UAEM | Universidad Autónoma del Estado de México

Centro Universitario UAEM Zumpango
Ingeniería en Computación

CENTRO UNIVERSITARIO UAEM ZUMPANGO

UA: ANÁLISIS DE LOS LENGUAJES DE PROGRAMACIÓN

UNIDAD II

ANÁLISIS DE LOS PARADIGMAS DE PROGRAMACIÓN

Temas: paradigma imperativo y paradigma orientado a objetos

Objetivos:

- Identificar los conceptos que dan soporte al paradigma imperativo.
- Conocer las principales características del paradigma imperativo.
- Comprender los conceptos de clase, objeto, herencia, polimorfismo y encapsulamiento.
- Aplicar el paradigma orientado a objetos en la solución de problemas.

PARADIGMA IMPERATIVO (1/2)

- El **primer paradigma** formalmente aceptado y el **más representativo** de programación.
- Apareció en los **años 50** con los primeros lenguajes de programación.
- En un principio fueron los **lenguajes de máquina** de las primeras computadoras.
- Esta basado en **comandos u órdenes** (enunciados imperativos) que actualizan variables almacenadas en memoria.
- Esta basado en el **modelo de Von Neumann**.
- A partir de **Fortran**, la programación imperativa permite la implementación de programas complejos.

PARADIGMA IMPERATIVO (2/2)

La programación imperativa describe a la programación en términos del **estado del programa** y **sentencias** que modifican dichos estados.

Los programas imperativos resultan en un conjunto de instrucciones que indican a la computadora la forma en que realizará su tarea.

MODULARIDAD (1/6)

La programación imperativa se rige por dos conceptos básicos para la construcción de programas: el **módulo** y la **estructura**.

Los dos tipos de programación imperativa señalan que un programa debe dividirse en **subprogramas**, segmentos a módulos para hacerlos más legible y manejable.

Los módulos debe tener **bien definida su tarea** y si necesitan de otros comunicarse por medio de una **interfaz de comunicación**.

MODULARIDAD (2/6)

Cada módulo es un procedimiento o función o conjunto de éstas, incluso librería con determinada funcionalidad que pueden ser llamadas desde un programa.

MODULARIDAD (3/6)

Actividad 1

Desarrollar un sistema de información para controlar la recepción de vehículos en un taller mecánico.

La recepción del vehículo se lleva a cabo directamente con el cliente, mismo que da sus observaciones sobre los problemas o detalles que el vehículo presenta. El mecánico efectúa un diagnóstico inicial y lo deja asentado en la orden. Si el cliente acepta el diagnóstico, el jefe de mecánicos define las tareas y refacciones a utilizar en el servicio.

Finalmente, el cliente es informado del presupuesto y decide si autoriza o rechaza el servicio.

MODULARIDAD (4/6)

Actividad 1

Definir los módulos conceptuales que propondrías para dividir el sistema a desarrollar.

Cada módulo debe incluir:

- Nombre
- Objetivo
- Parámetros de entrada
- Procesamiento de datos
- Valores de salida.

MODULARIDAD (5/6)

Actividad 2

Desarrollar una agenda electrónica para el control de citas de un consultorio dental. Se requiere que se pueda dar de alta en ella los datos de los pacientes, las fechas de sus consultas, alergias a medicamentos y generar para cada persona un plan de pagos para cada cita. Es necesario que la agenda pueda generar la lista de pacientes de un día dado, informar sobre los medicamentos que pueden ser peligrosos para el paciente y el estado de pagos para cada uno de ellos.

MODULARIDAD (6/6)

Actividad 2

Definir los módulos conceptuales que propondrías para dividir el sistema a desarrollar.

Cada módulo debe incluir:

- Nombre
- Objetivo
- Parámetros de entrada
- Procesamiento de datos
- Valores de salida.

CELDA DE MEMORIA VARIABLE (1/2)

La programación imperativa se basa en tres conceptos importantes: **celda de memoria variable**, **operaciones de asignación** y **operaciones de repetición**.

La memoria juega un papel primordial para que un programa pueda ser ejecutado.

En ella se **almacenan las instrucciones y datos** con los que trabajará.

Se identifican con una dirección única y son referenciadas con una variable.

