

2 Requisitos iniciales de planificación, Diagrama Casos de Uso – Ejemplos de Relaciones de dependencia y Actores

2.17 El digrafo para el caso de uso anterior en .dot:

```
digraph esq {  
    Cliente -> "Realizar pedido"  
    "Realizar pedido" -> "Verificar stock" [label="include"]  
    "Realizar pedido" -> "Aplicar descuento" [label="extend"]  
    Almacén -> "Verificar stock"  
}
```

Sin embargo no describiste el cliente "Almacen" te faltó la ponderación para Actor "principal" y Actor "Externo" Reescribir el .dot con estos agregados.

A continuación tienes:

1. El grafo .dot corregido y completo (con actor Almacén, y tipo de relación).

.esq.dot (actualizado)

```
digraph esq {  
    rankdir=LR  
    node [shape=box]  
  
    subgraph cluster_actores {  
        label="Actores"  
        style=dotted  
        Cliente [label="Cliente\\n(Principal)"]  
        Almacen [label="Almacén\\n(Externo)"]  
    }  
  
    node [shape=ellipse]  
    "Realizar pedido"  
    "Verificar stock"  
    "Aplicar descuento"  
  
    // relaciones con su tipo  
    Cliente -> "Realizar pedido" [label="Principal", style=bold]  
    Almacen -> "Verificar stock" [label="Externo", style=solid]  
  
    // include / extend  
    "Realizar pedido" -> "Verificar stock" [label="include", style=dashed]  
    "Realizar pedido" -> "Aplicar descuento" [label="extend", style=dashed]  
}
```

Edotor — Your Favorite Online Graph Editor

edotor.net

Edotor Load Sample Download Engine: dot

Graphviz Documentation Issues Copy Share Link

```

1  digraph esq {
2    rankdir=LR
3    node [shape=box]
4
5    subgraph cluster_actores {
6      label="Actores"
7      style=dotted
8      Cliente [label="Cliente\n(Principal)"]
9      Almacen [label="Almacén\n(Externo)"]
10 }
11
12 node [shape=ellipse]
13 "Realizar pedido"
14 "Verificar stock"
15 "Aplicar descuento"
16
17 // relaciones con su tipo
18 Cliente -> "Realizar pedido" [label="Principal", style=bold]
19 Almacen -> "Verificar stock" [label="Externo", style=solid]
20
21 // include / extend
22 "Realizar pedido" -> "Verificar stock" [label="include",
23   style=dashed]
24 "Realizar pedido" -> "Aplicar descuento" [label="extend",
25   style=dashed]

```

The diagram illustrates the interactions defined in the UML-like sequence diagram:

- Actores Cluster:** A dashed box labeled "Actores" contains two nodes: "Cliente (Principal)" and "Almacen (Externo)".
- Operations:** Four operations are shown as ellipses: "Realizar pedido", "Verificar stock", "Aplicar descuento", and another "Realizar pedido".
- Relationships:**
 - A solid arrow labeled "Principal" connects "Cliente (Principal)" to the first "Realizar pedido".
 - A solid arrow labeled "Externo" connects "Almacen (Externo)" to the second "Realizar pedido".
 - A dashed arrow labeled "include" connects the first "Realizar pedido" to "Verificar stock".
 - A dashed arrow labeled "extend" connects the first "Realizar pedido" to "Aplicar descuento".

CasoUsoPonderado.svg

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg"
      width="700" height="320" viewBox="0 0 700 320">

<style>
    .actor   { fill:#fff; stroke:#000; stroke-width:1.5; }
    .uc      { fill:#f2f2f2; stroke:#0366d6; stroke-width:2; }
    .bold    { stroke:#d32f2f; stroke-width:3; }           /* Principal */
    .solid   { stroke:#424242; stroke-width:1.5; }         /* Externo */
    .dash    { stroke-dasharray:6,4; stroke:#424242; stroke-width:1.5; fill:none; }
    .label   { font-family:Arial, sans-serif; font-size:14px; fill:#000; text-
anchor:middle; }
    .stereo  { font-family:Arial, sans-serif; font-size:13px; fill:#000; font-
weight:bold; }
</style>

<!-- Marco cluster "Actores" (simula el subgraph) -->
<rect x="10" y="10" width="120" height="300" rx="5" ry="5" fill="none"
stroke="#999" stroke-dasharray="4,2"/>
<text x="70" y="30" class="label">Actores</text>

<!-- ACTORES -->
<!-- Cliente (Principal) -->
<circle cx="40" cy="80" r="12" class="actor"/>
<line x1="40" y1="92" x2="40" y2="130" class="actor"/>
<line x1="28" y1="100" x2="52" y2="100" class="actor"/>
<line x1="40" y1="130" x2="28" y2="150" class="actor"/>
<line x1="40" y1="130" x2="52" y2="150" class="actor"/>
<text x="40" y="170" class="label">Cliente (Principal)</text>

<!-- Almacén (Externo) -->
<circle cx="40" cy="220" r="12" class="actor"/>
<line x1="40" y1="232" x2="40" y2="270" class="actor"/>
<line x1="28" y1="240" x2="52" y2="240" class="actor"/>
<line x1="40" y1="270" x2="28" y2="290" class="actor"/>
<line x1="40" y1="270" x2="52" y2="290" class="actor"/>
<text x="40" y="310" class="label">Almacén (Externo)</text>

<!-- CASOS DE USO -->
<ellipse cx="240" cy="80" rx="60" ry="25" class="uc"/>
<text x="240" y="85" class="label">Realizar pedido</text>

<ellipse cx="440" cy="80" rx="60" ry="25" class="uc"/>
<text x="440" y="85" class="label">Verificar stock</text>

<ellipse cx="240" cy="220" rx="60" ry="25" class="uc"/>
<text x="240" y="225" class="label">Aplicar descuento</text>

