Object Oriented Programming (a.k.a. OOP)

Digital House
Leo Lob - 2018

Qué NO es la OOP?

- La OOP NO es un lenguaje de programación.
- La OOP NO es una tecnología.
- La OOP NO es algo que debe aplicarse obligatoriamente.
- La OOP NO cura el cáncer, ni da vidas para el Candy Crush, ni permite obtener Big Macs gratis.

Entonces… qué es OOP?

- Es una forma de pensar el desarrollo de software.
- Se intenta relacionar con el mundo real.
- Facilita el entendimiento de un problema o situación compleja.

Objetos

- Son entidades funcionales de software
- Tienen una clara y definida responsabilidad
- Son independientes entre sí
- Tienen código y datos

Clases: qué es eso?

- Una clase es un molde, o plantilla, que define las características y comportamiento de una entidad

- Una clase es algo absolutamente abstracto, intangible, que existe solamente como concepto

- Todos los objetos se crean a partir de la definición de una clase.

- A la acción de crear un objeto en base a una clase se la llama "***Instanciar***".

Instanciar?

- Instanciar es crear un objeto a partir de una clase.
- Instanciar es presionar un botón mágico sobre el plano de una casa, para que se construya la casa.
- Una clase se puede instanciar muchas veces, obteniendo entonces muchos objetos de esa misma clase.
- La instanciación puede requerir parámetros, o no.

Atributos

- Los atributos son características de los objetos de una clase.
- Son propiedades que pueden tener un valor distinto (o no) entre distintos objetos de una misma clase.
- Todos los objetos de una misma clase tienen los mismos atributos, aunque podrían tener distinto valor
- Ejemplos: ***MiCancion.Titulo; UnaPelicula.Director, EquipoGanador.Goles;***

Métodos

- Son funciones, procedimientos, o rutinas de código que los objetos de una clase saben ejecutar.
- Pueden requerir parámetros, o no.
- Pueden devolver un valor, o ningún valor.
- Ejemplos: ***Canciones.MasVendida();***
  Peliculas.LasDeUnPais(“Uzbekistan”);
  Autos.ObtenerTodos(“Citroen”, 5, 100000);
  MiCancion.Reproducir(3);
  ElArchivo.GrabarRegistro( );
  LaFactura.ImprimirDetalle(1, “ABC”, True);
  ArticuloVendido.ModificarPrecio( );

Notación UML

Ejemplos

Coche
- cantidadNafta : Int - patente : String - precio : Double
+ encenderMotor () : void + incrementarVelocidad (int nuevaVelocidad): Boolean + eyectarAsiento (int altura) : void + getCantidadNafta ( ) : Int + getPatente ( ) : String

