TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Tijuana

Nombre de la materia:

Programación Orientada a Objetos (POO)

Clave:

AED-1286SC2A

Horario:

11:00-12:00

Nombre del docente:

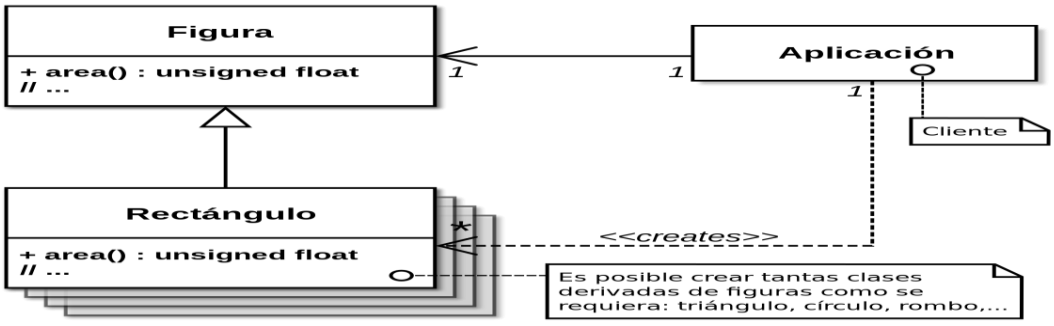
Dr. Mario García Valdez

Nombre del alumno:

Silvia Alejandra Salazar Félix

Número de control:

18212266



INDICE

UNIDAD 1

1.1 Introducción al paradigma de programación orientada a objetos

1.2. UML: Diagrama de clases

UNIDAD 1

1.1. Introducción al paradigma de la programación orientado a objetos.

Con tus palabras, pero investigando define los siguientes conceptos:

1. Paradigma.

A lo largo de la historia, el término paradigma fue objeto de muchas interpretaciones. En su origen griego, significaba modelo, ejemplo o patrón. Sobre este punto de partida, podemos hablar de un paradigma como un conjunto de creencias, prácticas y conocimientos que guían el desarrollo de una disciplina durante un período de tiempo. En diversas ramas de la ciencia, un conjunto de ideas en vigencia puede ser reemplazado drásticamente por otro que entre en conflicto con él y se demuestre más acertado. La programación tiene sus propios paradigmas, pero el término paradigma de programación no necesariamente representa un modelo único que deba ser respetado hasta que aparezca otro mejor. De hecho, actualmente muchos paradigmas coexisten en armonía.

Un paradigma de programación es un estilo de desarrollo de programas. Es decir, un modelo para resolver problemas computacionales. Los lenguajes de programación, necesariamente, se encuadran en uno o varios paradigmas a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis.

¿Cuáles son los principales paradigmas de programación?

- **Imperativo.** Los programas se componen de un conjunto de sentencias que cambian su estado. Son secuencias de comandos que ordenan acciones a la computadora.
- **Declarativo.** Opuesto al imperativo. Los programas describen los resultados esperados sin listar explícitamente los pasos a llevar a cabo para alcanzarlos.
- **Lógico.** El problema se modela con enunciados de lógica de primer orden.
- Funcional. Los programas se componen de funciones, es decir, implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida.
- **Orientado a objetos**. El comportamiento del programa es llevado a cabo por objetos, entidades que representan elementos del problema a resolver y tienen atributos y comportamiento.

Descripción con mis palabras:

Los paradigmas de programación son formas, modos o estilos de programación que son distintos entre sí pero que pueden coexistir en o durante un determinado tiempo. Entonces tomando en consideración en el paradigma orientado a objetos los programas se concentran en los objetos que se van a manipular, y no en la lógica para poder controlarlos. Estos pueden ser: Alumnos, personas, contactos, agendas, personajes de juegos, etc., en cada uno de ellos se tendrán métodos y atributos y otros ciertos valores que los identifica, además de la comunicación entre cada uno de ellos.

2. Abstracción, da dos ejemplos.

En programación:

La abstracción consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan. En programación, el término se refiere al énfasis en el "¿qué hace?" más que en el "¿cómo lo hace?". El común denominador en la evolución de los lenguajes de programación, desde los clásicos o imperativos hasta los orientados a objetos, ha sido el nivel de abstracción del que cada uno de ellos hace uso.
Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos, y, cuando lo están, una variedad de técnicas es requeridas para ampliar una abstracción.

Descripción con mis propias palabras:
Las características específicas de un objeto, aquellas que lo distinguen de los demás tipos de objetos y que logran definir límites conceptuales respecto a quien está haciendo dicha abstracción del objeto.
El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real.

Ejemplos:

1. Diríamos que una bicicleta es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca.

```
        //Atributos
        public string marca;
        public string modelo;
        public string frenos;
        public string peso;
        public int velocidades;
        public string color;

        //Métodos
        public void imprimir_marca()
        {
            Console.WriteLine(marca);
        }
        public void imprimir_modelo()
        {
            Console.WriteLine(modelo);
        }
        public void imprimir_frenos()
        {
            Console.WriteLine(frenos);
        }
        public void imprimir_peso()
        {
            Console.WriteLine(peso);
        }
        public void imprimir_velocidades()
        {
            Console.WriteLine(velocidades);
        }
        public void imprimir_color()
        {
            Console.WriteLine(color);
        }
```

2. Imaginemos que queremos aplicar la abstracción a un alumno. El objeto sería el alumno y sus características serian: nombre, edad, y correo electrónico.

```

namespace Ejercicio_materia
{
    class Alumno
    {
        //Atributos
        public string nombre;
        public int edad;
        public string correo;

        //Método
        public void imprime()
        {
            Console.WriteLine("{0},{1},{2}", nombre,edad,correo);
        }
    }
```

En artes:

El arte abstracto es el contrario del figurativo (es decir, la representación de objetos identificables mediante imágenes reconocibles). Por tanto, la abstracción no representa *«cosas»* concretas de la naturaleza, sino que propone una nueva realidad. Propone un arte puro mirando más allá de nuestra realidad.