CELDA DE MEMORIA VARIABLE (2/2)

- Dirección de memoria
- Apuntador
- Lenguajes tipificados
- Lenguajes no tipificados

OPERACIONES DE ASIGNACIÓN (1/1)

Cada valor calculado en un programa debe ser asignado a una localidad de memoria.

Tipos de asignación

- **Aritmética**: se asigna un valor numérico derivado de una operación aritmética. La variable se declara como tipo *integer*, *long* o *double*.
- **Lógica**: se asigna un valor de cierto o falso a una variable. La variable se declara como tipo *boolean*.
- **Cadena de caracteres**: se asigna como valor una cadena de caracteres a una variable. La variable se declara como tipo *string* o *char*.

OPERACIONES DE REPETICIÓN (1/2)

Un programa imperativo, normalmente realiza su tarea ejecutando una **secuencia de pasos elementales**, en este modelo la única forma de ejecutar algo complejo es repitiendo una secuencia de instrucciones.

El flujo del programa se controla a través de **estructuras**, que pueden ser secuenciales, repetitivas, selectivas y de salto. Por ejemplo:

- for
- while
- repeat

OPERACIONES DE REPETICIÓN (2/2)

Un vez que se ha determinado qué datos son los que un programa va a necesitar, se les asocia una **dirección de memoria** y se efectúa una **secuencia de transformación** en los datos hasta obtener el valor esperado.

Los datos se transforman desde un estado inicial hasta un estado final usando **reglas de transformación**.

Se crea un **grafo de estados intermedios** con las reglas y se analiza cuál es el estado intermedio más próximo la objetivo final.

CAMPOS DE APLICACIÓN (1/1)

Los lenguajes imperativos pueden resolver prácticamente **cualquier problema en cualquier área**.

Las aplicaciones pueden ser: control de nóminas, inteligencia artificial, control aéreo, cajeros automáticos, control de medicamentos, dispositivos móviles, aplicaciones en línea, tiempo real, entre otras.

Fue el primer paradigma y, por lo tanto, **puso orden a la manera de programar** por lo que su filosofía marcó la línea a seguir para resolver problemas cotidianos.

PARADIGMA ORIENTADO A OBJETOS (1/2)

- La programación orientada a objeto surge de la necesidad de contar con lenguajes que dieran **solución a problemas** de una forma más **parecida a la que utilizamos en la vida real**.

- Este paradigma **define objetos y clases como base** para la programación.

- Expresa un programa como un conjunto de **objetos que colaboran entre si** para realizar una tarea.

- Permite que los programas sean más **fáciles de escribir**, **mantener** y **reutilizar**.

PARADIGMA ORIENTADO A OBJETOS (2/2)

- Algunos de los lenguajes que soportan la orientación a objetos son:

✓Ada
✓C++
✓C#
✓Delphi
✓Eiffel
✓Java
✓Objective-C
✓Oz
✓PHP
✓PowerBuilder
✓Python
✓Ruby
✓Smalltalk
✓VB.NET

CLASES Y OBJETOS (1/2)

- Una **clase** es una patrón que **encapsula los datos** y las abstracciones de datos necesarios para **describir el contenido y comportamiento de alguna entidad del mundo real**.
- Es una descripción generalizada que describe una **colección de objetos similares**.

- Un **objeto** es una **instancia de una clase** específica.
- Estos heredan las **atributos** y **operaciones** de una clase (padre).
- Para modelar una clase y un objeto se utiliza el **lenguaje unificado de modelado** (UML).

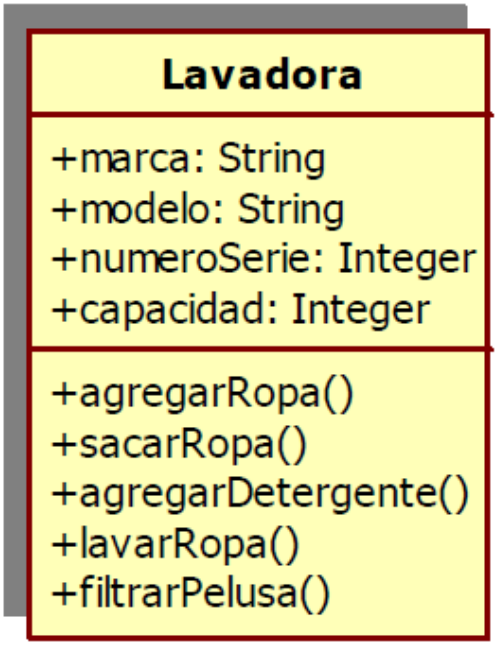
CLASES Y OBJETOS (2/2)

