Ejercicios UML

Juan de Lara

Grupo 46

Curso 2008/09

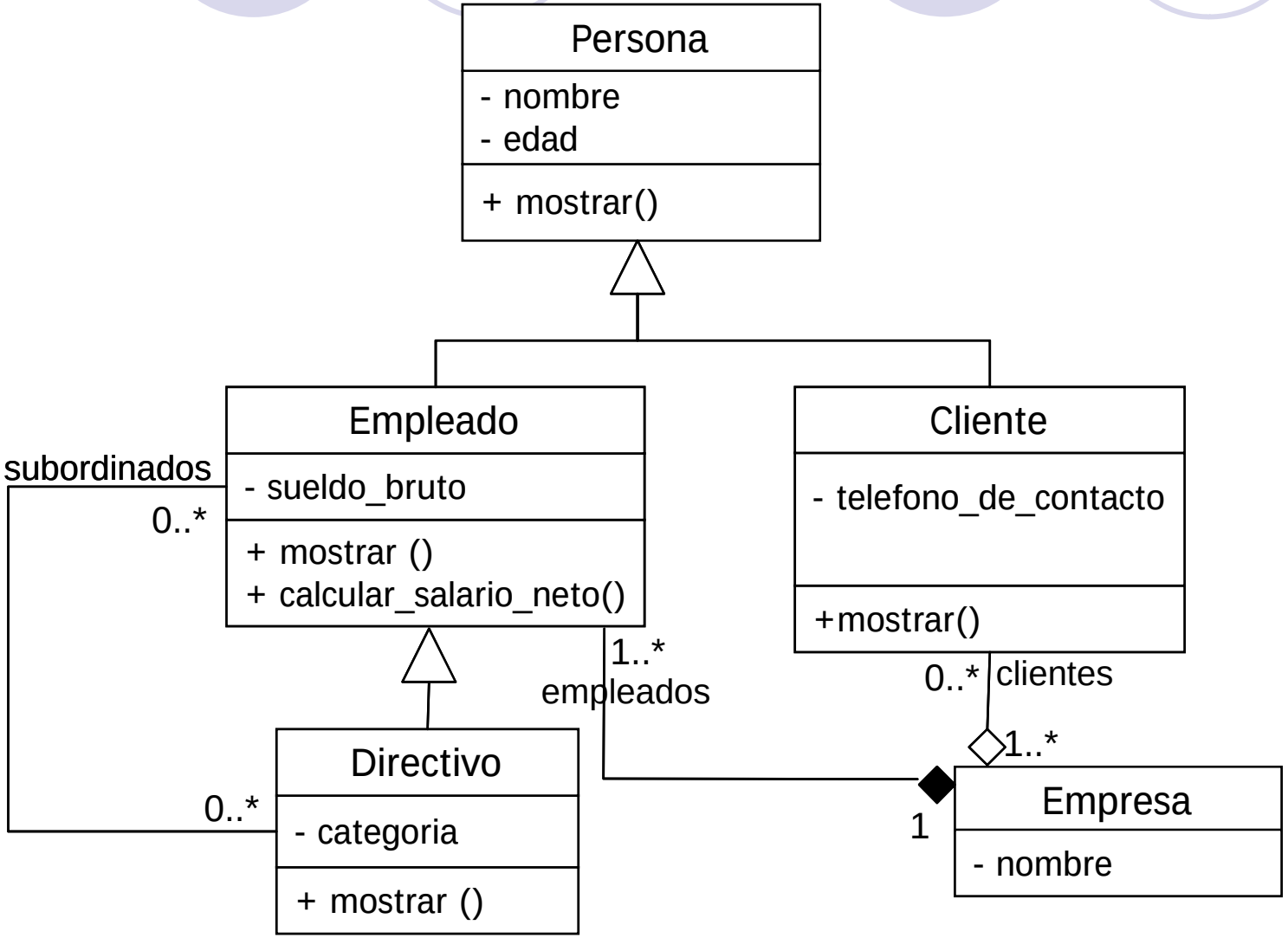
Indice

- **Diagramas de clases y OCL.**
- Diagramas de Transición de Estados
- Diagramas de Interacción.