Por un lado, al arte **abstracto expresivo**: subjetivo y espontáneo, improvisado a veces, donde el protagonismo es de la expresividad del artista, que prescinde de estructura y se vuelca en el gesto, el material y el sentimiento que provoca la obra. Es por tanto altamente ambiguo e interpretable.

Por otro está el arte **abstracto geométrico**: que pretende ser objetivo y universal, planificado, en la que la composición es estructurada y que evita toda expresividad mediante el uso de la geometría. Suele defender una factura impersonal y pretende evocar claridad y precisión.

En psicología:

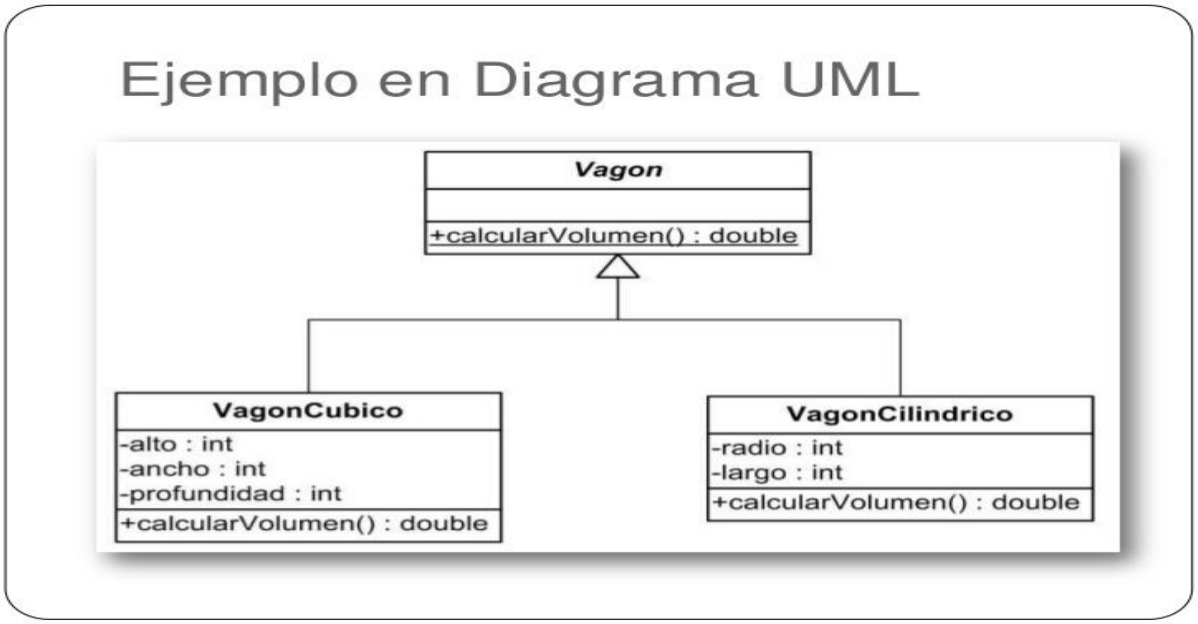
Para la psicología, la abstracción se trata de un proceso mediante el cual los componentes que resultan fundamentales en la información de un suceso o fenómeno, se reducen para que mediante la conservación de los rasgos que son más relevantes poder formar conceptos o categorías.

3. ¿Por qué decimos que la Clase es el mecanismo de abstracción de los lenguajes OO?

Una clase equivale a la generalización o abstracción de un tipo específico de objetos. Las clases actúan como intermediarias entre una abstracción y los clientes que pretenden utilizar la abstracción. De esta forma, la clase muestra:

2. **Visión externa** de comportamiento (interface), que enfatiza la abstracción escondiendo su estructura y secretos de comportamiento.
3. **Visión interna** (implementación), que abarca el código que se ofrece en la interface de la clase.

En otras palabras, las clases representan una abstracción, las cuales son las características que comparten cada uno de los objetos.



4. Explica el concepto de encapsulamiento, busca dos imágenes que te ayuden a describir el concepto, una que tenga algún sistema sin encapsulamiento y otra donde si lo tenga. Menciona porque es importante y que problemas puede evitar.

En **programación modular**, y más específicamente en **programación orientada a objetos**, se denomina encapsulamiento al ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que solo se pueda cambiar mediante las operaciones definidas para ese objeto.

Cada **objeto** está aislado del exterior, es un módulo natural, y la aplicación entera se reduce a un agregado o rompecabezas de objetos. El aislamiento protege a los datos asociados de un objeto contra su modificación por quien no tenga derecho a acceder a ellos, eliminando efectos secundarios e interacciones.

De esta forma el usuario de la clase puede obviar la implementación de los métodos y propiedades para concentrarse solo en cómo usarlos. Por otro lado, se evita que el usuario pueda cambiar su estado de maneras imprevistas e incontroladas.

Algunos autores dicen que Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la **cohesión** de los componentes del sistema.

El encapsulamiento es importante porque es utilizado para esconder detalles no importantes de otros objetos. Entonces, los detalles pueden cambiar en cualquier tiempo sin afectar otras partes del programa.