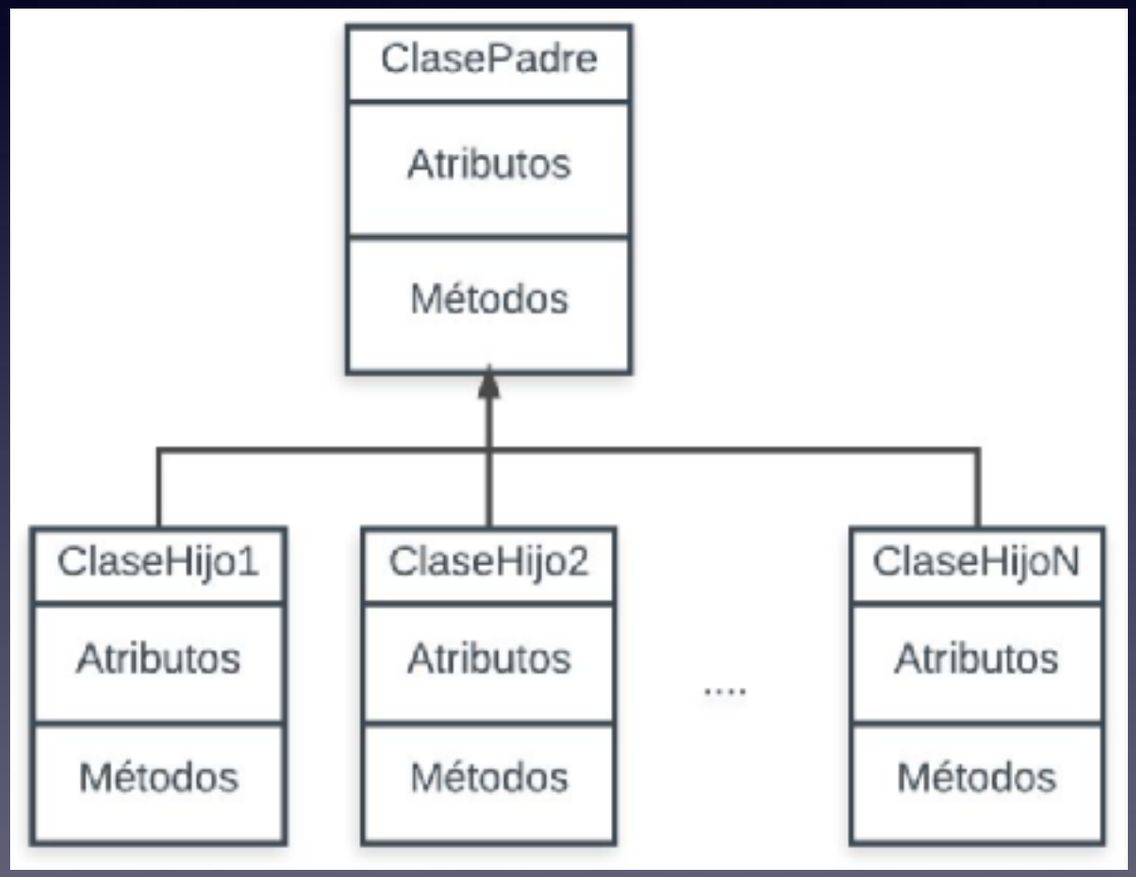
Principios fundamentales

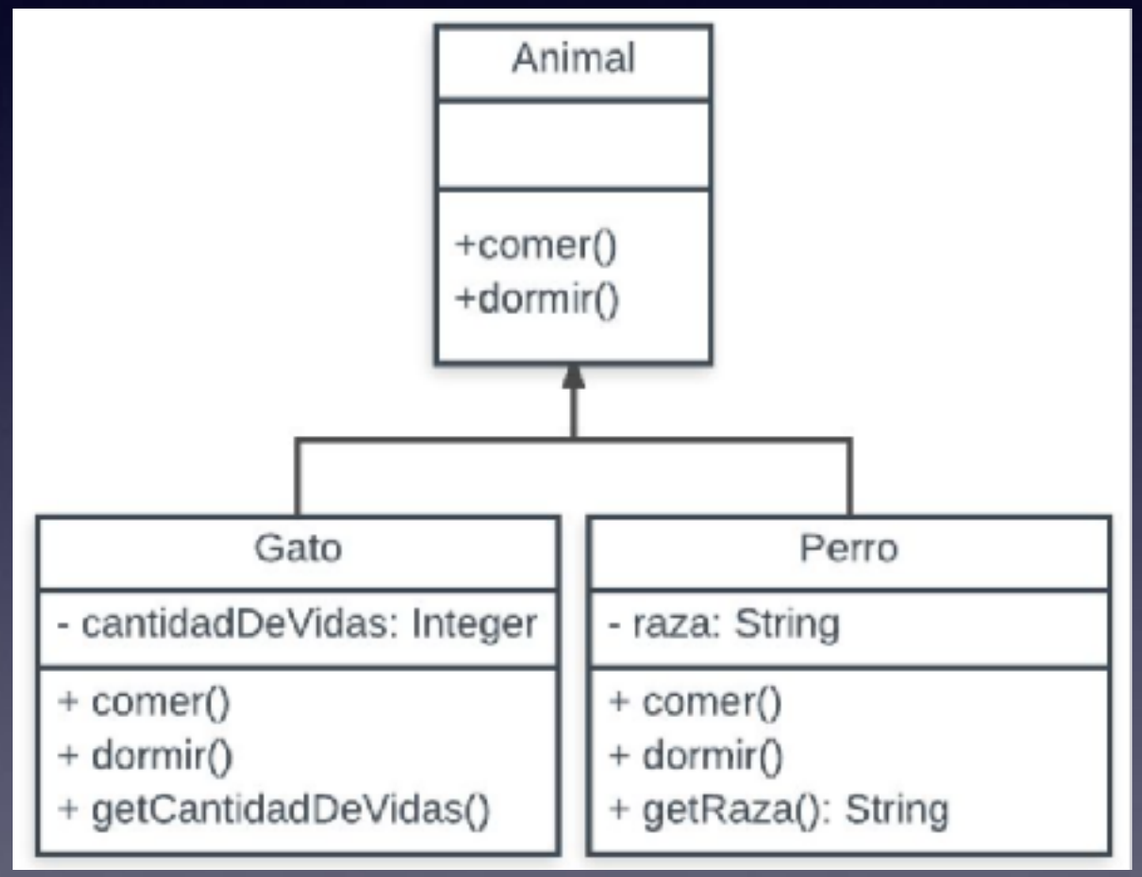
- Abstracción
- Encapsulamiento
- Herencia
- Polimorfismo