Identificar de la siguiente lista los elementos que pertenecen a una clase y los que representan un objeto:

- ✓ El nuevo modelo de PC.
- ✓ El carro de mi papá.
- ✓ Las coníferas de México.
- ✓ El oso panda del bosque de Chapultepec.
- ✓ Los alumnos de ICO
- ✓ Los clientes de la farmacia del ahorro
- ✓ La cuenta de la mesa 3.
- ✓ Las cuentas del restaurant las espadas brasileñas.
- ✓ El oso polar
- ✓ Auto de carrera
- ✓ El estacionamiento del CU UAEM Zumpango.
- ✓ El alumno con número de cuenta 93231.
- ✓ Cadena de tiendas walmart.

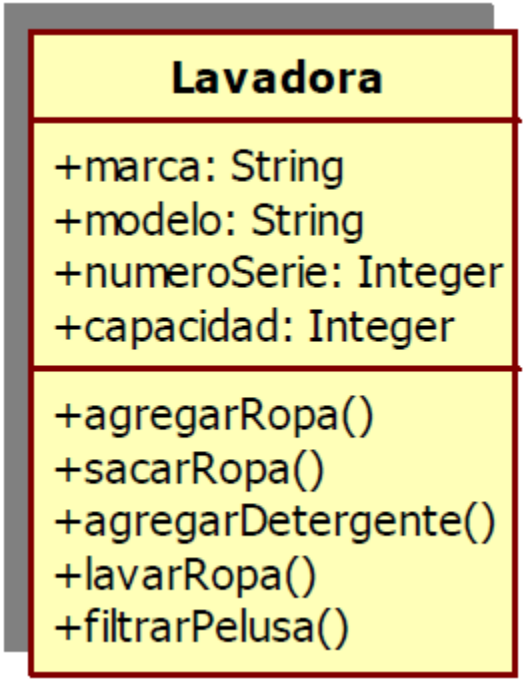
ATRIBUTOS Y MÉTODOS (1/3)

- Un **atributo** es una propiedad, rasgo o característica de una clase, que describe un rango de valores que la propiedad podrá contener en los objetos (instancias).



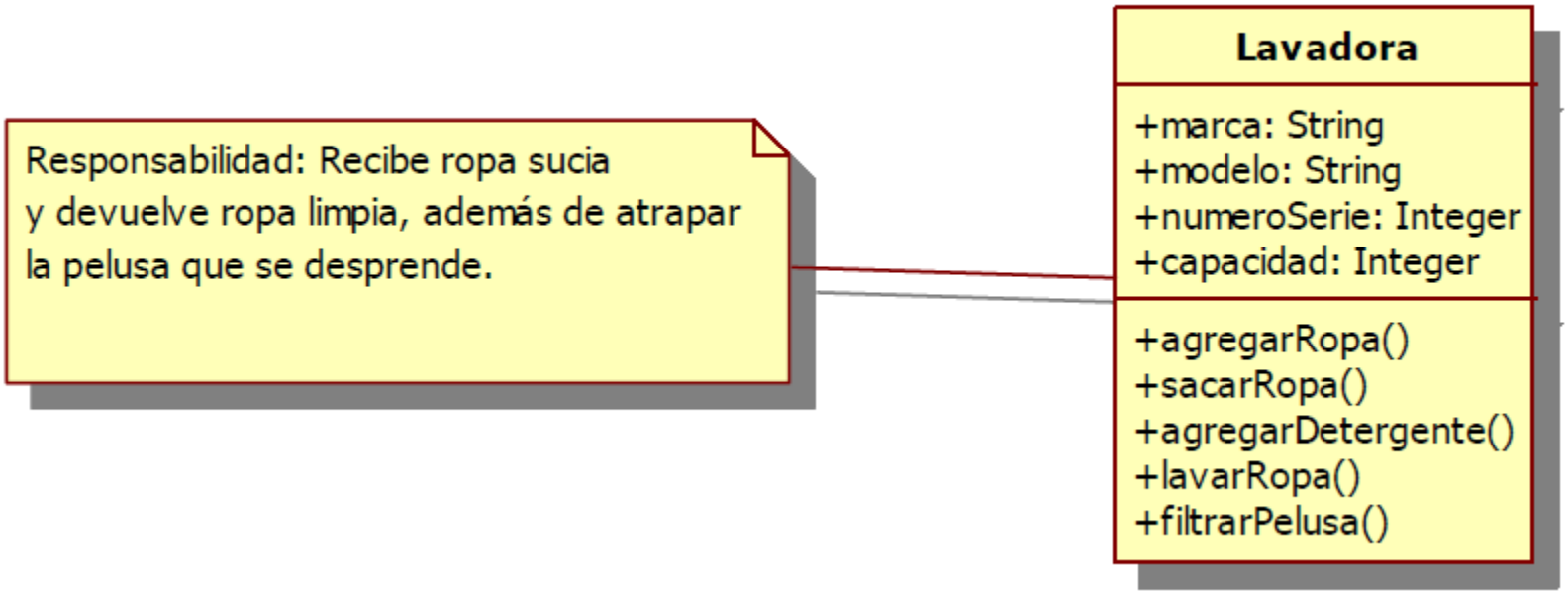
ATRIBUTOS Y MÉTODOS (2/3)