El encapsulamiento consiste en unir en la Clase las características y comportamientos, esto es, las variables y métodos. Es tener todo esto es una sola entidad. En los lenguajes estructurados esto era imposible. Es evidente que el encapsulamiento se logra gracias a la abstracción y el ocultamiento.

La utilidad del encapsulamiento va por la facilidad para manejar la complejidad, ya que tendremos a las Clases como cajas negras donde sólo se conoce el comportamiento, pero no los detalles internos, y esto es conveniente porque nos interesará será conocer qué hace la Clase, pero no será necesario saber cómo lo hace.

En la siguiente imagen se muestra un ejemplo sin encapsulamiento:

```
class Persona {
    String nombre;
    int edad;
    int edad() {
        return this.edad;
    }
}
```

En la siguiente imagen se muestra un ejemplo de un programa, pero con encapsulamiento:

```
class Persona {
    private String nombre;// <-- ya nadie lo puede ver
    private int age;// <-- ya nadie lo puede ver
    public int age() {
        return this.age;
    }
}
```

```java
public class UniversityStudent {
    private int id;
    private String name;
    private String gender;
    private String university;
    private String career;
    private int numSubjects;
    public UniversityStudent(int id, String name, String gender,
        String university, String  career, int numSubjects) {
        this.id = id;
        this.name = name;
        this.gender = gender;
        this.university = university;
        this.career = career;
        this.numSubjects = numSubjects;
    }

    public void inscribeSubjects() {
      // TODO: implement
    }
    public void cancelSubjects() {
       // TODO: implement
    }
    public void consultRatings() {
       // TODO: implement
    }
```

```java
    public void setNumSubjects(int numSubjects){
        if( numSubjects < 0 || numSubjects > 10 )
            System.out.println("Numero invalido de materias");
        else
            this.numSubjects = numSubjects;
    }
    public int getNumSubjects(){
        return numSubjects;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public String getGender() {
        return gender;
    }
    public String getUniversity() {
        return university;
    }
    public String getCareer() {
        return career;
    }
}
```

Encapsulamiento

```java
public class NumerosPrimos implements InterfacePrimos {
   private int i;
   private int j;
   private int tmp;
   private int cont;
   private String stringPrimos;

   public String numerosPrimos( int n ) {
      tmp = cont = 0;
      stringPrimos = "";
      for(i = 2; i <= n; i++){
         for(j = 2; j < i; j++){
            tmp = i % j;
            if( tmp == 0 ){
               tmp = -1;
               break;
            }
         }
         if( tmp != -1 )
            stringPrimos += i + "\n";
      }
      return stringPrimos;
   }
}
```

Encapsulamiento

```java
public interface InterfacePrimos{
     public String numerosPrimos( int n );
}
```

Interface

```java
public class PrincipalPrimos{
   public static void main( String args[] ){
      InterfacePrimos ip = new NumerosPrimos( );
      System.out.println( ip.numerosPrimos( Integer.parseInt( args[0] ) ) );
   }
}
```

Clase principal: No se preocupa por como esta implementado el Objeto; solo necesita comprender su interfaz.

5. Describe con tus palabras el concepto de herencia e ilustra el concepto con imágenes.

La herencia es específica de la programación orientada a objetos, donde una clase nueva se crea a partir de una clase existente. La herencia (a la que habitualmente se denomina subclase) proviene del hecho de que la subclase (la nueva clase creada) contiene las atributos y métodos de la clase primaria. La principal ventaja de la herencia es la capacidad para definir atributos y métodos nuevos para la subclase, que luego se aplican a los atributos y métodos heredados.

Esta particularidad permite crear una estructura jerárquica de clases cada vez más especializada. La gran ventaja es que uno ya no debe comenzar desde cero cuando desea especializar una clase existente. Como resultado, se pueden adquirir bibliotecas de clases que ofrecen una base que puede especializarse a voluntad (la compañía que vende estas clases tiende a proteger los datos miembros usando la encapsulación).

Descripción con mis propias palabras:

Entonces se puede decir que la herencia consiste en como su nombre lo dice “heredar” de una clase a otra sus características, métodos, etc., es decir tenemos una clase primaria y otra secundaria, se tomara el código de la clase primaria y será reutilizado en la secundaria, también hay muchos lugares en donde lo explican como padre e hija, siendo la clase principal el padre el que la va a heredar a sus hijas (clases secundarias).

1.2. UML: diagrama de clases.

1. Investiga la historia y haz un resumen del Lenguaje de Modelado Unificado, donde se mencione: quienes son sus principales autores (Booch, Rumbaugh, Jacobson), en qué tipo de sistemas se utiliza, en particular el Diagrama de Clases. Menciona algunas de las herramientas para el modelado en UML.

Historia

Aproximación de **Ivar Jacobson** (OOSE: Object- Oriented Software Engineering) mediante la metodología de casos de uso (use case).

El desarrollo de UML comenzó a finales de 1994 cuando **Grady Booch** y **Jim Rumbaugh** de Rational Software Corporation empezaron a unificar sus métodos. A finales de 1995, **Ivar Jacob** son y su compañía Objectory se incorporaron a Rational en su unificación, aportando el método OOSE.

De las tres metodologías de partida, las de Bco. y Rumbaugh pueden ser descritas como centradas en objetos, ya que sus aproximaciones se enfocan hacia el modelado de los objetos que componen el sistema, su relación y colaboración.