Ejercicio

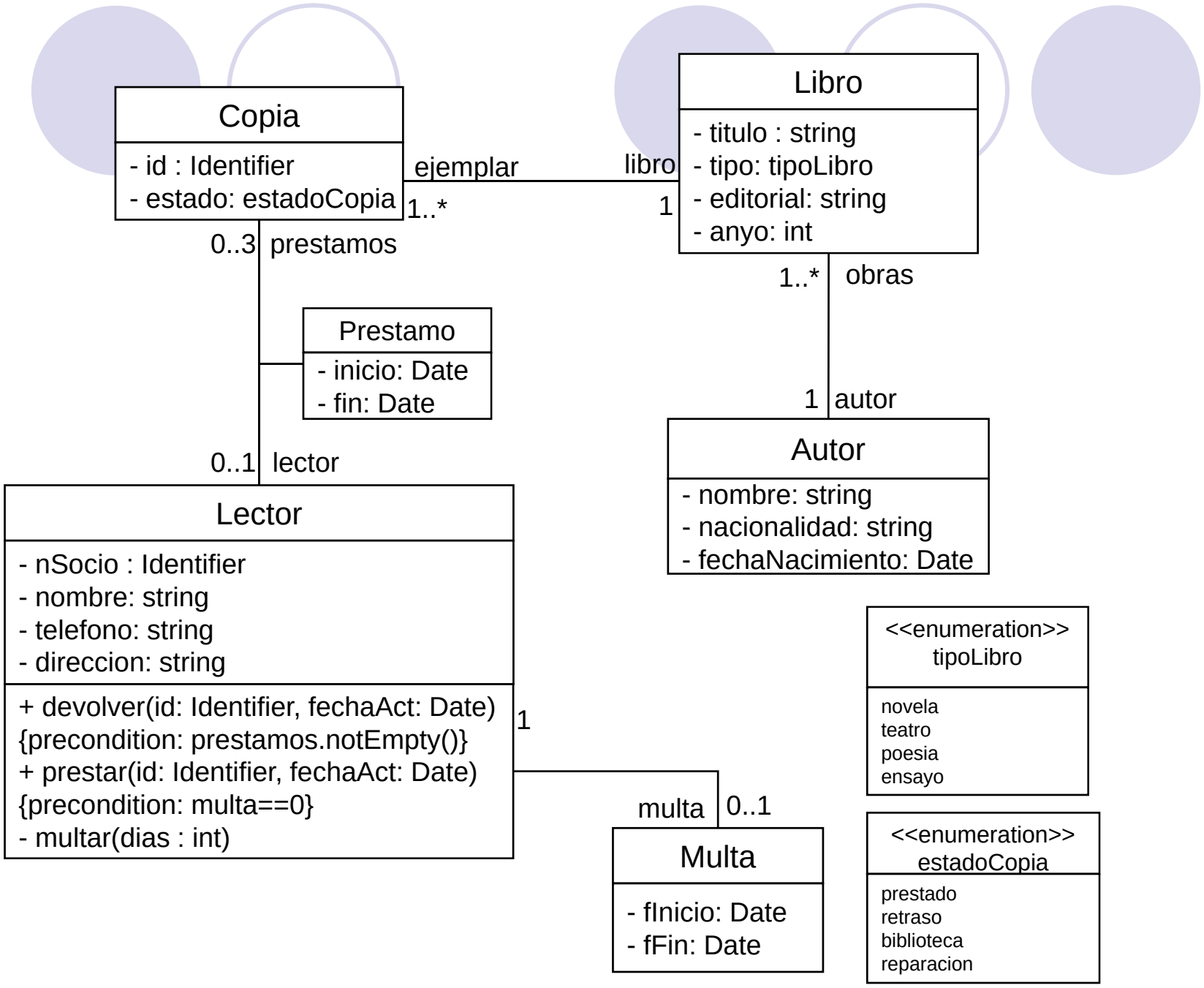
- Representa mediante un diagrama de clases la siguiente especificación:
  - Una aplicación necesita almacenar información sobre empresas, sus empleados y sus clientes.
  - Ambos se caracterizan por su nombre y edad.
  - Los empleados tienen un sueldo bruto, los empleados que son directivos tienen una categoría, así como un conjunto de empleados subordinados.
  - De los clientes además se necesita conocer su teléfono de contacto.
  - La aplicación necesita mostrar los datos de empleados y clientes.