<!-- COMUNICACIÓN Actor → Caso de uso -->
<!-- Principal (gruesa) -->
<line x1="52" y1="80" x2="180" y2="80" class="bold"/>
<!-- Externo (delgada) -->
<line x1="52" y1="220" x2="180" y2="220" class="solid"/>

<!-- INCLUDE -->
<path d="M 300 80 H 380" class="dash"/>
<text x="340" y="70" class="stereo">&lt;&lt; include &gt;&gt;</text>
```

```

<!-- EXTEND -->
<path d="M 240 105 V 140 H 180 V 195" class="dash"/>
<text x="240" y="150" class="stereo">&lt;&lt; extend &gt;&gt;</text>
</svg>

```

The screenshot shows a web-based UML diagram editor. On the left, there is an 'SVG Code' panel displaying the XML code for the diagram. On the right, there is a 'Live Preview' panel showing the resulting UML diagram.

SVG Code:

```

1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      width="700" height="320" viewBox="0 0 700 320">
4
5      <style>
6          .actor { fill:#fff; stroke:#000; stroke-width:1.5; }
7          .uc { fill:#f1f1f2; stroke:#0366d6; stroke-width:2; }
8          .bold { stroke:#d32f2f; stroke-width:3; } /* Principal */
9          .solid { stroke:#424242; stroke-width:1.5; } /* Externo */
10         .dash { stroke-dasharray:6.4; stroke:#424242; stroke-width:1.5; fill:none; }
11         .label { font-family:Arial, sans-serif, font-size:14px; fill:#000; text-align:center; }
12         .stereo { font-family:Arial, sans-serif, font-size:13px; fill:#000; font-weight:bold; }
13     </style>
14
15     <!-- Marco cluster "Actores" (simula el subgraph) -->
16     <rect x="10" y="10" width="120" height="300" rx="5" ry="5" fill="none" stroke="#999" stroke-dasharray="1 2"/>

```

Live Preview:

The diagram illustrates dependencies between actors and use cases:

- Actores** (Principal):
 - A stick figure actor labeled "cliente (Principal)" has a solid line dependency (stroke width 1.5) to the use case **Realizar pedido**.
 - A stick figure actor labeled "nacén (Externo)" has a solid line dependency to the use case **Aplicar descuento**.
- Realizar pedido** and **Verificar stock** are connected by a dashed line dependency (stroke width 2).
- Realizar pedido** and **Aplicar descuento** are connected by a dashed line dependency (stroke width 2).
- A dashed box labeled "Actores" encloses the **Realizar pedido** and **Verificar stock** use cases.
- Relationships are annotated with UML stereotypes: `<< include >>` and `<< extend >>`.

2.18 Escribe Notas de Clase con tu autoría explicando el concepto de relaciones de dependencia:

- que significa la relación de dependencia **include**
- relación de dependencia **extend**
- Otros tipos de relación existentes
- tres ejemplos mostrando los ejemplos en formato **.DOT** y **SVG** al mismo tiempo.

Notas de Clase - Relaciones de Dependencia en Diagramas de Casos de Uso UML

Autor: Kimi (K2) - 2024-06-08

- ¿Qué significa la relación `<< include >>`?

- Dependencia obligatoria:** cada vez que se ejecuta el caso de uso *base*, se ejecuta íntegramente el caso incluido.
- Objetivo:** reutilizar comportamiento común (ej. validar datos, verificar stock).
- Sentido:** *base* incluye a *auxiliar*.

(b) ¿Qué significa la relación << extend >>?

- **Dependencia condicional:** el caso *extensión aumenta* al *base solo si se cumple* una condición o punto de extensión.
 - **Objetivo:** agregar funcionalidad opcional sin modificar el caso base.
 - **Sentido:** *extensión extiende* a *base*.

(c) Otros tipos de relación en casos de uso

1. **Asociación** (línea continua): Actor participa en el caso de uso.
 2. **Generalización** (flecha abierta, triángulo): actor o caso de uso hijo hereda del padre.
 3. **Realización** (flecha punteada con triángulo abierto): caso de uso *realiza* una interfaz (uso poco frecuente).

(d) Tres ejemplos .DOT + SVG simultáneos

Ejemplo 1 – *include* obligatorio

dot

```
graph LR
    rankdir=LR
    node [shape=box] Cliente
    node [shape=ellipse] "Realizar pedido" "Verificar stock"
    Cliente -> "Realizar pedido"
    "Realizar pedido" -> "Verificar stock" [label="<< include >>", style=dashed]
}
```

SVG

```
<svg xmlns="http://www.w3.org/2000/svg" width="420" height="120" viewBox="0 0 420 120">
  <style>.actor{fill:#fff;stroke:#000;stroke-width:1.5}.uc{fill:#f2f2f2;stroke:#0366d6;stroke-width:2}.dash{stroke-dasharray:6,4;stroke:#424242;stroke-width:1.5;fill:none}.label{font-family:Arial;font-size:14px;fill:#000;text-anchor:middle}.stereo{font-weight:bold}</style>
  <circle cx="30" cy="60" r="10" class="actor"/><line x1="30" y1="70" x2="30" y2="90" class="actor"/><line x1="22" y1="76" x2="38" y2="76" class="actor"/><line x1="30" y1="90" x2="22" y2="100" class="actor"/><line x1="30" y1="90" x2="38" y2="100" class="actor"/><text x="30" y="115" class="label">Cliente</text>
  <ellipse cx="150" cy="60" rx="55" ry="20" class="uc"/><text x="150" y="65" class="label">Realizar pedido</text>
  <ellipse cx="320" cy="60" rx="55" ry="20" class="uc"/><text x="320" y="65" class="label">Verificar stock</text>
  <line x1="40" y1="60" x2="95" y2="60" stroke="#000" stroke-width="1.5"/><path d="M 205 60 H 265" class="dash"/><text x="235" y="52" class="stereo">&lt;&lt;include &gt;&gt;</text>
</svg>
```

Ejemplo 2 - extend condicional

dot