Por otro lado, la metodología de Jacobson es más centrada a usuario, ya que todo en su método se deriva de los escenarios de uso. UML se ha ido fomentando y aceptando como estándar desde el OMG, que es también el origen de CORBA, el estándar líder en la industria para la programación de objetos distribuidos.

En 1997 UML 1.1 fue aprobada por la OMG convirtiéndose en la notación estándar de facto para el análisis y el diseño orientado a objetos.

UML es el primer método en publicar un meta-modelo en su propia notación, incluyendo la notación para la mayoría de la información de requisitos, análisis y diseño. Se trata pues de un meta-modelo auto-referencial (cualquier lenguaje de modelado de propósito general debería ser capaz de modelarse a sí mismo).

¿Qué es?

El lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el Object Management Group (OMG).

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

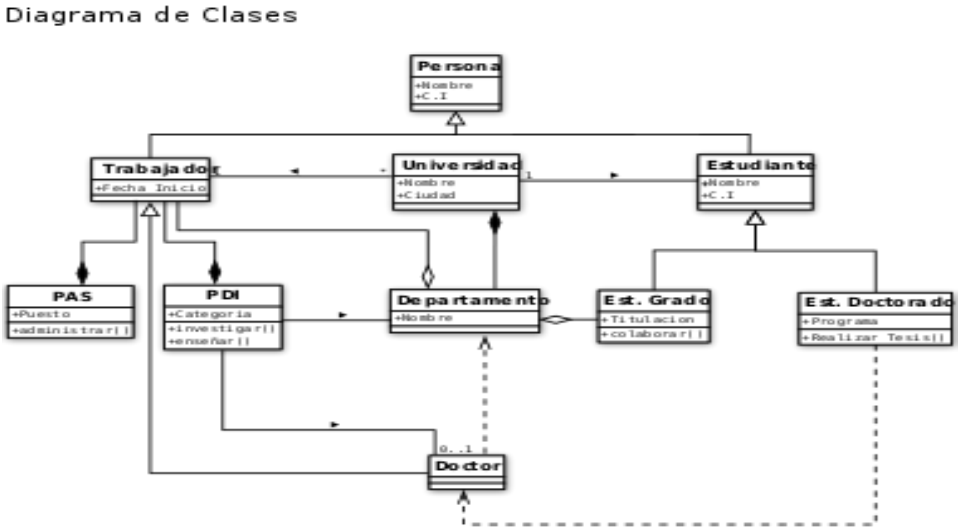
¿Qué tipo de sistemas utiliza?

Tipos de diagramas UML

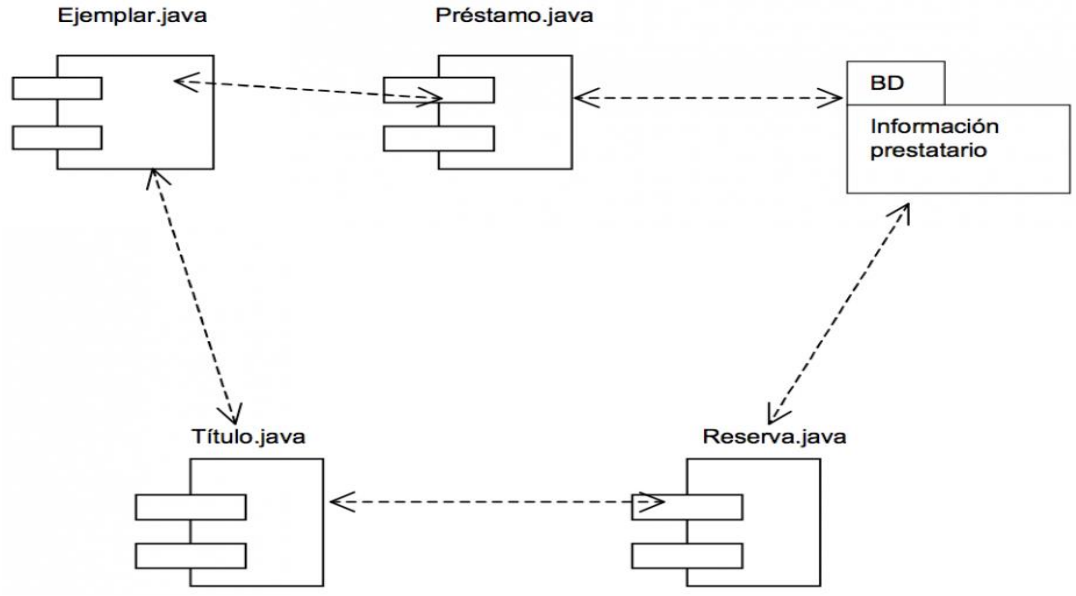
UML usa elementos y los asocia de diferentes formas para formar diagramas que representan aspectos estáticos o estructurales de un sistema, y diagramas de comportamiento, que captan los aspectos dinámicos de un sistema.

Diagramas UML estructurales

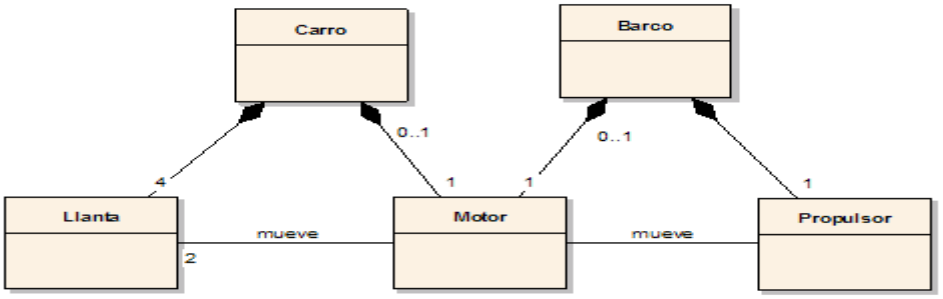
- **Diagrama de clases** El diagrama UML más comúnmente usado, y la base principal de toda solución orientada a objetos. Las clases dentro de un sistema, atributos y operaciones, y la relación entre cada clase. Las clases se agrupan para crear diagramas de clases al crear diagramas de sistemas grandes.