Ejercicio


Persona
- nombre
- edad
+ mostrar()
Empleado
- sueldo_bruto
+ mostrar ()
+ calcular_salario_neto()
Cliente
- telefono_de_contacto
+mostrar()
subordinados
0..*
1..*
empleados
0..*
clientes
1..*
Directivo
0..*
- categoria
+ mostrar ()
1
Empresa
- nombre

Ejercicio: Biblioteca

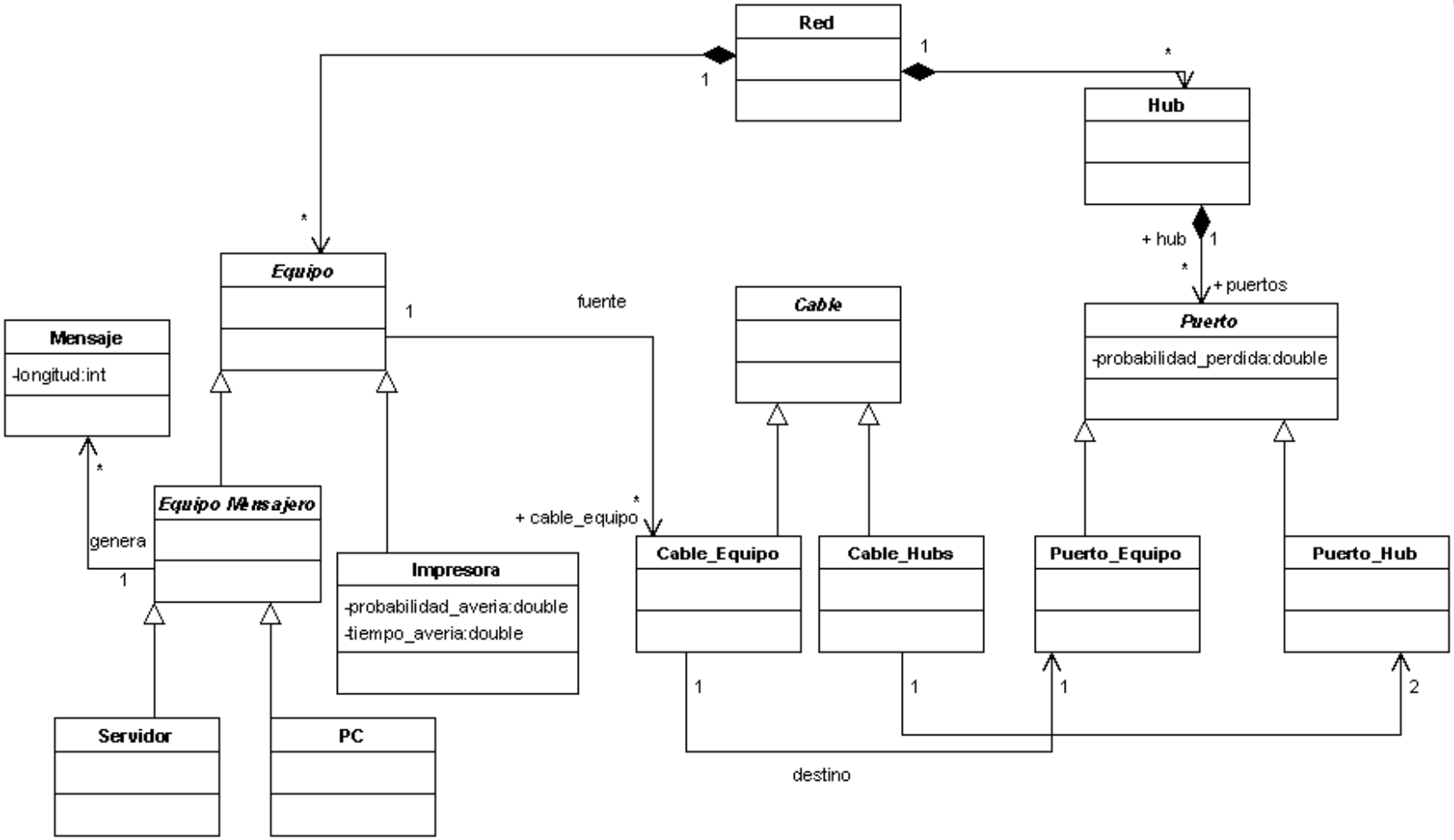
- Una biblioteca tiene copias de libros. Estos últimos se caracterizan por su nombre, tipo (novela, teatro, poesía, ensayo), editorial, año y autor.
- Los autores se caracterizan por su nombre, nacionalidad y fecha de nacimiento.
- Cada copia tiene un identificador, y puede estar en la biblioteca, prestada, con retraso o en reparación.
- Los lectores pueden tener un máximo de 3 libros en préstamo.
- Cada libro se presta un máximo de 30 días, por cada día de retraso, se impone una “multa” de dos días sin posibilidad de coger un nuevo libro.
- Realiza un diagrama de clases y añade los métodos necesarios para realizar el prestamo y devolución de libros.