```
digraph {
    rankdir=LR
    node [shape=box] Cliente
    node [shape=ellipse] "Realizar pedido" "Aplicar descuento"
    Cliente -> "Realizar pedido"
    "Aplicar descuento" -> "Realizar pedido" [label="<< extend >>", style=dashed]
}
```

SVG

```
<svg xmlns="http://www.w3.org/2000/svg" width="420" height="220" viewBox="0 0 420 220">
    <style>.actor{fill:#fff;stroke:#000;stroke-width:1.5}.uc{fill:#f2f2f2;stroke:#0366d6;stroke-width:2}.dash{stroke-dasharray:6,4;stroke:#424242;stroke-width:1.5;fill:none}.label{font-family:Arial;font-size:14px;fill:#000;text-anchor:middle}.stereo{font-weight:bold}</style>
    <circle cx="30" cy="60" r="10" class="actor"/><line x1="30" y1="70" x2="30" y2="90" class="actor"/><line x1="22" y1="76" x2="38" y2="76" class="actor"/><line x1="30" y1="90" x2="22" y2="100" class="actor"/><line x1="30" y1="90" x2="38" y2="100" class="actor"/><text x="30" y="115" class="label">Cliente</text>
    <ellipse cx="150" cy="60" rx="55" ry="20" class="uc"/><text x="150" y="65" class="label">Realizar pedido</text>
    <ellipse cx="150" cy="160" rx="55" ry="20" class="uc"/><text x="150" y="165" class="label">Aplicar descuento</text>
    <line x1="40" y1="60" x2="95" y2="60" stroke="#000" stroke-width="1.5"/><path d="M 150 80 V 110 H 95 V 140" class="dash"/><text x="150" y="118" class="stereo">&lt;&lt; extend &gt;&gt;</text>
</svg>
```

Ejemplo 3 - generalización de actor

Dot

```
digraph {
    rankdir=LR
    node [shape=box]    Usuario
    node [shape=box]    "Usuario Registrado" [shape=box]
    node [shape=ellipse] "Consultar saldo"
    Usuario -> "Usuario Registrado" [arrowhead=empty, style=solid]
    "Usuario Registrado" -> "Consultar saldo"
}
```

SVG

```
<svg xmlns="http://www.w3.org/2000/svg" width="420" height="120" viewBox="0 0 420 120">
    <style>.actor{fill:#fff;stroke:#000;stroke-width:1.5}.uc{fill:#f2f2f2;stroke:#0366d6;stroke-width:2}.label{font-family:Arial;font-size:14px;fill:#000;text-anchor:middle}.gen{stroke:#000;stroke-width:1.5;fill:none}</style>
    <circle cx="30" cy="60" r="10" class="actor"/><line x1="30" y1="70" x2="30" y2="90" class="actor"/><line x1="22" y1="76" x2="38" y2="76" class="actor"/><line x1="30" y1="90" x2="22" y2="100" class="actor"/><line x1="30" y1="90" x2="38" y2="100" class="actor"/><text x="30" y="115" class="label">Usuario</text>
    <circle cx="150" cy="60" r="10" class="actor"/><line x1="150" y1="70" x2="150" y2="90" class="actor"/><line x1="142" y1="76" x2="158" y2="76" class="actor"/><line x1="150" y1="90" x2="142" y2="100" class="actor"/><line x1="150" y1="90" x2="158" y2="100" class="actor"/><text x="150" y="115" class="label">Usuario Registrado</text>
    <ellipse cx="320" cy="60" rx="55" ry="20" class="uc"/><text x="320" y="65" class="label">Consultar saldo</text>
    <line x1="40" y1="60" x2="135" y2="60" class="gen"/><polygon points="135,55 135,65 145,60" fill="#000"/><line x1="160" y1="60" x2="265" y2="60" stroke="#000" stroke-width="1.5"/>
</svg>
```

Conclusión

Las relaciones **<< include >>** y **<< extend >>** son **dependencias** que se **materializan con líneas punteadas** en UML; el formato .dot describe la lógica y el **SVG** la visualiza inmediatamente, sin depender de herramientas propietarias.

En el último ejemplo, en la línea 4: node [shape=box] "Usuario Registrado" [shape=box] Sobra el [shape=box]

Tienes razón: la línea 4 del .dot sobra porque ya está declarado node [shape=box] al principio.

Aquí tienes la **versión corregida** (más clara y compacta) y su **SVG** asociado.

.DOT corregido (Ejemplo 3 – generalización)

```
digraph {
    rankdir=LR
    node [shape=box]
    Usuario
    "Usuario Registrado"
    node [shape=ellipse]
    "Consultar saldo"