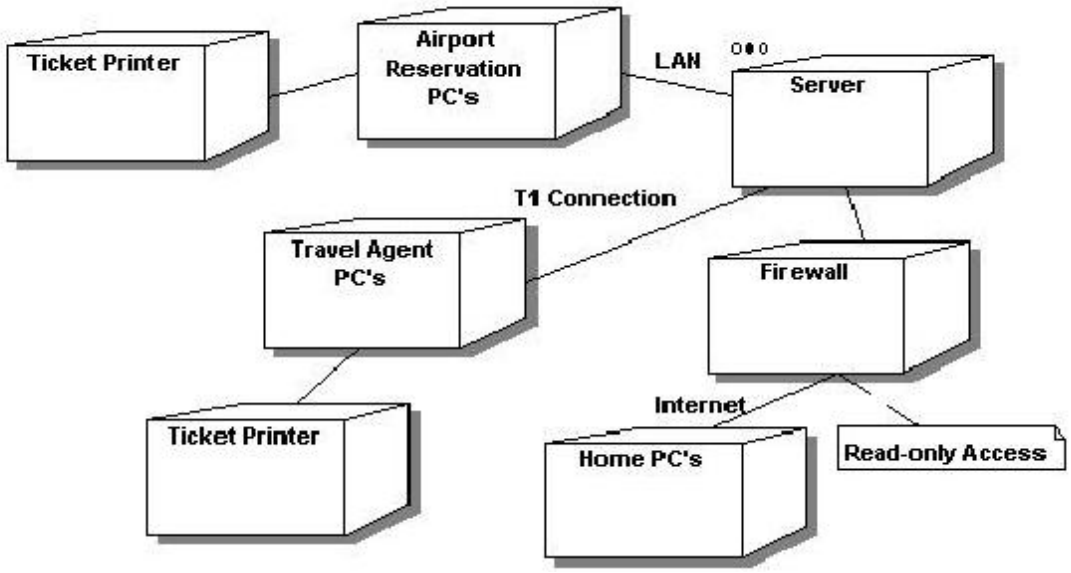
- **Diagrama de componentes** Muestra la relación estructural de los elementos del sistema de software, muy frecuentemente empleados al trabajar con sistemas complejos con componentes múltiples. Los componentes se comunican por medio de interfaces.



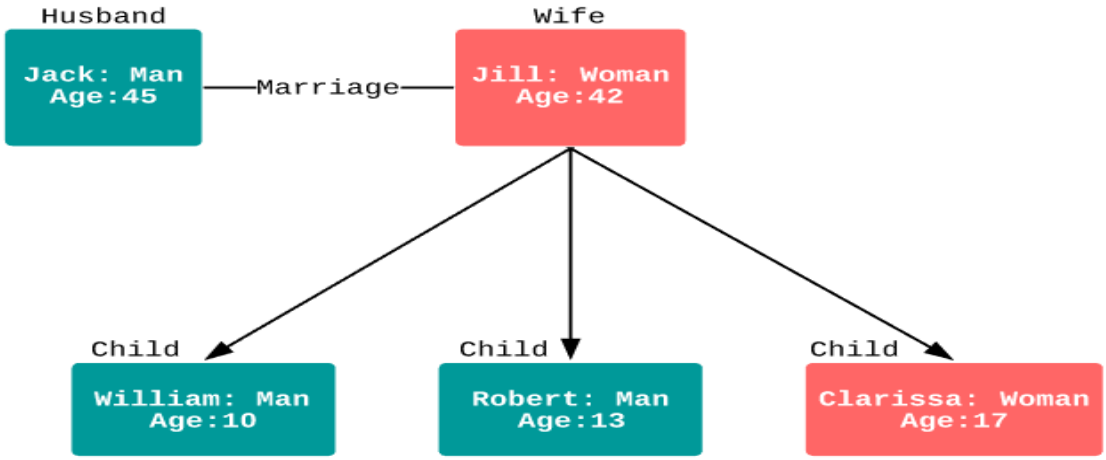
- **Diagrama de estructura compuesta** Los diagramas de estructura compuesta se usan para mostrar la estructura interna de una clase.