Copia
- id : Identifier
- estado: estadoCopia
ejemplar
1..*
libro
1
Libro
- titulo : string
- tipo: tipoLibro
- editorial: string
- anyo: int
0..3
prestamos
1..*
obras
Prestamo
- inicio: Date
- fin: Date
1
autor
Autor
- nombre: string
- nacionalidad: string
- fechaNacimiento: Date
0..1
lector
Lector
- nSocio : Identifier
- nombre: string
- telefono: string
- direccion: string
+ devolver(id: Identifier, fechaAct: Date)
{precondition: prestamos.notEmpty()}
+ prestar(id: Identifier, fechaAct: Date)
{precondition: multa==0}
- multar(dias : int)
1
multa
0..1
Multa
- fInicio: Date
- fFin: Date
<<enumeration>>
tipoLibro
novela
teatro
poesia
ensayo
<<enumeration>>
estadoCopia
prestado
retraso
biblioteca
reparacion

Ejercicio

- Especificar un diagrama de clases que describa redes de ordenadores.
- Los elementos que se pueden incluir en la red son:
  - Servidor, PC, Impresora.
  - Hub, Cable de red.
- Los PCs pueden conectarse con un único Hub, los servidores con uno o varios.
- Los Servidores y PCs pueden generar mensajes, con una cierta longitud.
- Los Hubs tienen un número de puertos, algunos de los cuales puede usarse para conectar con otros Hubs. Tienen cierta probabilidad de “perder” mensajes.
- Las impresoras pueden averiarse, con cierta probabilidad, durante cierto tiempo.

Ejercicio. Posible Solución.



"Los PCs pueden conectarse con un único Hub, los servidores con uno o varios"
Podemos modelarlo como una restricción OCL, o bien añadir asociaciones desde Servidor y PC

OCL

“Los PCs pueden conectarse con un único Hub, los servidores con uno o varios”

```
Context PC
Inv: cable_equipo->size() = 1

Context Servidor
Inv: cable_equipo->size() >= 1
```

“Un Hub no puede conectarse consigo mismo a través de un puerto”

```
Context Cable_Hubs
Inv: Puerto_Hub.hub->asSet()->size() = 2
```

Ejercicio

Examen Junio 2008.

