java.package.Class;
```

Considerando la estructura de la clase School, anteriormente mostrada, se realizaron modificaciones en su estructura, de tal manera que quedó como se muestra a continuación:

```
package school;
import user.User;
import java.util.List;
import java.util.ArrayList;
/**
*
* @author
*/
public class School {
    public String school_name;
    public int total_students;
    List<User> users = new ArrayList();
    public School(String name) {
        this.school_name = name;
        this.total_students = 0;
    }
    public void Welcome() {
        System.out.print("Welcome: " + this.school_name + "\n");
    }
```

```
public void save_user(User user) {
        this.users.add(user);
        this.total_students += 1;
        System.out.print("New user saved.\n");
    }
    public void students_list() {
        System.out.println("--------------------------------");
        System.out.println("---------- USERS LIST ----------");
        System.out.println("--------------------------------");
        users.forEach(user -> {
            user.info();
        });
        System.out.println("--------------------------------");
    }
}
```

A partir de lo cual, queremos mostrar que se pueden realizar distintos tipos de importe. Tenemos un importe de un paquete local, como es el caso de la clase User del paquete user, y un importe de un paquete de java, como es el caso de las clases List y ArrayList, pertenecientes al paquete Java.util. Adicionalmente, se agregan otros elementos que serán explicados más adelante. La clase User del paquete user es una clase que tiene la siguiente forma:

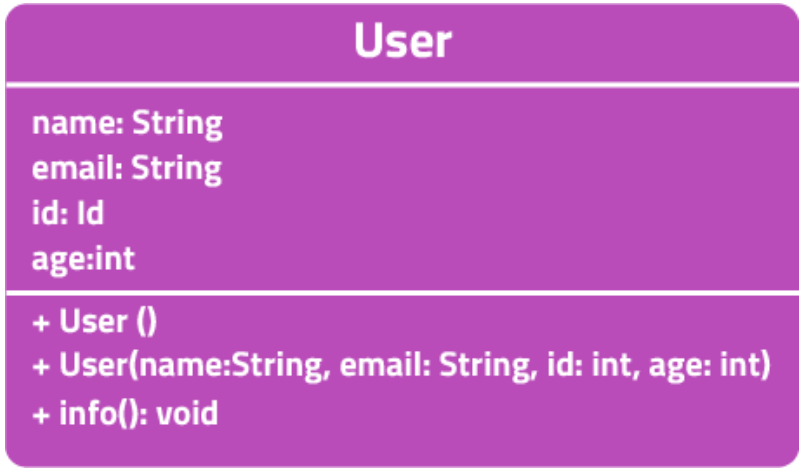


```
package user;

/**
*
* @author
*/
public class User {
    String name;
    String email;
    int id;
    int age;
    public User() {}
    public User(String name, String email, int id, int age) {
        this.name = name;
        this.email = email;
        this.id = id;
        this.age = age;
    }
```