- Las **operaciones**, también llamadas **métodos** o **servicios**, son algo que la clase puede hacer o que nosotros u otra clase pueden hacer a una clase.



ATRIBUTOS Y MÉTODOS (3/3)

- La **responsabilidad** es una descripción de lo que hará la clase, es decir, de lo que sus atributos y operaciones intentan realizar en conjunto.



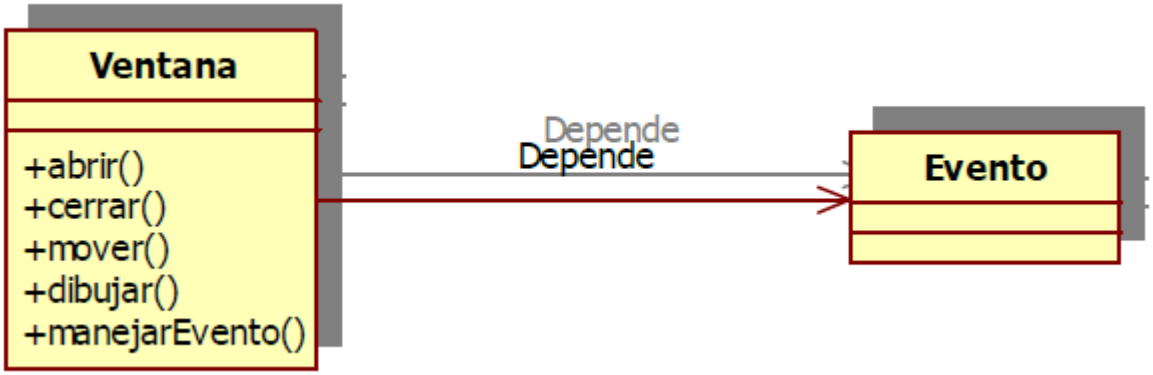
RELACIONES ESTÁTICAS Y DINÁMICAS ENTRE OBJETOS (1/6)

- Una **relación** es una conexión entre elementos. Las tres relaciones importantes son:

1. Dependencia
2. Generalización (herencia)
3. Asociación
   - Agregación
   - Composición

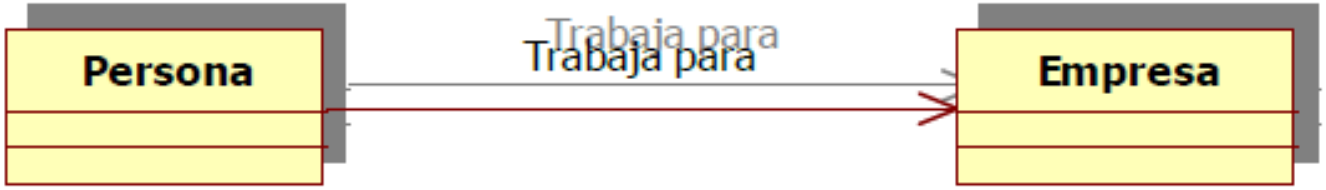
RELACIONES ESTÁTICAS Y DINÁMICAS ENTRE OBJETOS (2/6)

- Una **dependencia** es una relación de uso que declara que un cambio en la especificación de un elemento afecta a otro elemento que la utiliza, pero no necesariamente a la inversa.