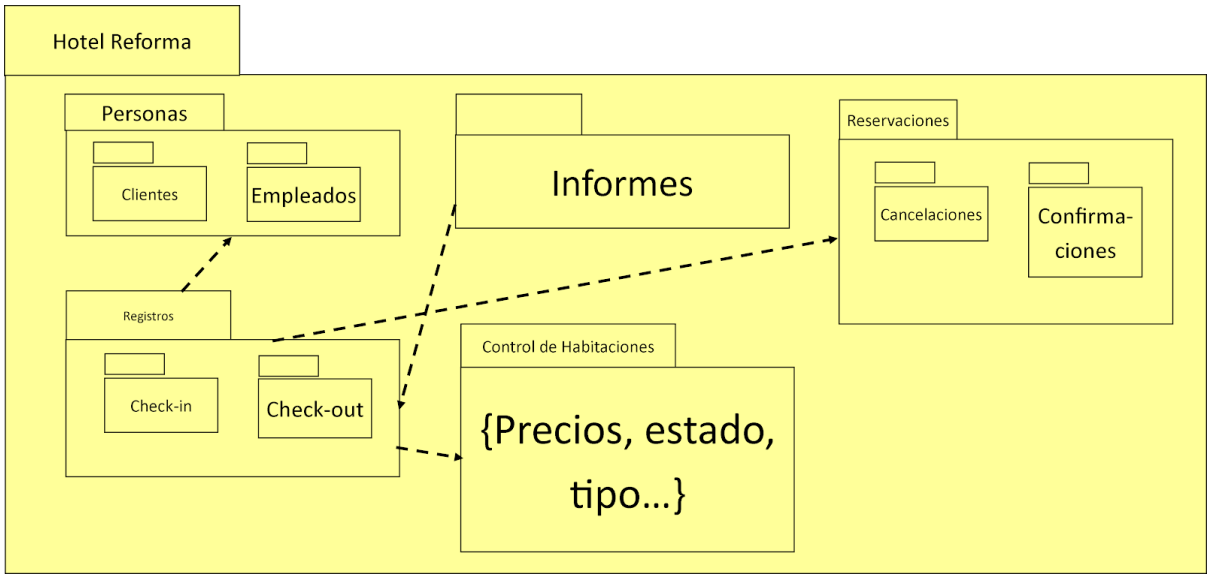
- **Diagrama de implementación** Ilustra el hardware del sistema y su software. Útil cuando se implementa una solución de software en múltiples máquinas con configuraciones únicas.



- **Diagrama de objetos** Un diagrama de objetos muestra un conjunto de objetos y sus relaciones en un momento concreto.

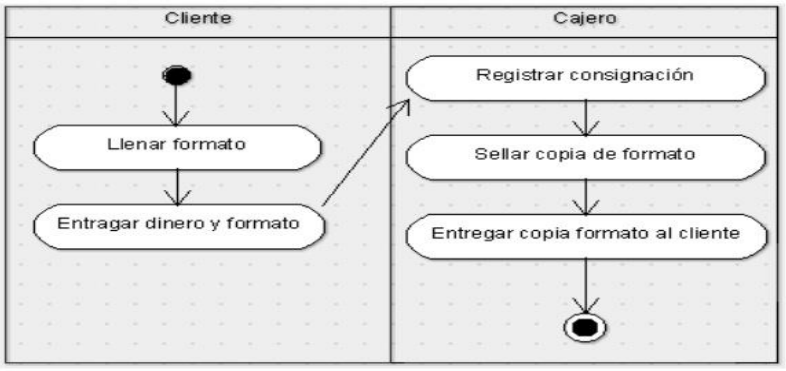


- **Diagrama de paquetes** Hay dos tipos especiales de dependencias que se definen entre paquetes: la importación de paquetes y la fusión de paquetes. Los paquetes pueden representar los diferentes niveles de un sistema para revelar la arquitectura. Se pueden marcar las dependencias de paquetes para mostrar el mecanismo de comunicación entre niveles.

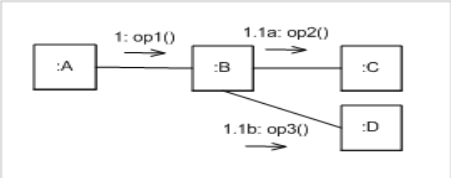


Diagramas UML de comportamiento

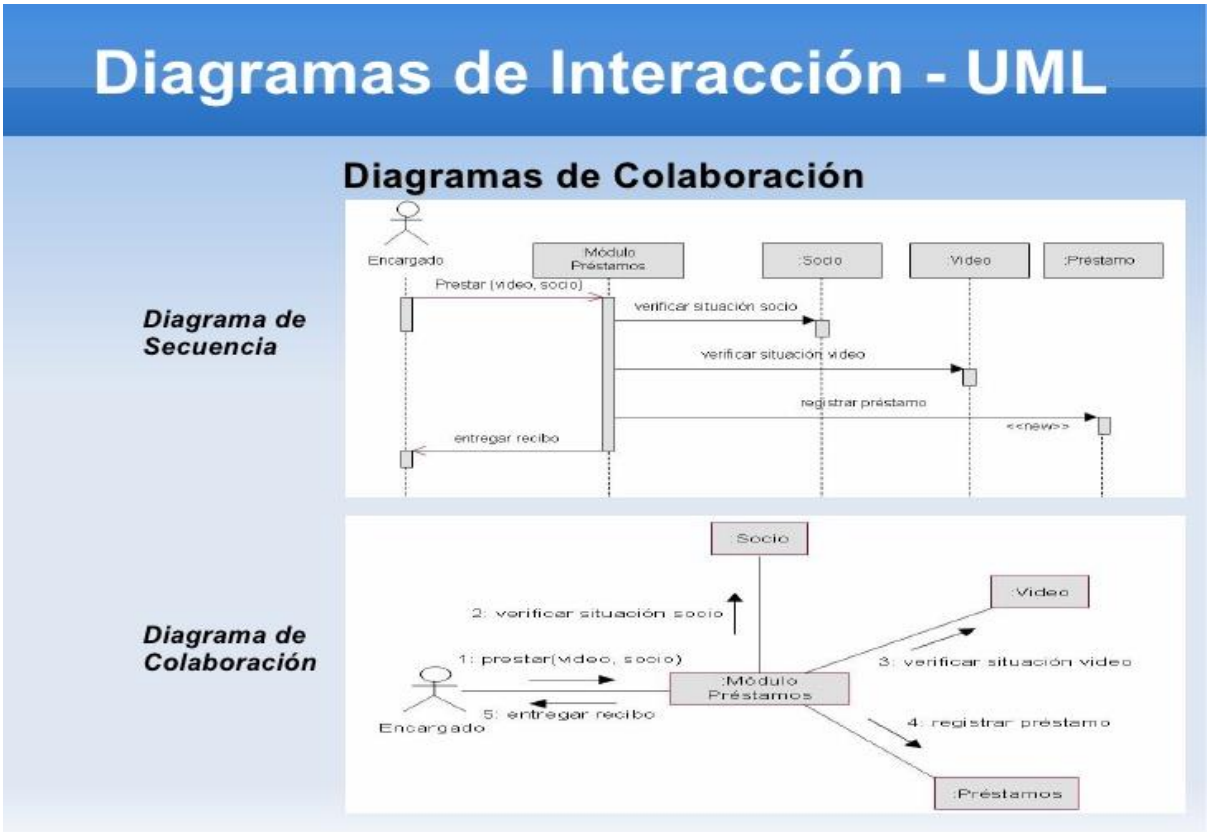
- **Diagramas de actividades** Flujos de trabajo de negocios u operativos representados gráficamente para mostrar la actividad de alguna parte o componente del sistema. Los diagramas de actividades se usan como una alternativa a los diagramas de máquina de estados.