```
public void info() {
        System.out.println("--------------------------------");
        System.out.println("----------- USUARIO ------------");
        System.out.println("--------------------------------");
        System.out.println("Full name: " + this.name);
        System.out.println("Email: " + this.email);
        System.out.println("ID: " + this.id);
        System.out.println("Age: " + this.age);
      System.out.println("--------------------------------");
    }
}
```

Donde se pueden ver cuatro atributos, name, email, id y age, un constructor que inicializa los atributos, y un método llamado void_info(), que visualiza un mensaje de salida con la información del usuario. En una representación UML, se puede mostrar la relación que existe entre estas dos clases mediante una composición, esto de la siguiente manera:



Donde se tienen la representación de los dos paquetes, *school* y *user,* y los diagramas de clase de cada una de las clases pertenecientes a estos paquetes, y la relación que hay entre ellas.

Vamos a mirar un ejemplo donde se aplican varios métodos , la idea es revisar el diagrama de clases y poder interpretar qué hace el programa, luego se va a correr el programa.

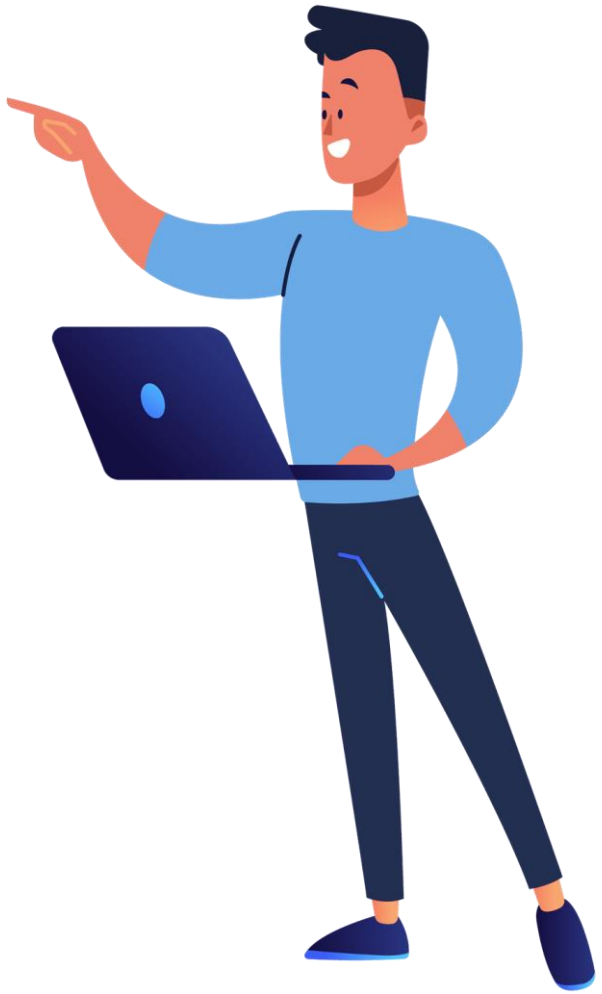
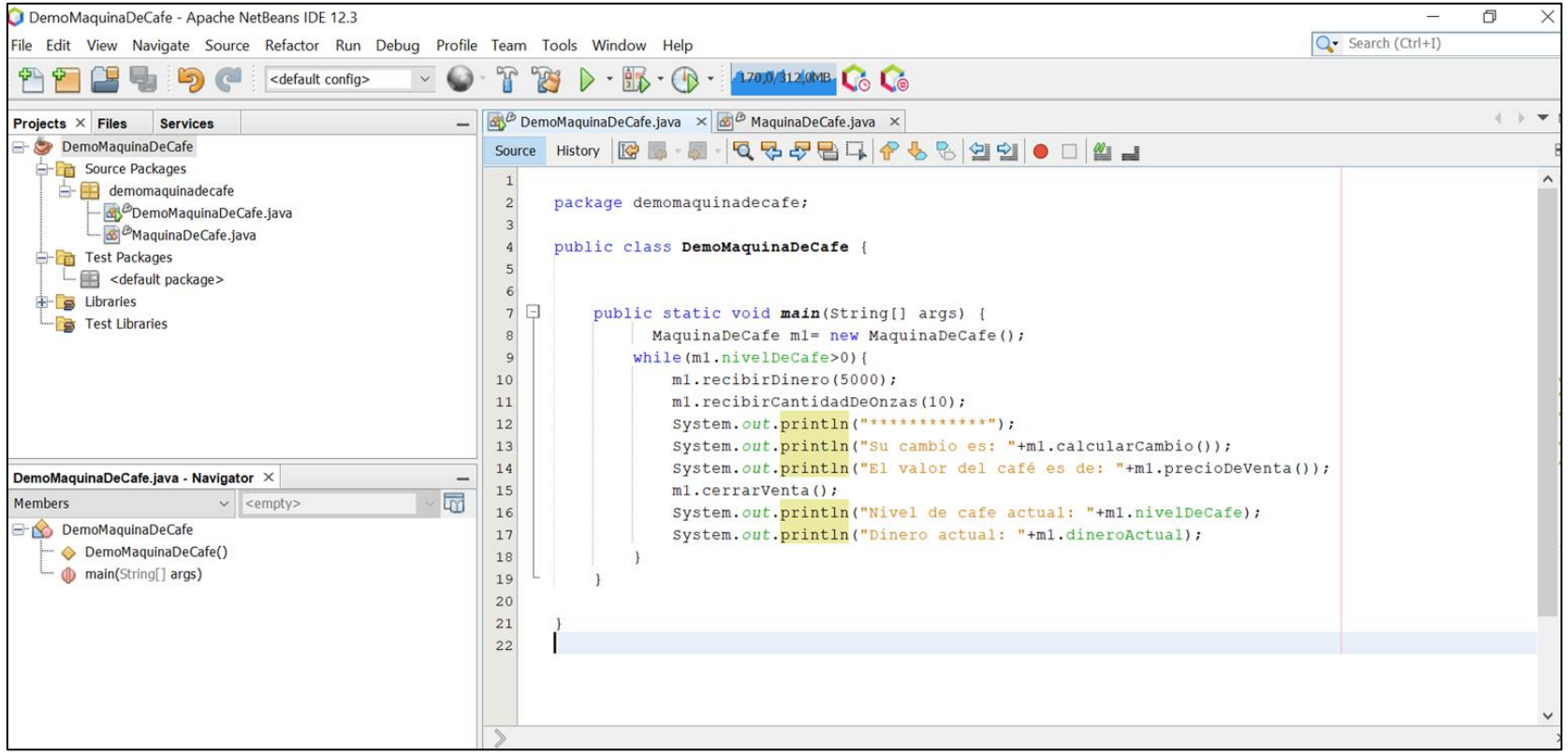


Diagrama de clases del ejemplo:

MaquinaDeCafe
+nivelDeCafe: int
+valorDeLaOnza: int
+dineroActual: int