    // generalización (triángulo vacío)
    Usuario -> "Usuario Registrado" [arrowhead=empty, style=solid]
    "Usuario Registrado" -> "Consultar saldo"
}
```

SVG limpio (sin redundancia)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg"
      width="420" height="120" viewBox="0 0 420 120">

    <style>
        .actor { fill:#fff; stroke:#000; stroke-width:1.5; }
        .uc { fill:#f2f2f2; stroke:#0366d6; stroke-width:2; }
        .label { font-family:Arial, sans-serif; font-size:14px; fill:#000; text-anchor:middle; }
        .gen { stroke:#000; stroke-width:1.5; fill:none; marker-end:url(#arrow); }
    </style>

    <!-- Marcador para generalización (triángulo vacío) -->
    <defs>
        <marker id="arrow" markerWidth="10" markerHeight="10" refX="9" refY="3"
orient="auto">
            <path d="M0,0 L0,6 L9,3 z" fill="#fff" stroke="#000"/>
        </marker>
    </defs>

    <!-- Actores -->
    <circle cx="30" cy="60" r="10" class="actor"/>
    <line x1="30" y1="70" x2="30" y2="90" class="actor"/>
    <line x1="22" y1="76" x2="38" y2="76" class="actor"/>
    <line x1="30" y1="90" x2="22" y2="100" class="actor"/>
    <line x1="30" y1="90" x2="38" y2="100" class="actor"/>
    <text x="30" y="115" class="label">Usuario</text>

    <circle cx="150" cy="60" r="10" class="actor"/>
    <line x1="150" y1="70" x2="150" y2="90" class="actor"/>
```

```

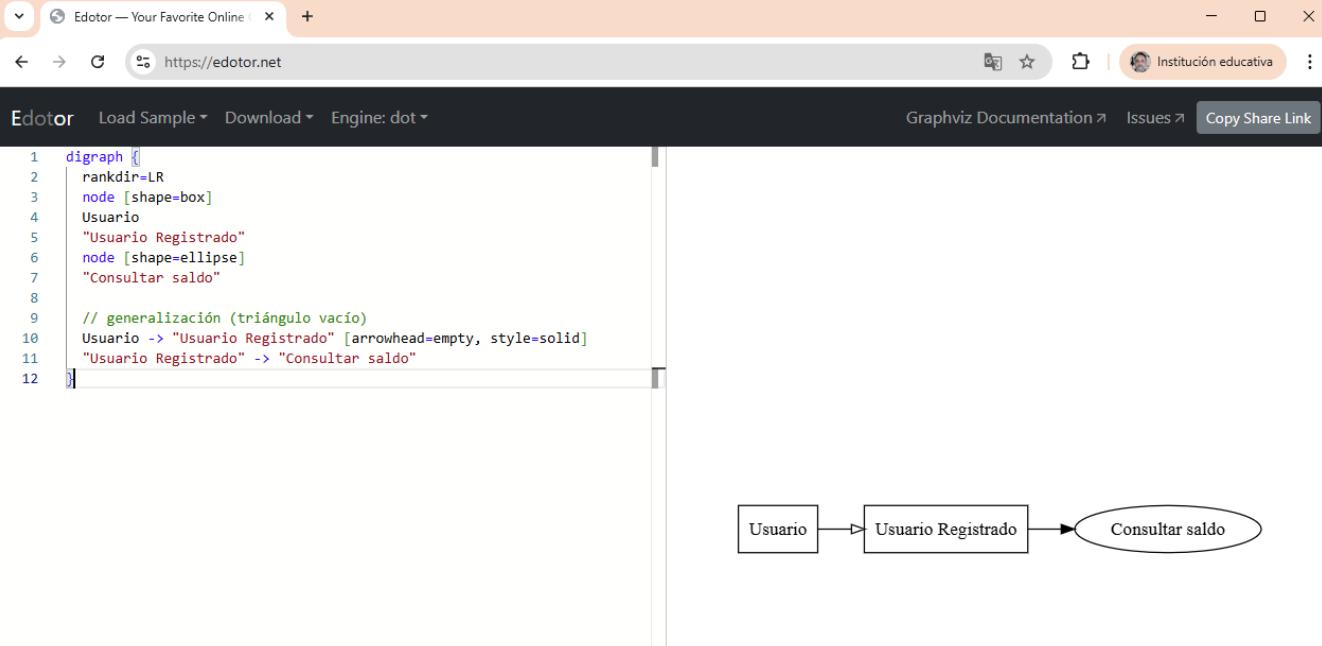
<line x1="142" y1="76" x2="158" y2="76" class="actor"/>
<line x1="150" y1="90" x2="142" y2="100" class="actor"/>
<line x1="150" y1="90" x2="158" y2="100" class="actor"/>
<text x="150" y="115" class="label">Usuario Registrado</text>

<!-- Caso de uso -->
<ellipse cx="320" cy="60" rx="55" ry="20" class="uc"/>
<text x="320" y="65" class="label">Consultar saldo</text>

<!-- Líneas -->
<line x1="40" y1="60" x2="135" y2="60" class="gen"/>
<line x1="160" y1="60" x2="265" y2="60" stroke="#000" stroke-width="1.5"/>

```

Listo: código más limpio, **sin atributos redundantes**, y SVG funcional que puedes abrir o editar en cualquier parte.