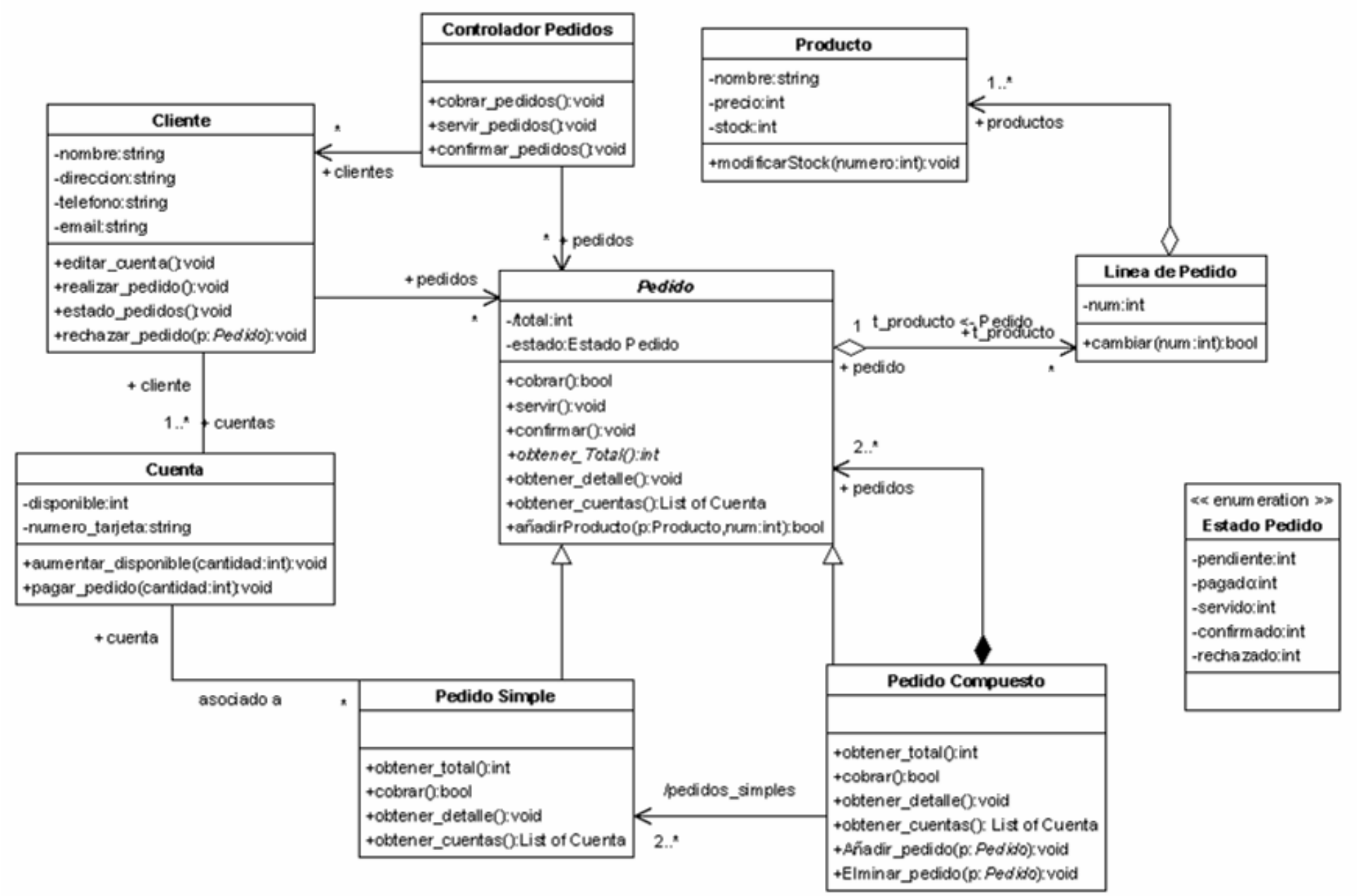
Realiza el diseño de una aplicación para la gestión de pedidos. La aplicación deberá manejar clientes (se guarda su nombre, dirección, teléfono y e-mail), que pueden realizar pedidos de productos, de los cuales se anota la cantidad en stock. Un cliente puede tener una o varias cuentas para el pago de los pedidos. Cada cuenta está asociada a una tarjeta de crédito, y tiene una cierta cantidad disponible de dinero, que el cliente debe aumentar periódicamente para poder realizar nuevos pedidos.

Un cliente puede empezar a realizar un pedido sólo si tiene alguna cuenta con dinero disponible. Al realizar un pedido, un cliente puede agruparlos en pedidos simples o compuestos. Los pedidos simples están asociados a una sola cuenta de pago y (por restricciones en la distribución) contienen un máximo de 20 unidades del mismo o distinto tipo de producto. A su vez, un pedido compuesto contiene dos o más pedidos, que pueden ser simples o compuestos. Como es de esperar, el sistema debe garantizar que todos los pedidos simples que componen un pedido compuesto se paguen con cuentas del mismo cliente. Además, sólo es posible realizar peticiones de productos en stock.

Existe una clase (de la cual debe haber una única instancia en la aplicación) responsable del cobro, orden de distribución y confirmación de los pedidos. El cobro de los pedidos se hace una vez al día, y el proceso consiste en comprobar todos los pedidos pendientes de cobro, y cobrarlos de la cuenta de pago correspondiente. Si una cuenta no tiene suficiente dinero, el pedido se rechaza (si es parte de un pedido compuesto, se rechaza el pedido entero). Una vez que el pedido está listo para servirse, se ordena su distribución, y una vez entregado, pasa a estar confirmado.

Se pide un diagrama de clases de diseño. Añade las restricciones OCL necesarias.