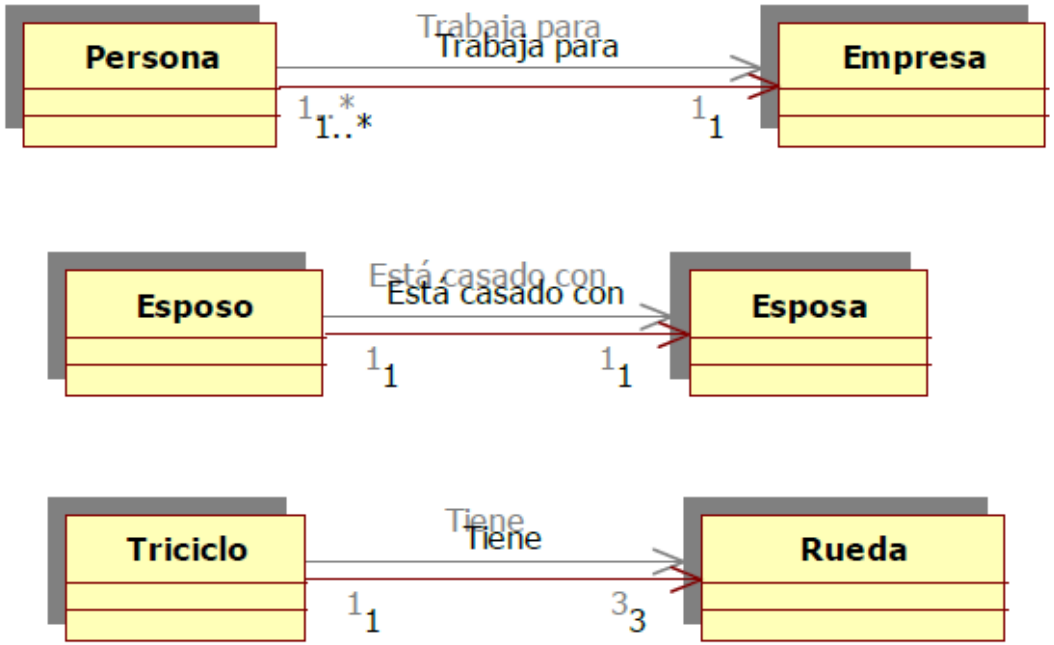
RELACIONES ESTÁTICAS Y DINÁMICAS ENTRE OBJETOS (3/6)

- Una **asociación** es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro.



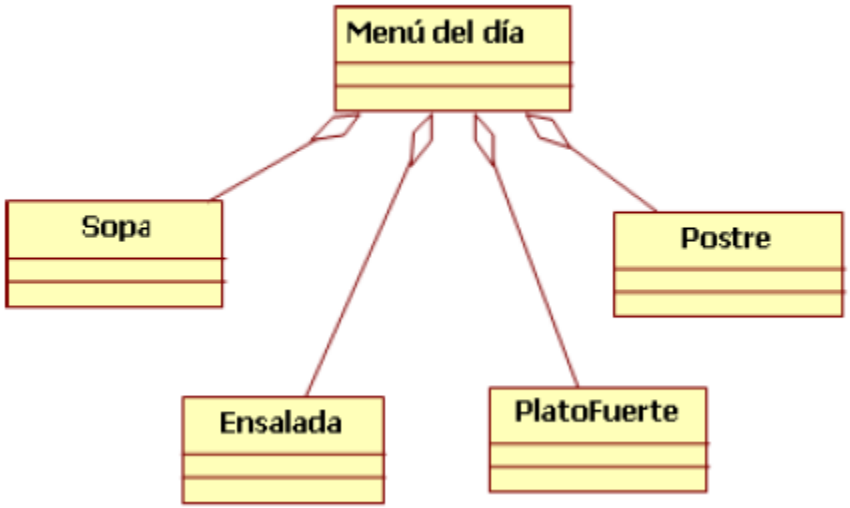
RELACIONES ESTÁTICAS Y DINÁMICAS ENTRE OBJETOS (4/6)

- La **Multiplicidad** es la cantidad de objetos de una clase que pueden relacionarse con un objeto de la clase asociada.