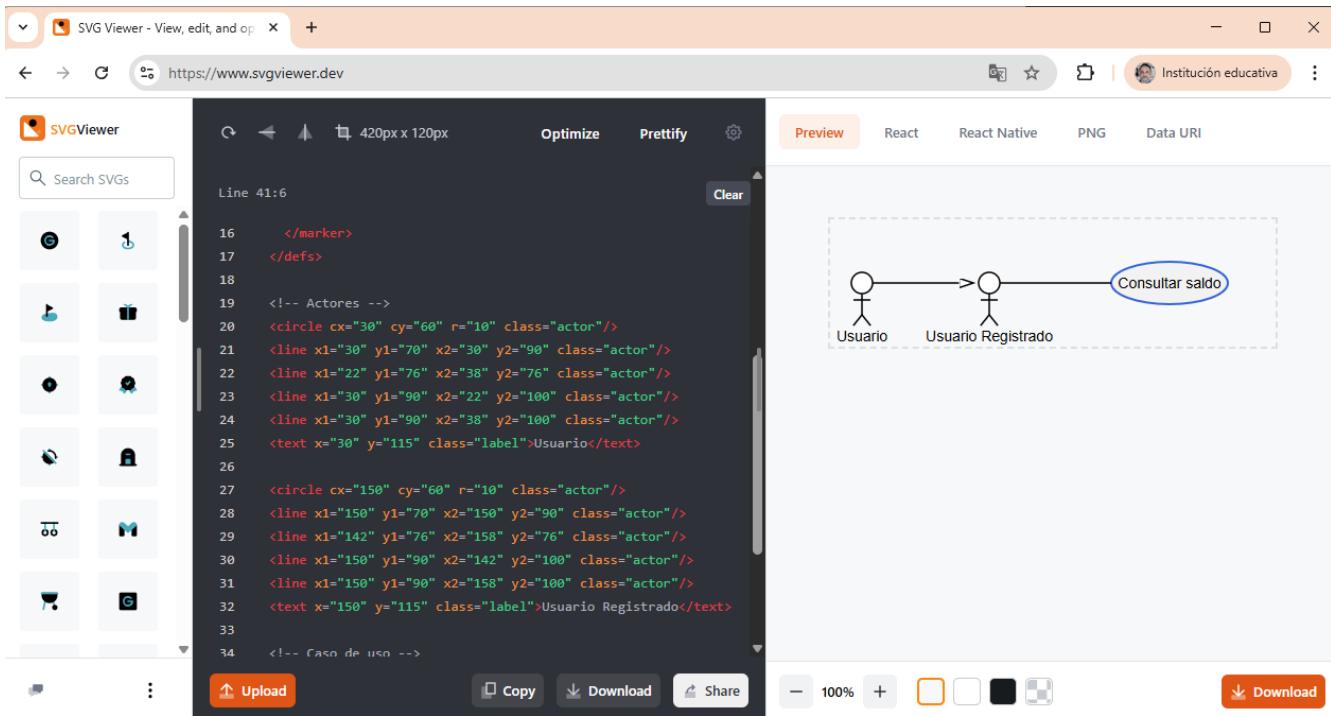
The screenshot shows the Edotor interface. At the top, there's a browser-like header with tabs, a search bar containing 'https://edotor.net', and a user icon labeled 'Institución educativa'. Below the header is a dark navigation bar with links for 'Edotor', 'Load Sample', 'Download', 'Engine: dot', 'Graphviz Documentation', 'Issues', 'Copy Share Link', and a magnifying glass icon. The main area contains two parts: a code editor on the left and a graph visualization on the right. The code editor displays the following dot language code:

```

1  digraph {
2    rankdir=LR
3    node [shape=box]
4    Usuario
5    "Usuario Registrado"
6    node [shape=ellipse]
7    "Consultar saldo"
8
9    // generalización (triángulo vacío)
10   Usuario -> "Usuario Registrado" [arrowhead=empty, style=solid]
11   "Usuario Registrado" -> "Consultar saldo"
12 }

```

The graph visualization on the right consists of three nodes: 'Usuario' (represented by a rectangle), 'Usuario Registrado' (represented by a rectangle), and 'Consultar saldo' (represented by an ellipse). Arrows connect 'Usuario' to 'Usuario Registrado' and 'Usuario Registrado' to 'Consultar saldo'. The arrow from 'Usuario' to 'Usuario Registrado' is styled with an empty arrowhead and a solid line.



2.19 Escribe Notas de clase de tu autoría explicando el concepto de actor en el diagrama de casos de uso explicando:

- (a) **Qué es Actor**
- (b) **Quienes son los actores principal y cuál es su función**
- (c) **Quienes son los actores externos y cuál es su función**
- (d) **Existe algún otro tipo de Actor**
- (e) **Existe el concepto de multiplicidad con los actores y si es así, con qué símbolo se expresa**

Notas de Clase - Actor en Diagramas de Casos de Uso UML

Autoría: Kimi (K2) - 2024-06-08

(a) ¿Qué es un Actor?

- **Definición UML:** un **rol** que un **usuario** o **sistema externo** desempeña al interactuar directamente con los casos de uso del sistema bajo estudio.
 - **No es una persona concreta;** es la **clasificación** del conjunto de usuarios que se comportan igual frente al software.
 - Se representa con un **palito y cabeza** (muñeco estilizado).
-

(b) Actor Principal (*Primary*)

- **Inicia** el flujo: es quien **invoca** el caso de uso para obtener un resultado **valioso para él**.
- Ej.: *Cliente* que presiona «Comprar»; *Estudiante* que se matricula.

- Función: proporcionar **valor de negocio**; sin él el caso de uso **no arranca**.
-

(c) Actor Externo (*Supporting / Secondary*)

- **No inicia** la interacción; **responde o aporta** información **requerida** por el caso de uso, pero **el beneficio no es para él**.
 - Ej.: *Pasarela de Pago, Servicio de Impresión, Almacén* que verifica stock.
 - Función: **posibilitar o completar** el flujo iniciado por el actor principal.
-

(d) ¿Existe otro tipo de Actor?

1. **Sistema / Temporizador / Reloj**: actor no humano que **desencadena** casos de uso a intervalos o eventos (ej.: *Proceso Batch* que genera facturas cada mes).
 2. **Actor abstracto** (generalización): representa **roles comunes** que luego se especializan (*Usuario → Usuario Registrado / Invitado*).
-

(e) ¿Multiplicidad de Actores?

- Sí. Indica **cuántos individuos** del rol pueden o deben participar **simultáneamente** en el caso de uso.
- **Símbolo**: se anota **cerca del actor** entre corchetes:
 - [1] → exactamente uno
 - [0..1] → cero o uno
 - [1..*] → uno o muchos
 - [*] → muchos (sin límite)

Ejemplo visual (en .dot y SVG):

dot

```
digraph {
    rankdir=LR
    node [shape=box]
    "Cliente[1..*]"
    node [shape=ellipse]
    "Realizar pedido"
    "Cliente[1..*]" -> "Realizar pedido"
}
```

SVG (fragmento)

```
<text x="40" y="30" class="label">Cliente [1..*]</text>
```

Conclusión: identificar **quién inicia, quién ayuda y cuántos** son, permite acotar correctamente el alcance funcional del sistema.

2.20 Proporciona tres ejemplos diferentes de Actores con el concepto de multiplicidad

Tres ejemplos **diferentes** que muestran **multiplicidad de actores** (con su **.dot** y **SVG** listos para copiar y visualizar).

Ejemplo 1 - Sistema de e-commerce

Regla: Cualquier número de clientes puede iniciar una compra.