Solución


Controlador Pedidos
+cobrar_pedidos():void
+servir_pedidos():void
+confirmar_pedidos():void
Producto
-nombre:string
-precio:int
-stock:int
+modificarStock(numero:int):void
Cliente
-nombre:string
-direccion:string
-telefono:string
-email:string
+editar_cuenta():void
+realizar_pedido():void
+estado_pedidos():void
+rechazar_pedido(p:Pedido):void
*
+ clientes
* + pedidos
1..*
+ productos
+ pedidos
*
Pedido
-/total:int
-estado:Estado Pedido
+cobrar():bool
+servir():void
+confirmar():void
+obtener_Total():int
+obtener_detalle():void
+obtener_cuentas():List of Cuenta
+añadirProducto(p:Producto,num:int):bool
Linea de Pedido
-num:int
+cambiar(num:int):bool
1 t_producto <- Pedido
+t_producto
+ pedido
*
+ cliente
1..* + cuentas
Cuenta
-disponible:int
-numero_tarjeta:string
+aumentar_disponible(cantidad:int):void
+pagar_pedido(cantidad:int):void
2..*
+ pedidos
«« enumeration »»
Estado Pedido
-pendiente:int
-pagado:int
-servido:int
-confirmado:int
-rechazado:int
+ cuenta
asociado a *
Pedido Simple
+obtener_total():int
+cobrar():bool
+obtener_detalle():void
+obtener_cuentas():List of Cuenta
/pedidos_simples
2..*
Pedido Compuesto
+obtener_total():int
+cobrar():bool
+obtener_detalle():void
+obtener_cuentas(): List of Cuenta
+Añadir_pedido(p:Pedido):void
+Elminar_pedido(p:Pedido):void

Restricciones OCL:

```
Context Cliente::realizar_pedido:
pre: self.cuentas->exists(c | c.disponible > 0)

Context Pedido Compuesto:
inv: self.pedidos_simples->cuenta->cliente->asSet()->size() = 1

Context Pedido:
inv: self.t_productos.num->sum() <= 20

Context Pedido::añadirProducto(p: Producto, num: int):
pre: p.stock>=num

Context Cliente::rechazar_pedido (p:Pedido):
pre: self.cuentas.disponible->sum()<p.total
```