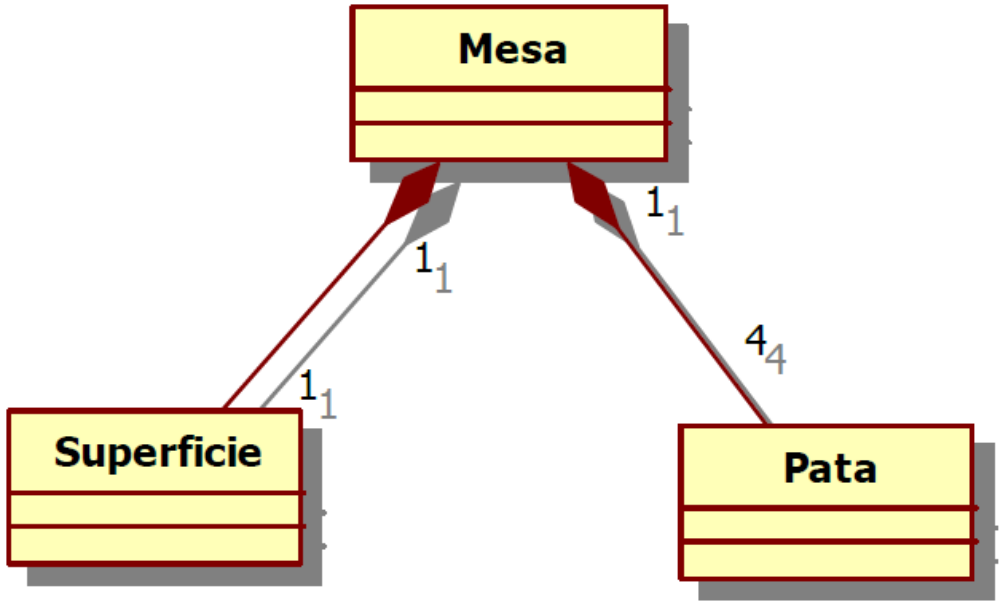
RELACIONES ESTÁTICAS Y DINÁMICAS ENTRE OBJETOS (5/6)

- La **Agregación** es asociación donde una clase consta de otras clases.



RELACIONES ESTÁTICAS Y DINÁMICAS ENTRE OBJETOS (6/6)

- Una **composición** es un tipo especial de agregación donde cada componente dentro de una composición puede pertenecer tan solo a un todo.



HERENCIA (GENERALIZACIÓN) (1/3)

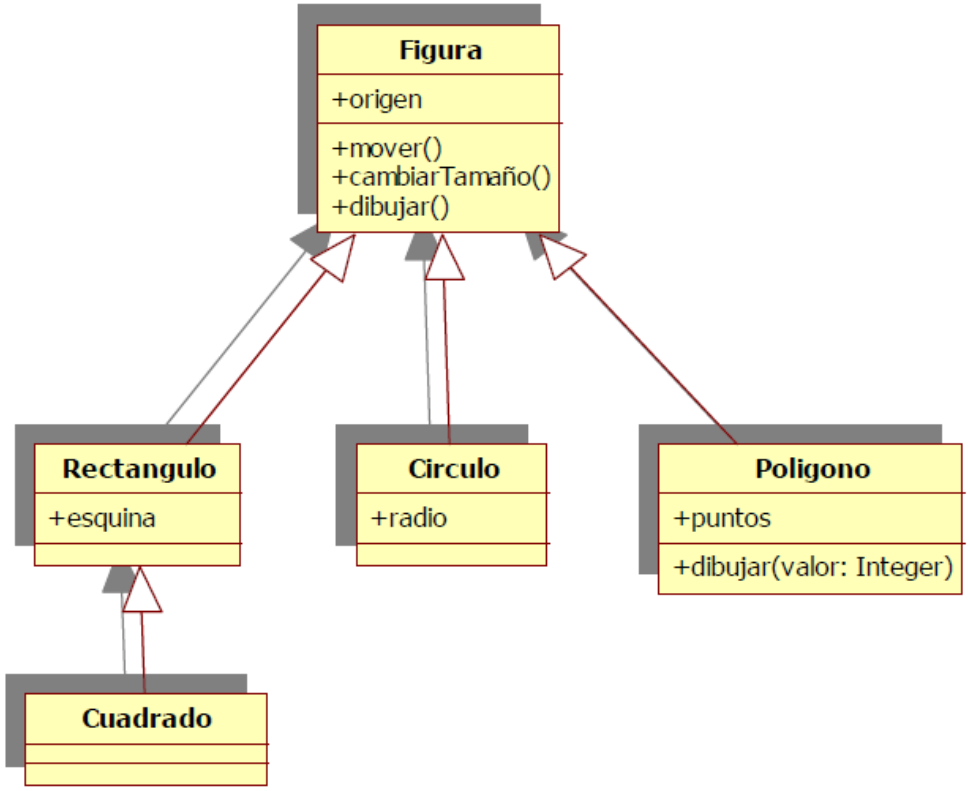
- Una **generalización** es una relación entre un elemento general (superclase o clase padre) y un caso más específico de ese elemento (subclase o clase hija).