- **Diagrama de comunicación** Similar a los diagramas de secuencia, pero el enfoque está en los mensajes que se pasan entre objetos. La misma información se puede representar usando un diagrama de secuencia y objetos diferentes.

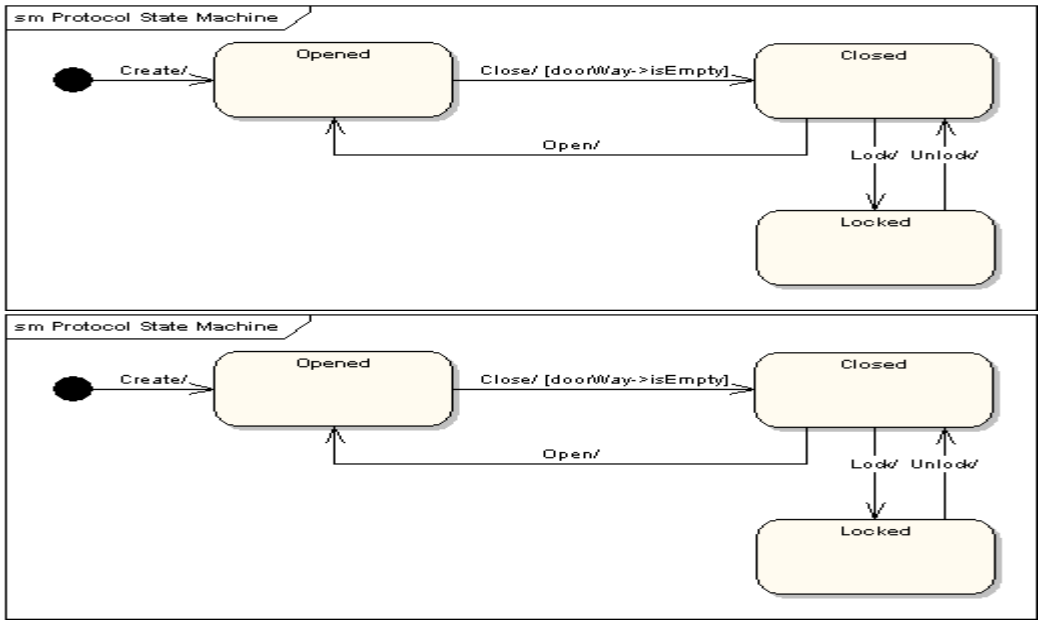


- **Diagrama de panorama de interacciones** Hay siete tipos de diagramas de interacciones. Este diagrama muestra la secuencia en la cual actúan.

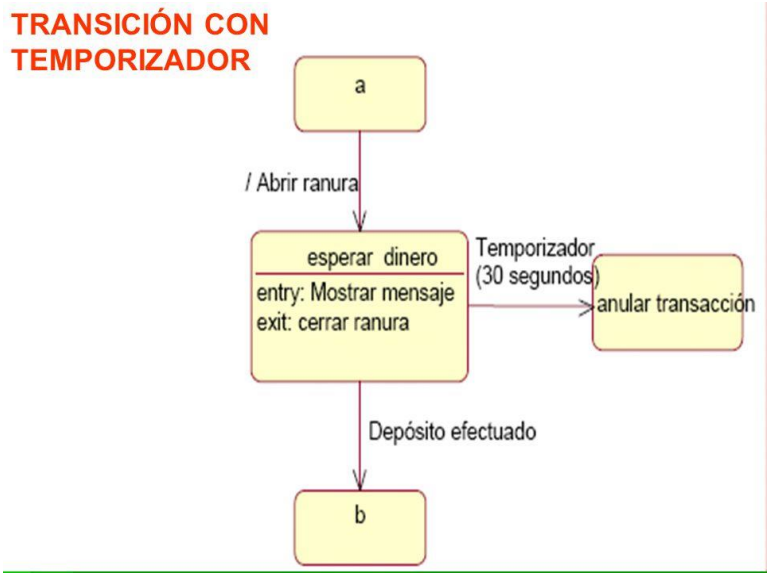


- **Diagrama de secuencia** Muestra cómo los objetos interactúan entre sí y el orden de la ocurrencia. Representan interacciones para un escenario concreto.