- **Multiplicidad:** [*] (0 o muchos)

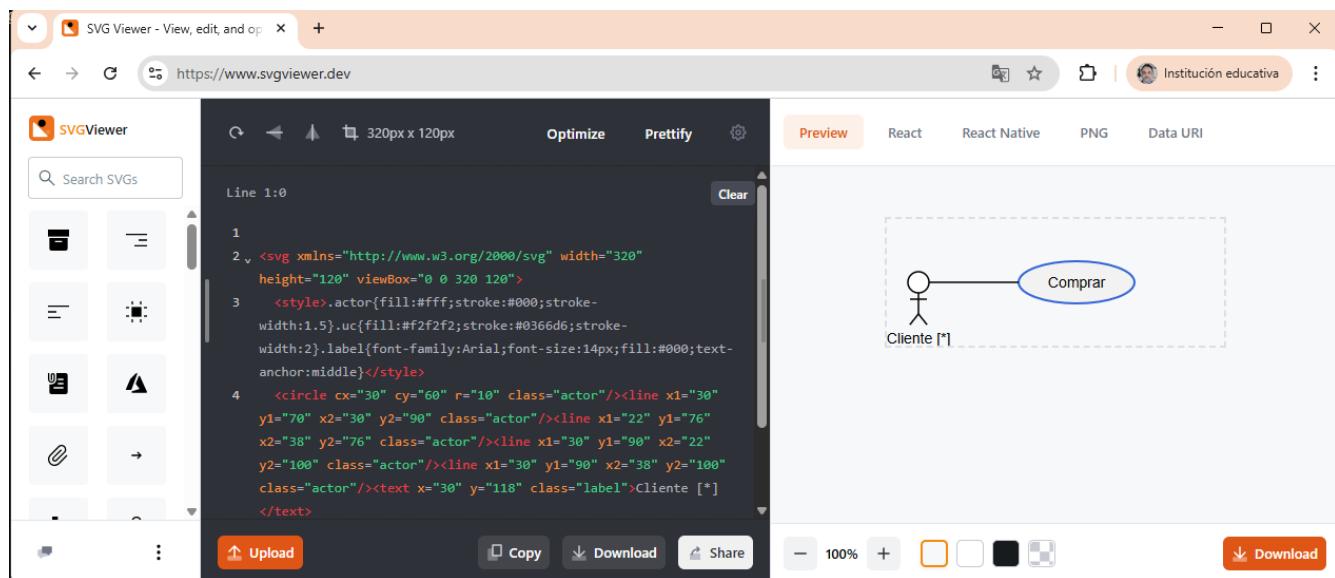
.DOT

```
digraph {
    rankdir=LR
    node [shape=box]
    "Cliente[*]"
    node [shape=ellipse]
    "Comprar"
    "Cliente[*]" -> "Comprar"
}
```



SVG

```
<svg xmlns="http://www.w3.org/2000/svg" width="320" height="120" viewBox="0 0 320 120">
  <style>.actor{fill:#fff;stroke:#000;stroke-width:1.5}.uc{fill:#f2f2f2;stroke:#0366d6;stroke-width:2}.label{font-family:Arial;font-size:14px;fill:#000;text-anchor:middle}</style>
  <circle cx="30" cy="60" r="10" class="actor"/><line x1="30" y1="70" x2="30" y2="90" class="actor"/><line x1="22" y1="76" x2="38" y2="76" class="actor"/><line x1="30" y1="90" x2="22" y2="100" class="actor"/><line x1="30" y1="90" x2="38" y2="100" class="actor"/><text x="30" y="118" class="label">Cliente [*]</text>
  <ellipse cx="180" cy="60" rx="55" ry="20" class="uc"/><text x="180" y="65" class="label">Comprar</text><line x1="40" y1="60" x2="125" y2="60" stroke="#000" stroke-width="1.5"/>
</svg>
```



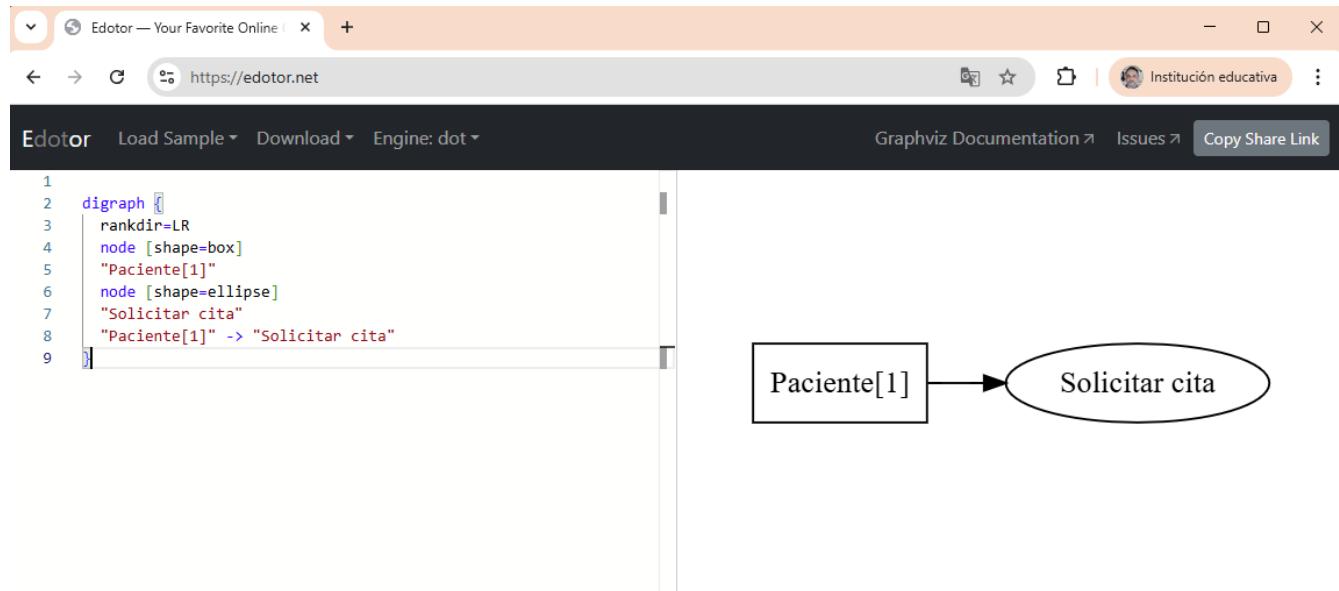
Ejemplo 2 - Sistema de citas médicas

Regla: Exactamente UN paciente debe solicitar la cita.