- La generalización es la implementación de concepto **herencia**. En la generalización un hijo puede sustituir a padre.

- La clase hija hereda atributos y operaciones de su o sus padres.

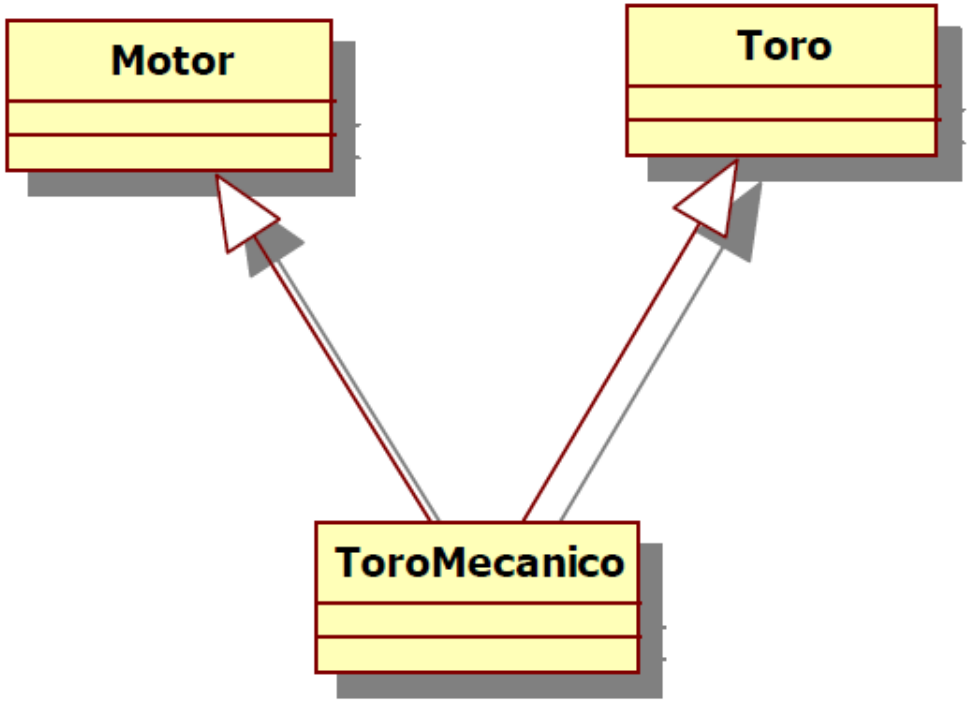
HERENCIA (GENERALIZACIÓN) (2/3)

- Una hija puede tener sus propios atributos y operaciones e incluso redefinir o implementar dichas operaciones a través del concepto de polimorfismo.



HERENCIA (GENERALIZACIÓN) (3/3)

- La **herencia múltiple** significa que una clase puede ser hija de dos o mas clases.



POLIMORFISMO (1/2)

- Es cuando las operaciones de los objetos de una misma clase tienen comportamientos diferentes. Incluso el comportamiento de la clase padre difiere del comportamiento de las clases hijas.

POLIMORFISMO (2/2)

Existen dos tipos de polimorfismo:

- **Sobrecarga**: es cuando un método (operación) de una clase recibe diferentes parámetros para realizar su tarea.

- **Sobreescritura**: es cuando el comportamiento de un método (operación) es diferente con respecto a las demás instancias o incluso al mismo padre. eso significa que redefine el código del método. Cuando esto pasa, es necesario definir el método padre como abstracto; esto es, no definir ningún comportamiento del padre, solo el nombre, para dejarlo a las instancias que lo definan según sus necesidades.

ENCAPSULAMIENTO (1/3)

- El **encapsulamiento** es el ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que sólo se pueda cambiar mediante las operaciones definidas para ese objeto.