Herencia

- Especificación vs. generalización
- Evitar repeticiones de código
- Establece prioridades y jerarquías
- Permite concentrase únicamente en las particularidades de los herederos.

Herencia: notación UML





Clases abstractas

- Son clases que NO pueden ser instanciadas
- Son clases que DEBEN ser heredadas
- Se usan para obligar a que sus métodos sean programados

Tipos de atributos

- Públicos
- Privados
- Protegidos

Declarar vs. Instanciar

- **Declarar** no es lo mismo que **instanciar**.

- Al declarar, lo único que estamos haciendo es decirle al compilador que, dentro de un rato, cuando se nos de la gana, vamos a usar un objeto de *tal clase*, y nos gustaría muchísimo que nos reserve un espacio en la memoria.

- El compilador no nos permite declarar una variable que ya estaba declarada.

- La declaración no es una instrucción ejecutable. Por eso, puede hacerse fuera de las llaves delimitantes de un método o función.

- La sintaxis de la declaración es, siempre, ***<Tipo> <Nombre del objeto>***

- Ejemplos: ***int Numero; String Nombre; Cancion MiTemaPreferido;***

Declarar vs. Instanciar

- ***Instanciar*** es el proceso de crear un objeto de una clase.

- Es una instrucción ejecutable, por lo que debe estar dentro de llaves de bloque de ejecución.

- Es una invocación a alguno de los constructores de la clase.

- Solo puede instanciarse un objeto que esté previamente declarado.

- Un objeto ya instanciado puede volverse a instanciar cuantas veces querramos.

- Ejemplos: ***MiCancionFavorita = New Cancion();***

Asociación vs. Composición

- Asociación: “Usa un…”
- Composición: “Se compone de…”

Asociación

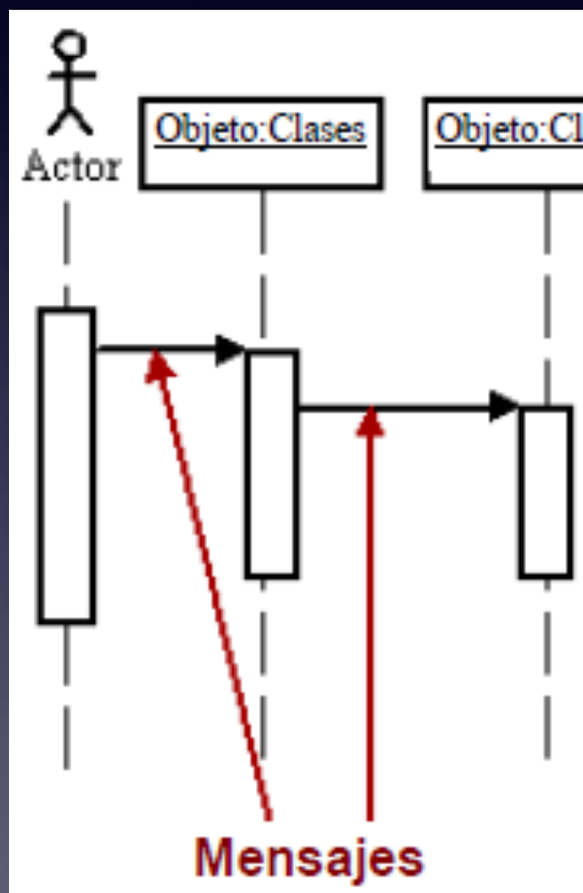
- Un profesor dicta un curso
- Un alumno asiste a un curso
- Un curso es dictado por un profesor, y asistido por varios alumnos.

Composición

- Un auto se compone de ruedas, motor y carrocería.

- Una rueda sirve para componer un auto.