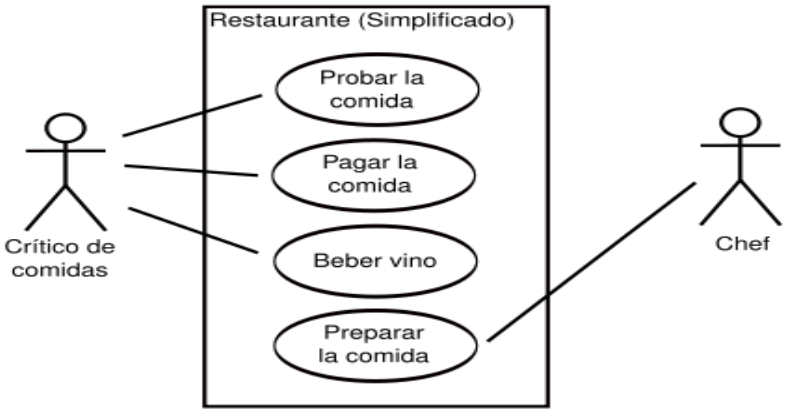
- **Diagrama de máquina de estados** Similar a los diagramas de actividades, describen el comportamiento de objetos que se comportan de diversas formas en su estado actual**.**



- **Diagrama de temporización** Al igual que en los diagramas de secuencia, se representa el comportamiento de los objetos en un período de tiempo dado. Si hay un solo objeto, el diagrama es simple. Si hay más de un objeto, las interacciones de los objetos se muestran durante ese período de tiempo particular.



- **Diagrama de caso de uso** Representa una funcionalidad particular de un sistema. Se crea para ilustrar cómo se relacionan las funcionalidades con sus controladores (actores) internos/externos.

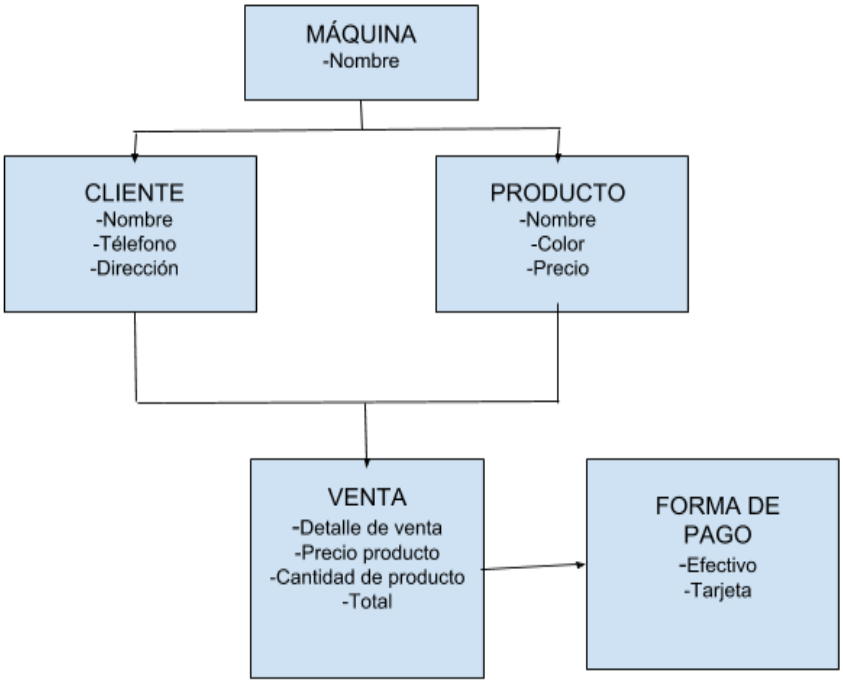


¿Sabes de alguna empresa local que utilice este lenguaje?

Una empresa que utiliza este lengua es **GH-BUS**,es una empresa de transporte para pasajeros, de rutas interprovinciales (Lima - Tarapoto, Lima - Chachapoyas, Chiclayo - Tarapoto). Con modernas Unidades (Buses semi-cama, bus cama y buses panorámicos), que cuentan con servicios como (Música, Sistema de Video Televisores LCD, Aire acondicionado, servicios higiénicos, baño con extractor, inodoro de acero quirúrgico, alimentación a bordo (desayuno, almuerzo y cena).

2. Escribe una propuesta de una máquina que venda distintos artículos y haz el diagrama de clases del sistema que propones.

Recuerda que puede haber composición (un teclado se compone de botones) y generalización (tipo de productos, tipo de pago).



REFERENCIAS

Autores: Julián Pérez Porto y María Merino. Publicado: 2008. Actualizado: 2012. Definiciones: Definición de paradigma (https://definicion.de/paradigma/)

Autores: Julián Pérez Porto y Ana Gardey. Publicado: 2008. Actualizado: 2012. Definición.de: Definición de abstracción (https://definicion.de/abstraccion/)

Wikipedia, 2019, de Wikipeadia, https://es.wikipedia.org/wiki/Abstracci%C3%B3n

Título: Abstracción. Sitio: Definición ABC. Fecha: 30/06/2014. Autor: Florencia Ucha. URL: https://www.definicionabc.com/general/abstraccion.php

Esteban Martini, s.f, "Objetos, clases y herencia simple", sitio web: https://emartini.wordpress.com/2008/08/06/orientacion-a-objetos-objetos-clases-y-herencia-simple/

Carlos Villagómez, 2017, "Encapsulamiento de datos", sitio web: https://es.ccm.net/contents/410-poo-encapsulacion-de-datos