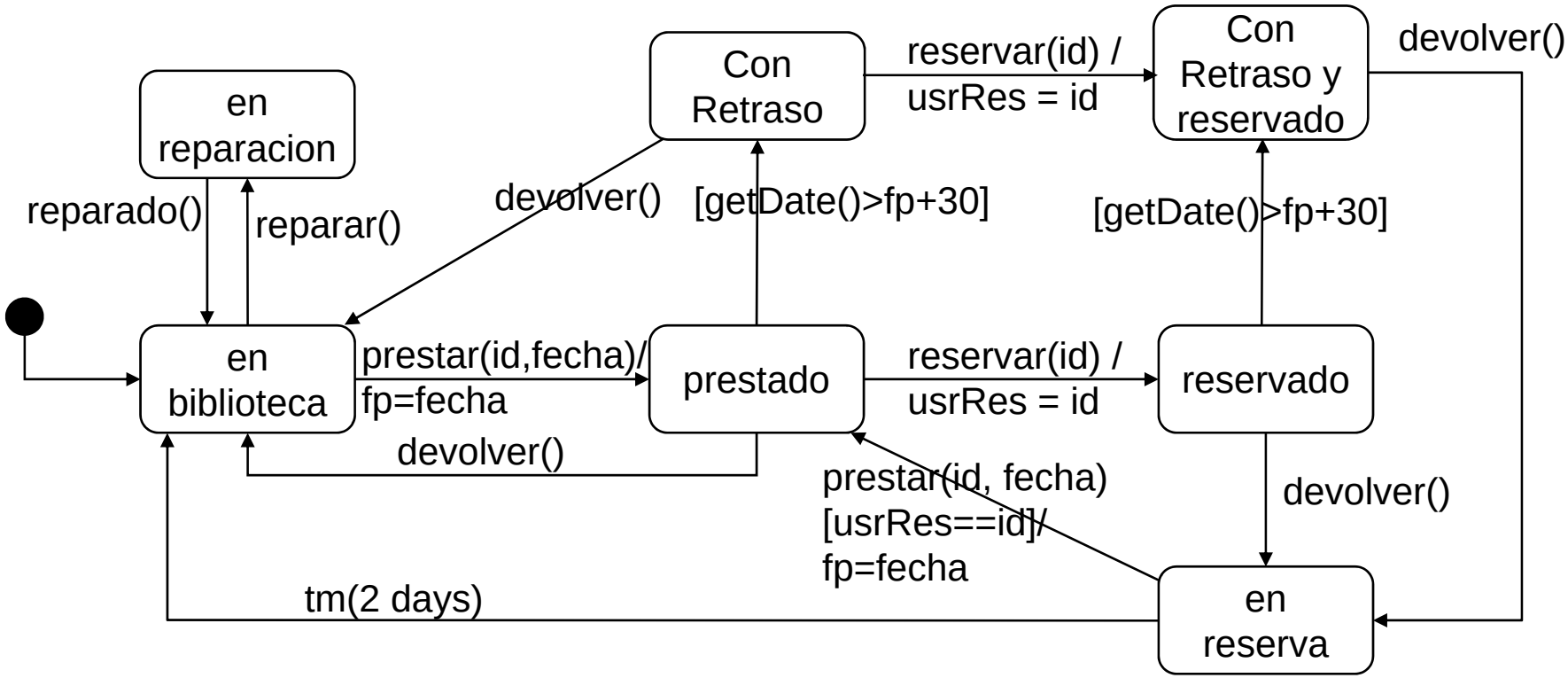
Indice

- Diagramas de clases
- **Diagramas de Transición de Estados**
- Diagramas de Interacción.