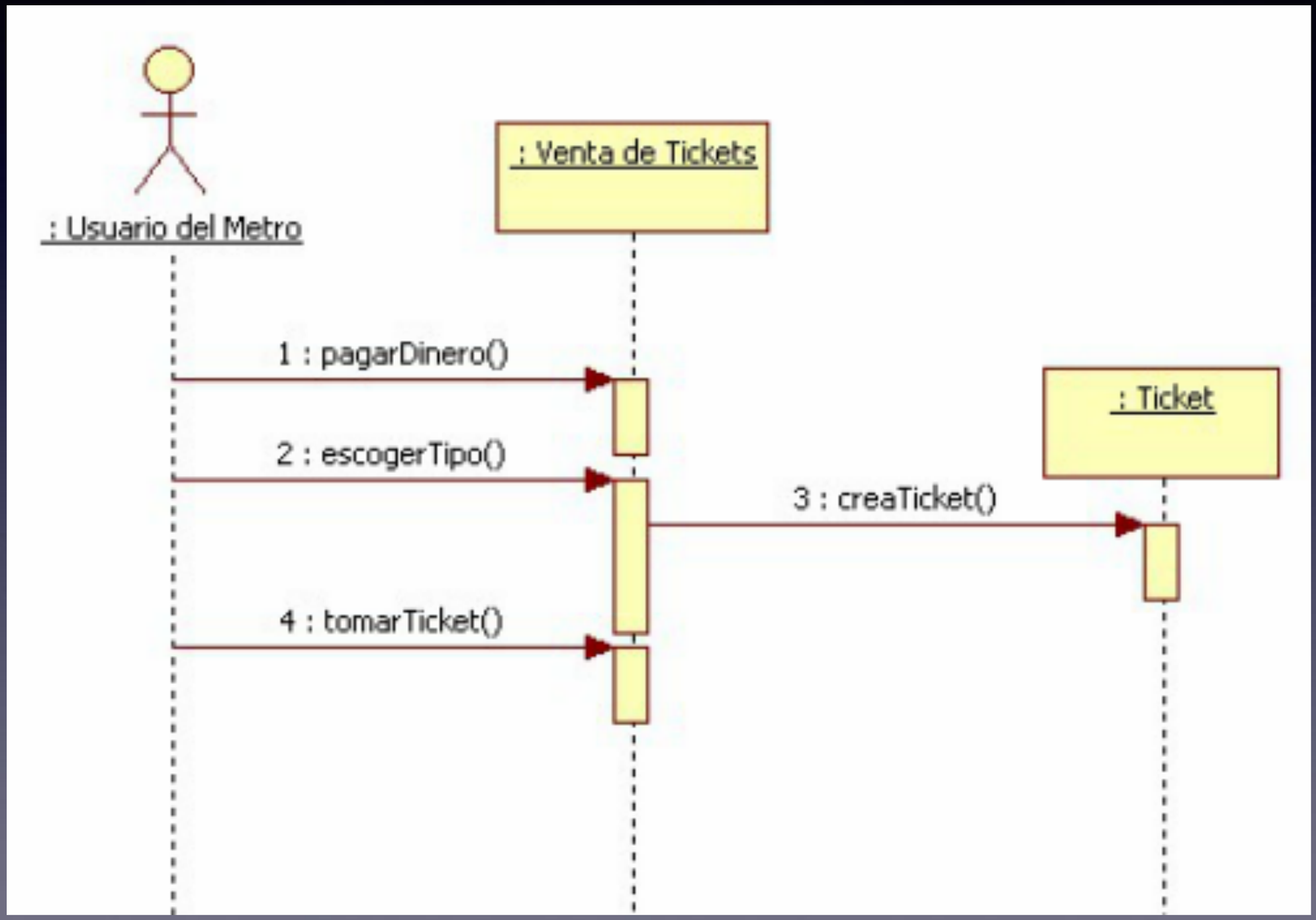
UML: Diagrama de secuencias



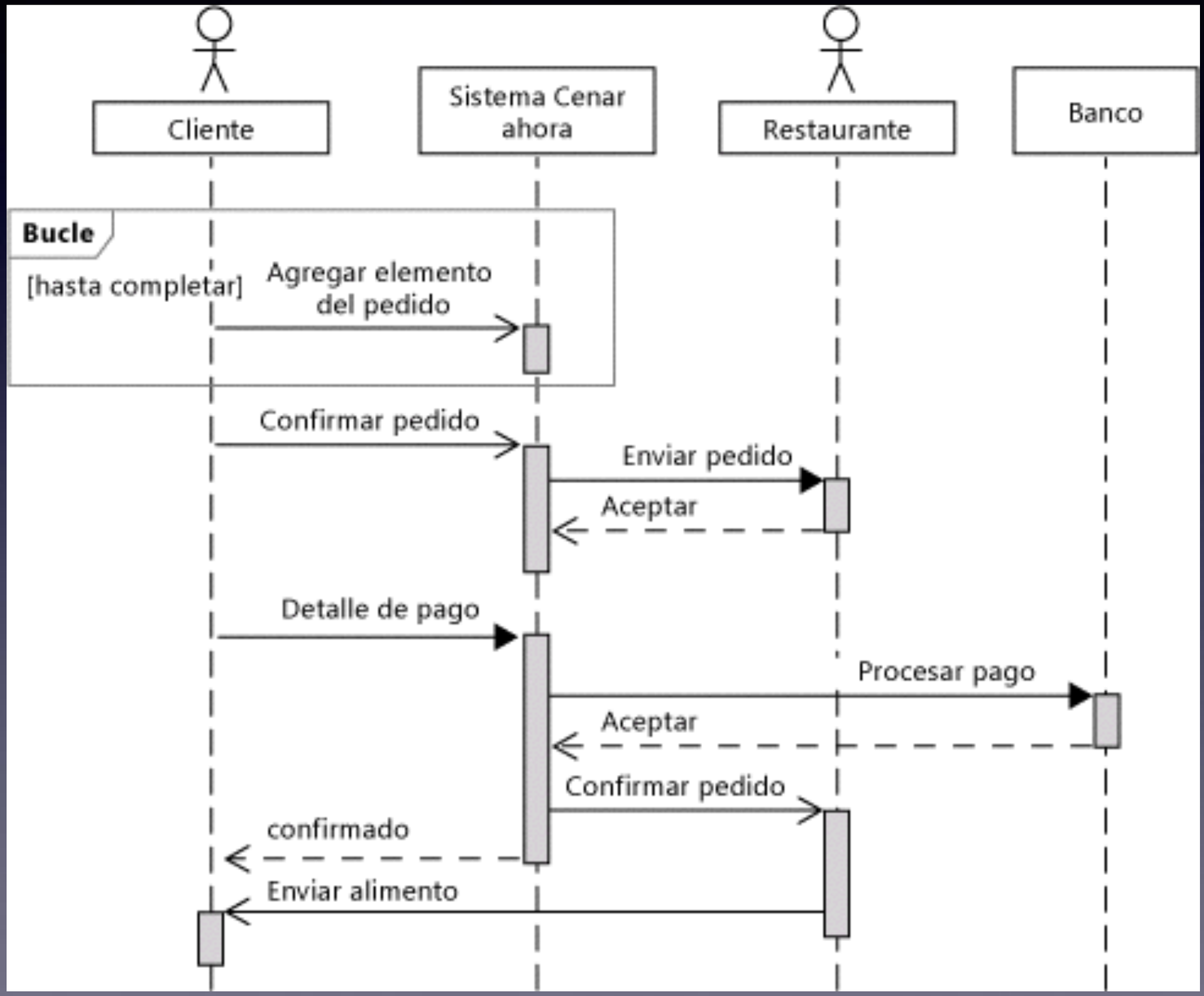
Líneas que se pueden utilizar para representar los mensajes

Flecha	Tipo de mensaje
→	Simple
→	Sincrónico
→	Asincrónico
↩	Rechazado
🕒→	Time out

UML: Diagrama de secuencias



UML: Diagrama de secuencias



Ejercicio

- UML - Diagrama de clases
- UML - Diagrama de secuencia

Clases en PHP

```
<?php
    class Coche
    {

    }
?>
```

Atributos

```
<?php
    class Coche
    {
        public $marca;
        public $patente;
        public $kilometraje;
    }
?>
```

Instanciación

```
<?php
    $AutoDeLeo = new Coche();
    $AutoDeLeo->marca = “Citroen”;
    $AutoDeLeo->color = “Gris”;
    $AutoDeLeo->kilometraje = 50000;

    $AutoDeGaby = new Coche();
    $AutoDeGaby->marca = “Volkswagen”;
    $AutoDeGaby->color = “Gris”;
    $AutoDeGaby->kilometraje = 80000;

?>
```