+dineroRecibido: int
+cantidadDeOnzasAVender: int
+MaquinaDeCafe ( )
+calcularCambio( ): int
+precioDeVenta( ): int
+recibirDinero( ): void
+recibirCantidadDeOnzas( ): void
+cerrarVenta( ): void

Lo primero que debemos hacer es crear un proyecto y le vamos a colocar el nombre de DemoMaquinaDeCafe; de igual manera, nos va a crear el paquete, se copia este código en el programa principal, si observamos la imagen desde el public static void main , vamos a correr el programa, debemos instalar el objeto.



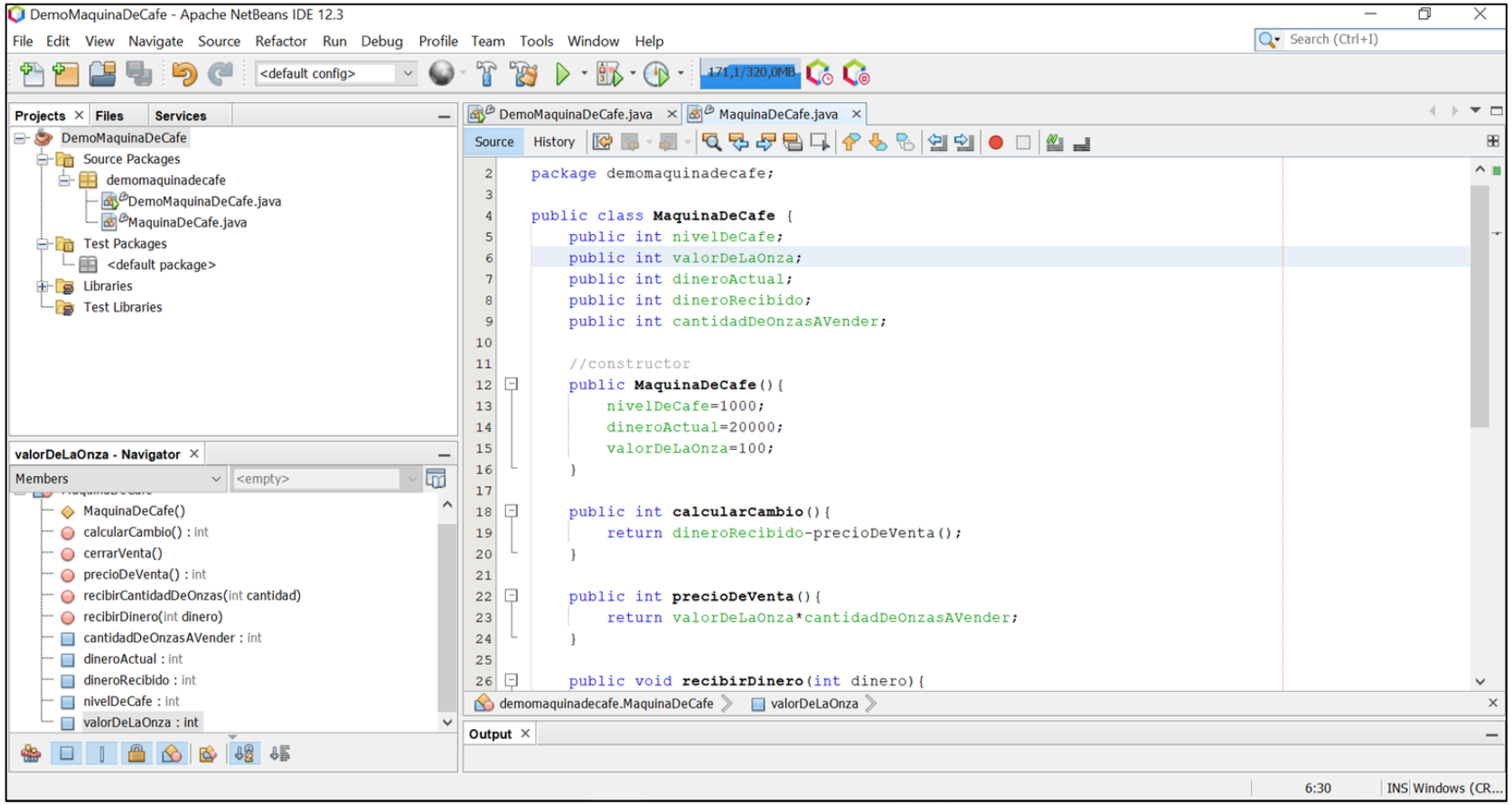
Vamos a crear una clase la cual se llama **MaquinaDeCafe** y vamos a colocar el siguiente código en la clase el cual tiene los métodos que vamos a utilizar:

```
public class MaquinaDeCafe {
public int nivelDeCafe;
public int valorDeLaOnza;
public int dineroActual;
public int dineroRecibido;
public int cantidadDeOnzasAVender;
//constructor
public MaquinaDeCafe(){
    nivelDeCafe=1000;
    dineroActual=20000;
    valorDeLaOnza=100;
}
```

```
 public int calcularCambio(){
        return dineroRecibido-precioDeVenta();
    }

    public int precioDeVenta(){
        return valorDeLaOnza*cantidadDeOnzasAVender;
    }
    public void recibirDinero(int dinero){
        dineroRecibido+=dinero;
    }
    public void recibirCantidadDeOnzas(int cantidad){
        cantidadDeOnzasAVender+=cantidad;
    }
    public void cerrarVenta(){
        nivelDeCafe-=cantidadDeOnzasAVender;
        dineroActual+=precioDeVenta();
        cantidadDeOnzasAVender=0;
        dineroRecibido=0;
    }
}
```

Les muestro el pantallazo de la clase:



Anotaciones

Java permite la inclusión de información adicional en sus archivos fuentes, a esta información es a la que se le conoce como anotaciones, y esta no altera el comportamiento de un sistema de software. Las anotaciones en Java tienen unas características generales, dentro de las cuales encontramos:

Empiezan con el carácter '@'.

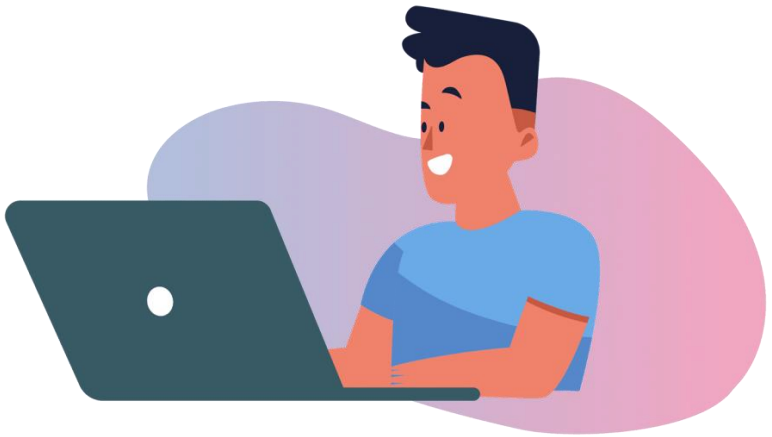
No cambian el comportamiento de un programa o sistema de software.

No son comentarios, ya que pueden cambiar la forma en la que el compilador trata al programa.

Relacionan los componentes del programa con datos.

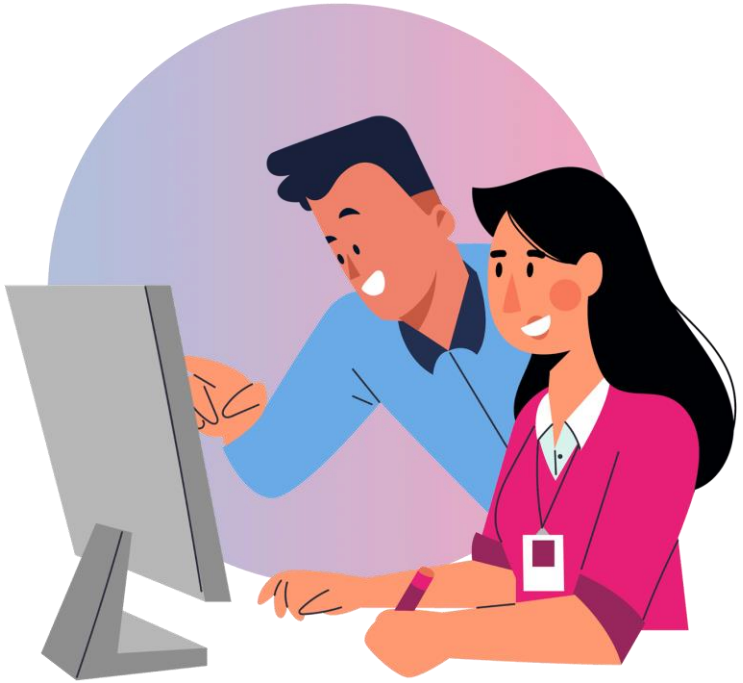
Para profundizar en el tema relacionado con las anotaciones en Java se recomienda el siguiente enlace:

https://www.tokioschool.com/noticias/anotaciones-en-java/



4. Los niveles de visibilidad (modificadores de acceso)

Las clases, los métodos y los atributos, pueden ser declarados de acuerdo a cierto nivel de visibilidad. Existen distintos niveles de visibilidad dentro del lenguaje de programación Java, que restringen el acceso dentro de un sistema de software tanto a las clases, como a los atributos, y a los métodos. Estos niveles de acceso pueden ser modificados mediante un modificador que acompaña a la declaración de cada uno de estos elementos. Cada modificador ofrece un nivel de visibilidad dentro del sistema de software que se esté desarrollando, en el ejemplo de la máquina de café el modificador de acceso que utilizamos fue el public.



5. Contextos estáticos y dinámicos

Las clases, los métodos o los atributos, pueden ser declarados estáticos o no estáticos. Si dentro de una clase algún atributo o algún método es declarado estático este podrá ser accedido sin necesidad de recurrir a una instancia de la clase u objeto. Como ejemplo tenemos la clase java.lang.Math, cuyas características pueden ser revisadas en https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html, en donde podemos encontrar muchos métodos declarados de manera estática mediante la palabra reservada static. Uno de estos métodos es max(float a, float b) cuyo uso puede ser el siguiente:

```
import java.lang.Math;
public class Test {
    public static void main(String[] args) {
        float a = 5.0f;
        float b = 8.0f;
        float maximum = Math.max(a, b);
        System.out.println(maximum);
    }
}
```

Por medio de ese método es calculado el máximo valor entre dos números tipo float. Los métodos o atributos declarados como no estáticos, no pueden ser referenciados desde un contexto estático, esto generaría un error al momento de compilar el programa. Una característica de declarar un método tipo estático es evitar su sobreescritura.

Para profundizar en la temática relacionada con contextos estático y dinámico, recomendamos el siguiente enlace:

https://guru99.es/java-static-variable-methods/

Recurso audiovisual

Video: Aprendiendo sobre el paradigma orientado a objetos

El futuro digital es de todos

MinTIC

Hechos QUE CONECTAN

CICLO 2

EJE TEMÁTICO 2

PROGRAMACIÓN ORIENTADA A OBJETOS Y UML

Universidad Industrial de Santander

UIS

MisiónTIC 2022