



Instituto Politécnico Nacional



ESCOM "ESCUELA SUPERIOR DE CÓMPUTO"

ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS


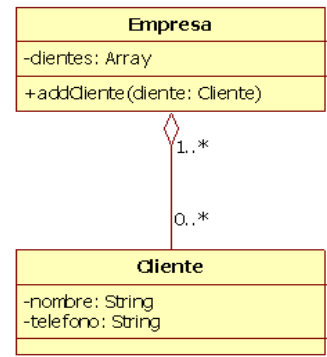
RELACIONES ENTRE CLASES/


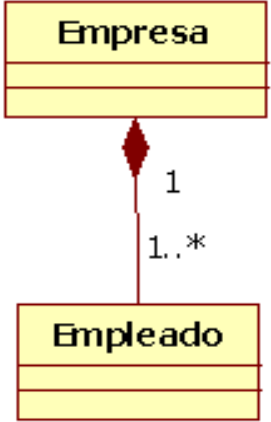

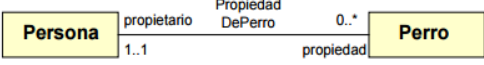
PROFA: Reyna Melara Abarca


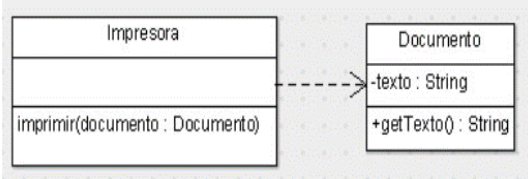

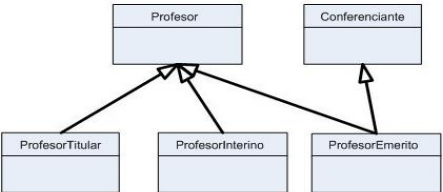
ALUMMNO: Rojas Alvarado Luis Enrique

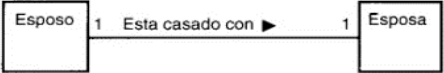
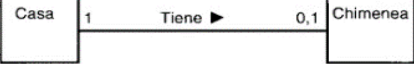
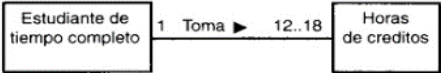

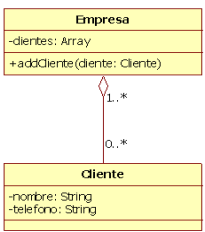

CORREO: lrojase1@hotmail.com

GRUPO: 2CM9

Relacion entre clases				
Tipo	Aspectos que modela	Notacion en UML	Ejemplos usando UML	Ejemplo en codigo
Agregacion	Esta relacion se presenta entre una clase TODO y una clase Parte que es componente de TODO		 <pre> classDiagram class Empresa { -clientes: Array +addCliente(diente: Cliente) } class Cliente { -nombre: String -telefono: String } Empresa "1..*" -- "0..*" Cliente </pre>	<p>Contacto.php</p> <pre> 1 <?php 2 /* Clase Contacto */ 3 Class Contacto 4 { 5 private \$_nombre; 6 private \$_telefono; 7 8 public function __construct(\$nombre, \$telefono) { 9 \$this->_nombre = \$nombre; 10 \$this->_telefono = \$telefono; 11 } 12 } 13 >> </pre> <p>Agenda.php</p> <pre> 1 <?php 2 /* Incluimos la clase Contacto */ 3 require_once 'Contacto.php'; 4 5 /* Clase Agenda */ 6 class Agenda 7 { 8 private \$_contactos = array(); 9 10 public function __construct() { 11 } 12 13 public function addContacto(Contacto \$contacto) 14 { 15 \$this->_contactos[] = \$contacto; 16 } 17 } 18 19 >> </pre> <p>En la practica:</p> <pre> 1 <?php 2 /* Creamos una instancia de Agenda */ 3 \$miAgenda = new Agenda(); 4 5 /* Agregamos algunos Contactos */ 6 \$miAgenda->addContacto(new Contacto("Pepe", "3-21-54-87")); 7 \$miAgenda->addContacto(new Contacto("Juan", "3-32-65-98")); 8 \$miAgenda->addContacto(new Contacto("Luis", "3-78-13-46")); 9 10 >> </pre>