- Esto permite reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Así **aumenta la cohesión** de los componentes del sistema.

ENCAPSULAMIENTO (2/3)

- Cada objeto esta asilado del exterior y expone una interfaz a otros objetos que describe cómo pueden interactuar con el.

- El aislamiento **protege a las propiedades de un objeto** contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado.

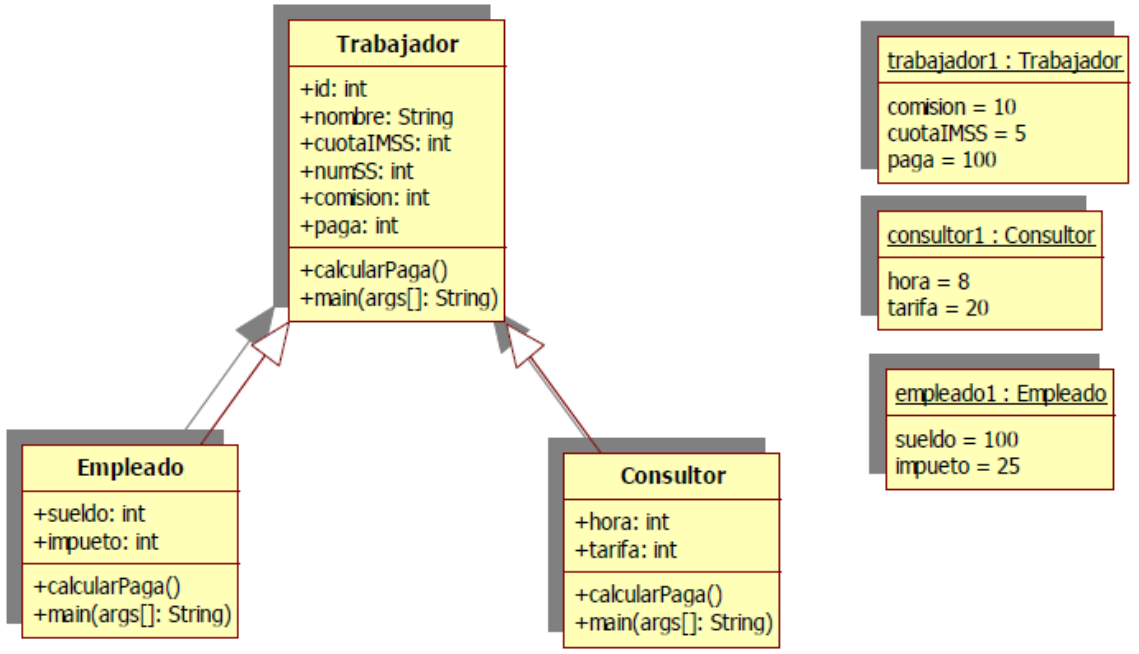
ENCAPSULAMIENTO (3/3)

Formas de encapsular:

- **Estándar** (predeterminado).
- **Abierto**: hace que el miembro de la clase pueda ser accedido desde el exterior de la Clase y cualquier parte del programa.
- **Protegido**: solo es accesible desde la Clase y las clases que heredan (a cualquier nivel).
- **Semicerrado**: solo es accesible desde la clase heredada.
- **Cerrado**: Solo es accesible desde la Clase.

DIAGRAMA DE CLASES (1/1)

- Un diagrama de clases es una representación gráfica que **modela las clases y sus relaciones**. La notación estándar para un diagrama de clases es la que define



CAMPOS DE APLICACIÓN (1/1)

- Los lenguajes orientados a objetos al igual que los imperativos pueden resolver prácticamente **cualquier problema.**

- Las aplicaciones pueden ser: controlar el plan de vuelo de una aeronave, aplicaciones para web, aplicaciones gráficas, dispositivos móviles, videojuegos, realidad virtual, etc.

REFERENCIAS (1/1)

1. Louden K,. Lambert K.(2011). Programming languages. Principles and Practice. 3nd Edition. Course Technology.
2. Pandey A.K (2008). Programming languages: principles and paradigms. Alpha Science Intl Ltd.
3. Sebesta, R.W. 2009. Concepts of programming languages. Addison-Wesley.
4. Tucker, A.; Noonan, R. 2006. Programming languages.. McGraw Hill.