Lectura de atributos

```
<?php
    $AutoDeLeo = new Coche();
    $AutoDeLeo->marca = “Citroen”;
    $AutoDeLeo->color = “Gris”;
    $AutoDeLeo->kilometraje = 50000;

    echo "Tengo un {$AutoDeLeo->marca} de
 color  {$AutoDeLeo->color}"
?>
```

Funciones

```
<?php
    class Coche
    {
        public $marca;
        public $patente;
        public $kilometraje;

        public function getMarca() {
          return $this->marca;
        }
    }
?>
```

Métodos

```php
<?php
    class Coche
    {
        public $marca;
        public $patente;
        public $kilometraje;

        public function setMarca($MarcaAAsignar) {
          $this->marca = $MarcaAAsignar;
        }

        public function FalsearKilometraje () {
          if ($this->kilometraje>10000) {
            $this->kilometraje = $this->kilometraje - 10000;
          }
        }
    }
?>
```

Invocación

```
<?php
   $AutoDeLeo = new Coche();
   $AutoDeLeo->setMarca(“Citroen”);
   $AutoDeLeo->color = “Gris”;
   $AutoDeLeo->kilometraje = 50000;
   $AutoDeLeo->FalsearKilometraje();

    echo “Tengo un auto marca {$AutoDeLeo->getMarca()} que cuando no está sucio se ve que es de color {$AutoDeLeo->color}” ;

?>
```

Constructores

- Es el fragmento de código de una clase que se ejecuta al instanciar un objeto.
- Puede recibir parámetros.
- Es lo que invocamos con el ***New***.
- Ejemplos: ***MiCancion=New Cancion("Love of my life");***

Constructor

```
<?php
    class Coche
    {
        public $marca;
        public $patente;
        public $kilometraje;

        public function __construct($MarcaInicial) {
            $this->marca = $MarcaInicial;
        }
    }
?>
```

Herencia

```
<?php
    class Vehiculo
    {
        public $marca;
        public $patente;
            public function EncenderMotor() {
            }
    }

    class Auto extends Vehiculo
    {


    }

?>
```

Herencia

```
<?php
    class Vehiculo
    {
       public $marca;
       public $patente;
             public function EncenderMotor() {
             }
    }

    class Auto extends Vehiculo
    {
       public $CantidadPuertas;
       public function $CambiarRueda() {  }
       public function EncenderMotor() { }

    }

?>
```

Singular vs. Plural

- Las clases en singular se refieren a un ejemplar del conjunto.
- Las clases en singular se llaman "***Clases de entidad***"
- Los objetos de una clase en singular no saben que existen otros objetos como ellos.
- Las clases de entidad se usan **siempre** instanciadas: ***$UnaPeli = new pelicula();***
- Las clases de entidad solo tienen métodos o funciones que se refieran al propio objeto, nunca al conjunto de objetos de la misma clase.
- ***MiCancion->Titulo***
- ***ElGranJugador->GolesConvertidos***
- ***ElMejorAlumno->getNumeroLegajo()***

Singular vs. **Plural**

- Las clases en plural se refieren **siempre** al Universo de objetos de esa clase.
- Las clases en plural se llaman "***Clases de universo***"
- Las clases de universo no se usan **nunca** instanciadas. O sea que sus métodos y funciones son siempre estáticos.
- Sus métodos o funciones pueden devolver valores de tipos nativos (Int, String, Bool), o bien objetos del tipo de la clase de entidad que manejan:
- ***Canciones.MayorDuracion()***
- ***Alumnos.HayReprobados***
- ***Docentes.MasCopado()***

Clases de universo

- Sus métodos, funciones y atributos deben llevar la cláusula ***static***:
  - *public static $Cantidad;*
  - *public static $TodasLasPeliculas;*
  - *public static function ObtenerTodas() { }*

Clases de universo

- Para referirse a un atributo de la clase (desde dentro de la propia clase), no se usa ***$this->***, sino ***self::***

- *if (! isset (self::$TodasLasPeliculas)) { }*

- *self::$Cantidad=count($PeliculasADevolver);*

Clases de universo

- Para referirse a un atributo o método de la clase (desde el exterior de la clase), también se usa el ***::***

- *$MisPeliculas=Peliculas::ObtenerTodas();*
- *$CantidadPeliculas=Peliculas::$Cantidad;*
- *$PeliDeMejorRating=Peliculas::getLaDeMejorRating();*