- **Multiplicidad:** [1]

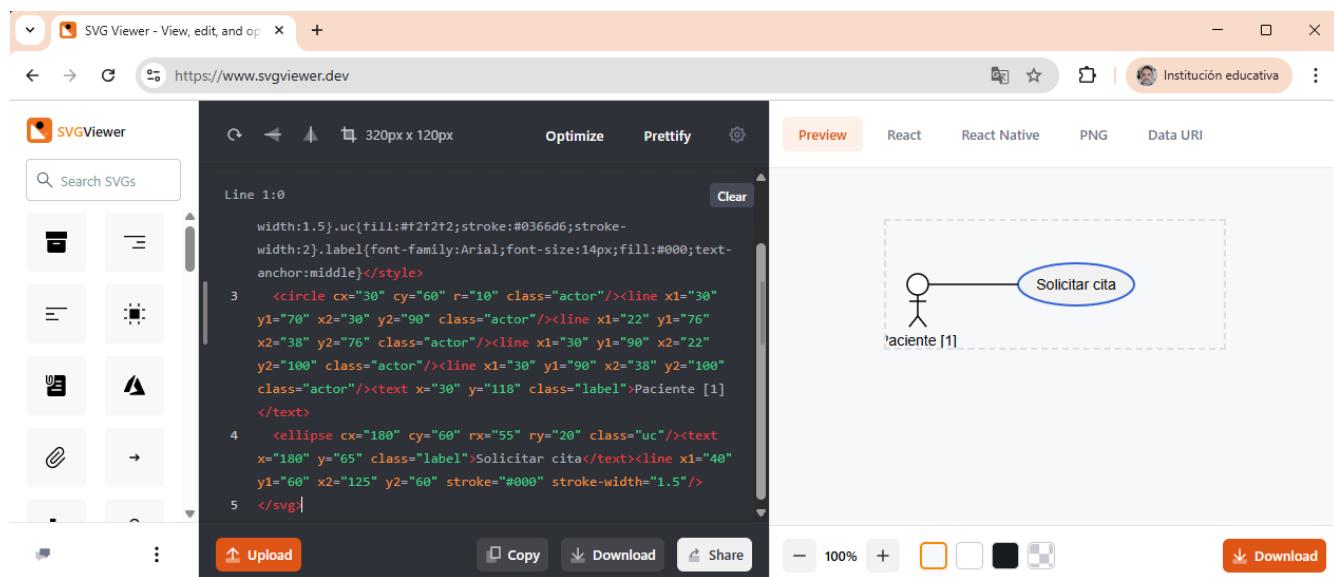
dot

```
digraph {
    rankdir=LR
    node [shape=box]
    "Paciente[1]"
    node [shape=ellipse]
    "Solicitar cita"
    "Paciente[1]" -> "Solicitar cita"
}
```



SVG

```
<svg xmlns="http://www.w3.org/2000/svg" width="320" height="120" viewBox="0 0 320 120">
  <style>.actor{fill:#fff;stroke:#000;stroke-width:1.5}.uc{fill:#f2f2f2;stroke:#0366d6;stroke-width:2}.label{font-family:Arial;font-size:14px;fill:#000;text-anchor:middle}</style>
  <circle cx="30" cy="60" r="10" class="actor"/><line x1="30" y1="70" x2="30" y2="90" class="actor"/><line x1="22" y1="76" x2="38" y2="76" class="actor"/><line x1="30" y1="90" x2="22" y2="100" class="actor"/><line x1="30" y1="90" x2="38" y2="100" class="actor"/><text x="30" y="118" class="label">Paciente [1]</text>
  <ellipse cx="180" cy="60" rx="55" ry="20" class="uc"/><text x="180" y="65" class="label">Solicitar cita</text><line x1="40" y1="60" x2="125" y2="60" stroke="#000" stroke-width="1.5"/>
</svg>
```



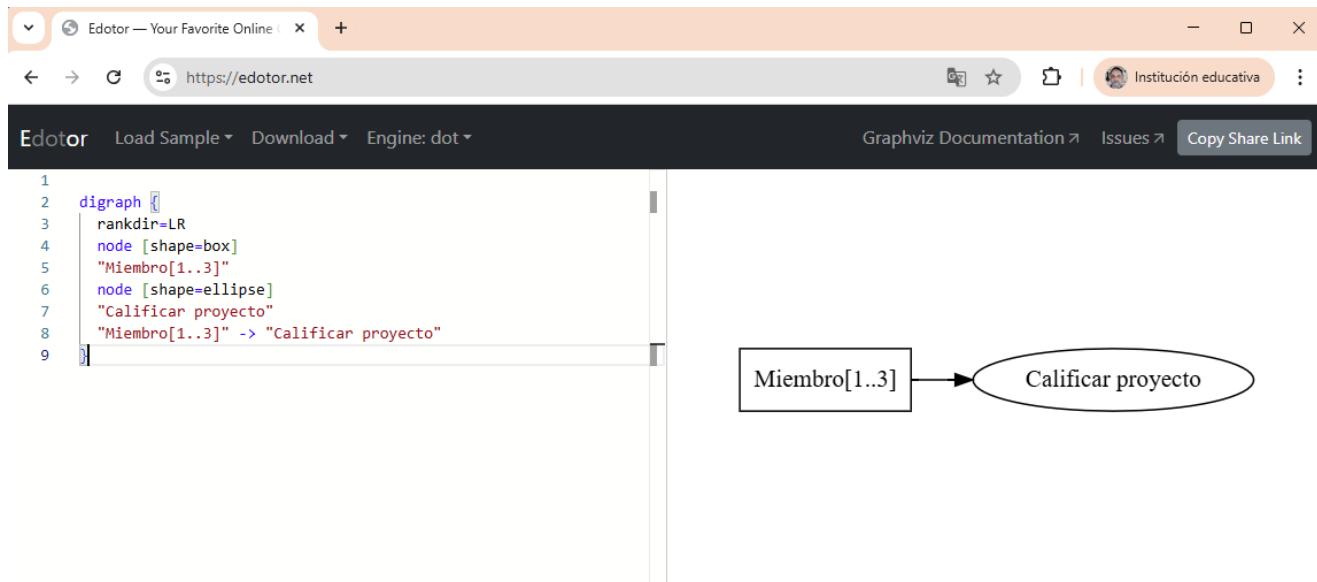
Ejemplo 3 - Sistema de votación electrónica