Composicion	<p>Implica que los componentes de un objeto solo pueden pertenecer a un solo objeto agregado, de forma que cuando el objeto agregado es destruido todas sus partes son destruidas tambien</p>			<pre> Contacto.php 1 <?php 2 /* Clase Contacto */ 3 class Contacto 4 { 5 private \$ nombre; 6 private \$ telefono; 7 8 public function __construct(\$nombre, \$telefono) { 9 \$this-> nombre = \$nombre; 10 \$this-> telefono = \$telefono; 11 } 12 } 13 ?> </pre> <pre> Agenda.php 1 <?php 2 /* Incluimos la clase Contacto */ 3 require_once 'Contacto.php'; 4 5 /* Clase Agenda */ 6 class Agenda 7 { 8 private \$ contactos = array(); 9 10 public function __construct() { 11 } 12 13 public function addContacto(Contacto \$contacto) 14 { 15 \$this-> contactos[] = \$contacto; 16 } 17 } 18 ?> </pre> <p>En la practica:</p> <pre> 1 <?php 2 /* Creamos una instancia de Agenda */ 3 \$miAgenda = new Agenda(); 4 5 /* Agregamos algunos Contactos*/ 6 \$miAgenda->addContacto(new Contacto("Pepé", "3-21-54-87")); 7 \$miAgenda->addContacto(new Contacto("Juan", "3-32-85-98")); 8 \$miAgenda->addContacto(new Contacto("Luis", "3-78-13-46")); 9 ?> </pre>
Asociacion	<p>Es una relación de estructura entre clases, es decir, una entidad se construye a partir de otra u otras. Aunque este tipo de relación es mas fuerte que la Dependencia es más débil que la Agregación, ya que el tiempo de vida de un objeto no depende de otro.</p>			<pre> Chofer.java 1 /* Clase Chofer */ 2 class Chofer { 3 private String nombre; 4 5 public Chofer(String nombre) { 6 this.nombre = nombre; 7 } 8 9 public String getNombre() { 10 return this.nombre; 11 } 12 } </pre> <pre> Taxi.java 1 /* Clase Taxi */ 2 class Taxi { 3 private Chofer chofer; 4 private String matricula; 5 6 public Taxi(Chofer chofer, String matricula) { 7 this.chofer = chofer; 8 this.matricula = matricula; 9 } 10 11 public void printMatricula() { 12 System.out.println(this.matricula); 13 } 14 15 public void printChofer() { 16 String nombreChofer = this.chofer.getNombre(); 17 System.out.println(nombreChofer); 18 } 19 } </pre> <p>Solo bastaría instanciar las clases y hacer uso de sus métodos:</p> <pre> 1 Chofer miChofer = new Chofer("Pedro"); 2 Taxi miTaxi = new Taxi(miChofer, "AHD-1050"); 3 miTaxi.printChofer(); 4 miTaxi.printMatricula(); </pre>

Dependencia	Es una relación de uso entre dos clases (una usa a la otra). Esta relación es la más básica entre clases y comparada con los demás tipos de relación, la mas débil.		 <pre> classDiagram class Impresora { imprimir(documento : Documento) } class Documento { -texto : String +getTexto() : String } Impresora ..> Documento </pre>	<p>Documento.java</p> <pre> 1 /* Clase Documento */ 2 class Documento { 3 private String texto; 4 5 public Documento(String texto) { 6 this.texto = texto; 7 } 8 9 public String getTexto() { 10 return this.texto; 11 } 12 } </pre> <p>Impresora.java</p> <pre> 1 /* Clase Impresora */ 2 class Impresora { 3 4 public Impresora() { 5 6 } 7 8 public void imprimir(Documento documento) { 9 String texto = documento.getTexto(); 10 System.out.println(texto); 11 } 12 } </pre> <p>En funcionamiento:</p> <pre> 1 Documento miDocumento = new Documento("Hello World!"); 2 Impresora miImpresora = new Impresora(); 3 miImpresora.imprimir(miDocumento); </pre>
Herencia (Generalización y especialización)	La Generalización consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general. Los nombres usados: clase padre – clase hija. Otros nombres: superclase – subclase, clase base – clase derivada.		 <pre> classDiagram class Profesor class Conferenciante class ProfesorTitular class ProfesorInterino class ProfesorEmento Profesor < -- ProfesorTitular Profesor < -- ProfesorInterino Profesor < -- ProfesorEmento </pre>	<pre> public class SelecciónFutbol { protected int id; protected String Nombre; protected String Apellidos; protected int Edad; // constructor, getter y setter public void Concentrarse() { ... } public void Viajar() { ... } } </pre> <pre> public class Futbolista extends SelecciónFutbol { private int dorsal; private String demarcacion; public Futbolista() { super(); } // getter y setter public void jugarPartido() { ... } public void entrenar() { ... } } </pre> <pre> public class Entrenador extends SelecciónFutbol { private String idFederacion; public Entrenador() { super(); } // getter y setter public void dirigirPartido() { ... } public void dirigirEntreno() { ... } } </pre> <pre> public class Masajista extends SelecciónFutbol { private String Titulación; private int añosExperiencia; public Masajista() { super(); } // getter y setter public void darMasaje() { ... } } </pre>

Multiplicidad			
Tipo	Aspectos que m	Notacion en	Ejemplos usando UML
Reflexiva	uno y solo uno	1	 <p>uno a uno</p>
Binaria	cero o uno	0...1	 <p>uno a ninguno o un</p>
N-aria	Desde N hasta M	N...M	 <p>uno a 12 hasta 18</p>
N-aria	Cero o varios	*	 <p>uno a muchos</p>
N-aria	Cero o varios	0...*	
N-aria	o o varios(al menos un	1...*	 <p>uno a muchos</p>