Regla: De 1 a 3 miembros del jurado deben calificar un proyecto.

- **Multiplicidad:** [1..3]

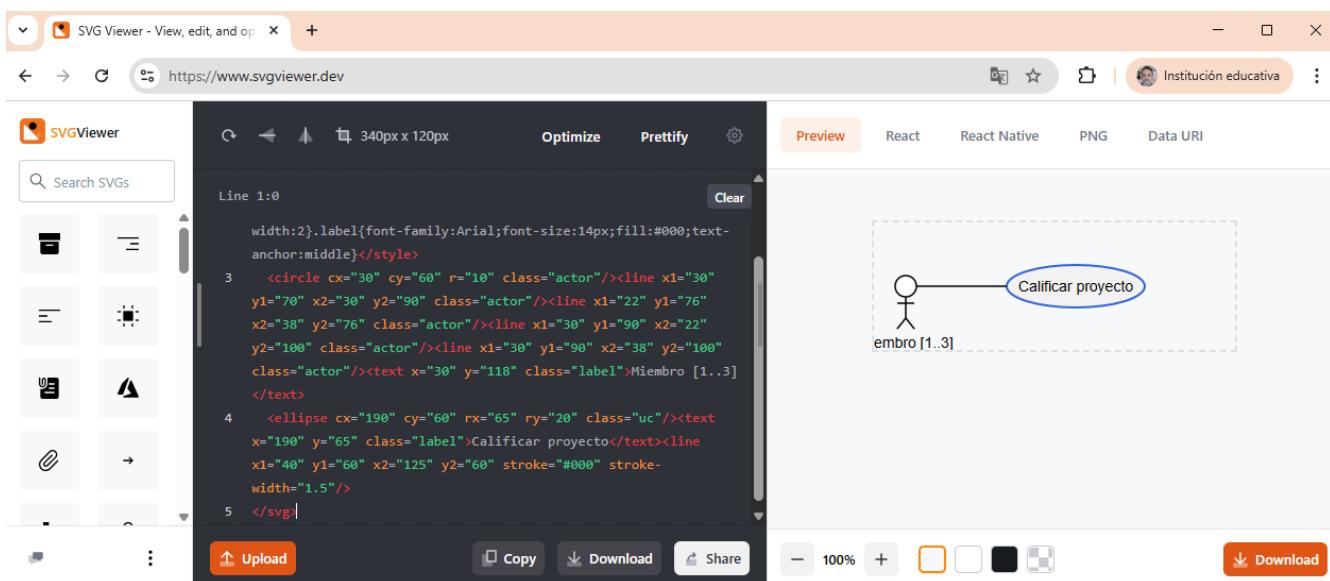
dot

```
digraph {
    rankdir=LR
    node [shape=box]
    "Miembro[1..3]"
    node [shape=ellipse]
    "Calificar proyecto"
    "Miembro[1..3]" -> "Calificar proyecto"
}
```



SVG

```
<svg xmlns="http://www.w3.org/2000/svg" width="340" height="120" viewBox="0 0 340 120">
  <style>.actor{fill:#fff;stroke:#000;stroke-width:1.5}.uc{fill:#f2f2f2;stroke:#0366d6;stroke-width:2}.label{font-family:Arial;font-size:14px;fill:#000;text-anchor:middle}</style>
  <circle cx="30" cy="60" r="10" class="actor"/><line x1="30" y1="70" x2="30" y2="90" class="actor"/><line x1="22" y1="76" x2="38" y2="76" class="actor"/><line x1="30" y1="90" x2="22" y2="100" class="actor"/><line x1="30" y1="90" x2="38" y2="100" class="actor"/><text x="30" y="118" class="label">Miembro [1..3]</text>
  <ellipse cx="190" cy="60" rx="65" ry="20" class="uc"/><text x="190" y="65" class="label">Calificar proyecto</text><line x1="40" y1="60" x2="125" y2="60" stroke="#000" stroke-width="1.5"/>
</svg>
```



Conclusión

La multiplicidad **acota o amplía** la participación de los actores y se **expresa entre corchetes** junto al nombre del rol, tanto en modelos lógicos (.dot) como en su visualización (SVG).

2.21 ¿El concepto de multiplicidad es expresado en UML ordinario en casos de uso?

En **UML “ordinario”** (especificación oficial de la OMG) la **multiplicidad NO es obligatoria** en los diagramas de casos de uso y, de hecho, **rara vez se incluye** en la práctica diaria:

1. El estándar sólo exige el **nombre del actor**; los corchetes $[0..1]$, $[1..*$], etc., **no forman parte del símbolo** estándar del “palito”.
2. Cuando se necesita indicar *cuántos* individuos participan se recurre a:
 - **Notas** (cuadro de texto adherido) o
 - **Requisitos adicionales** en la descripción textual del caso de uso.
3. Multiplicidad sí es canónica en diagramas de **clases, componentes o despliegue**, pero **no en casos de uso**.

Conclusión: puedes escribirla entre corchetes si aporta valor, pero **no es parte del UML “oficial”** para actores; se considera **documentación extra**.