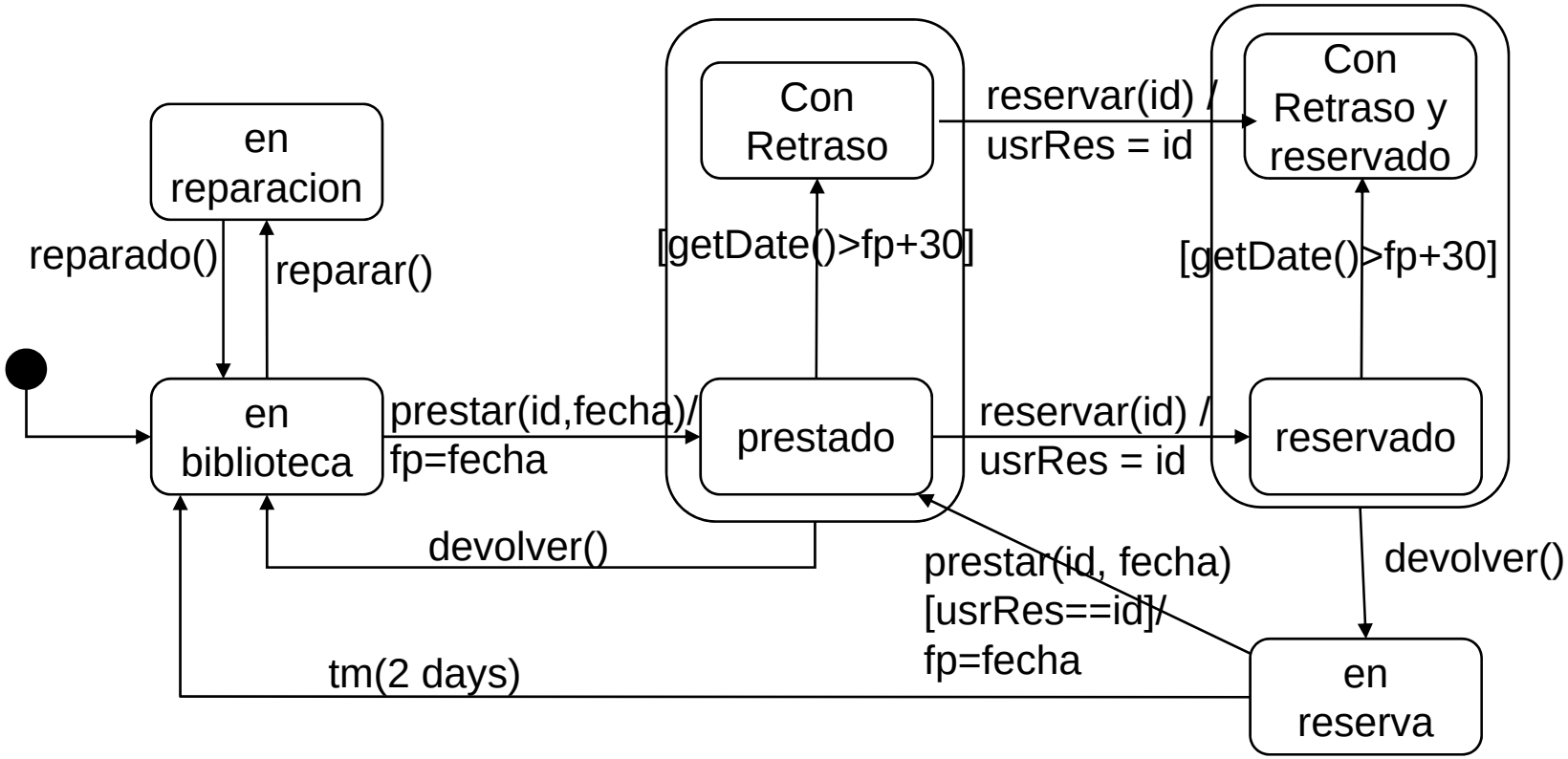
Ejercicio: Biblioteca

- Una biblioteca tiene copias de libros. Estos últimos se caracterizan por su nombre, tipo (novela, teatro, poesía, ensayo), editorial, año y autor.
- Los autores se caracterizan por su nombre, nacionalidad y fecha de nacimiento.
- Cada copia tiene un identificador, y puede estar en la biblioteca, prestada, reservada, con retraso o en reparación.
- Los lectores pueden tener un máximo de 3 libros en préstamo.
- Cada libro se presta un máximo de 30 días, por cada día de retraso, se impone una “multa” de dos días sin posibilidad de coger un libro.
- Realiza el diagrama de estados de la clase “copia”.

Solucion


en reparacion
reparado()
reparar()
Con Retraso
devolver()
[getDate()>fp+30]
reservar(id) / usrRes = id
Con Retraso y reservado
devolver()
[getDate()>fp+30]
en biblioteca
prestar(id,fecha)/ fp=fecha
devolver()
prestado
reservar(id) / usrRes = id
reservado
prestar(id, fecha) [usrRes==id]/ fp=fecha
devolver()
tm(2 days)
en reserva

Solucion: Estados Jerárquicos



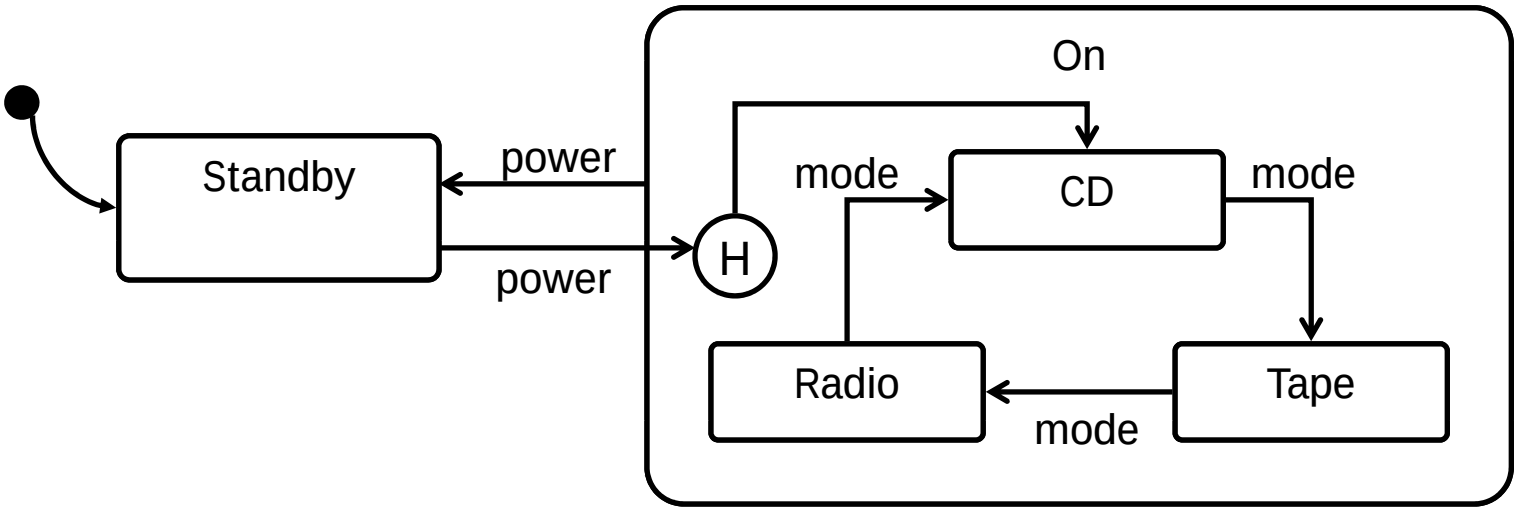
Máquinas de Estados

Estado Histórico. Ejercicio.

- Modelar el comportamiento de una cadena de música. Esta puede estar encendida (ON) o apagada (Standby). La cadena tiene reproductor de CD, Radio y Cinta. Se cambia de uno a otro con el botón “mode”. Cuando se enciende la cadena se recuerda el último estado en el que estuvo.

Máquinas de Estados

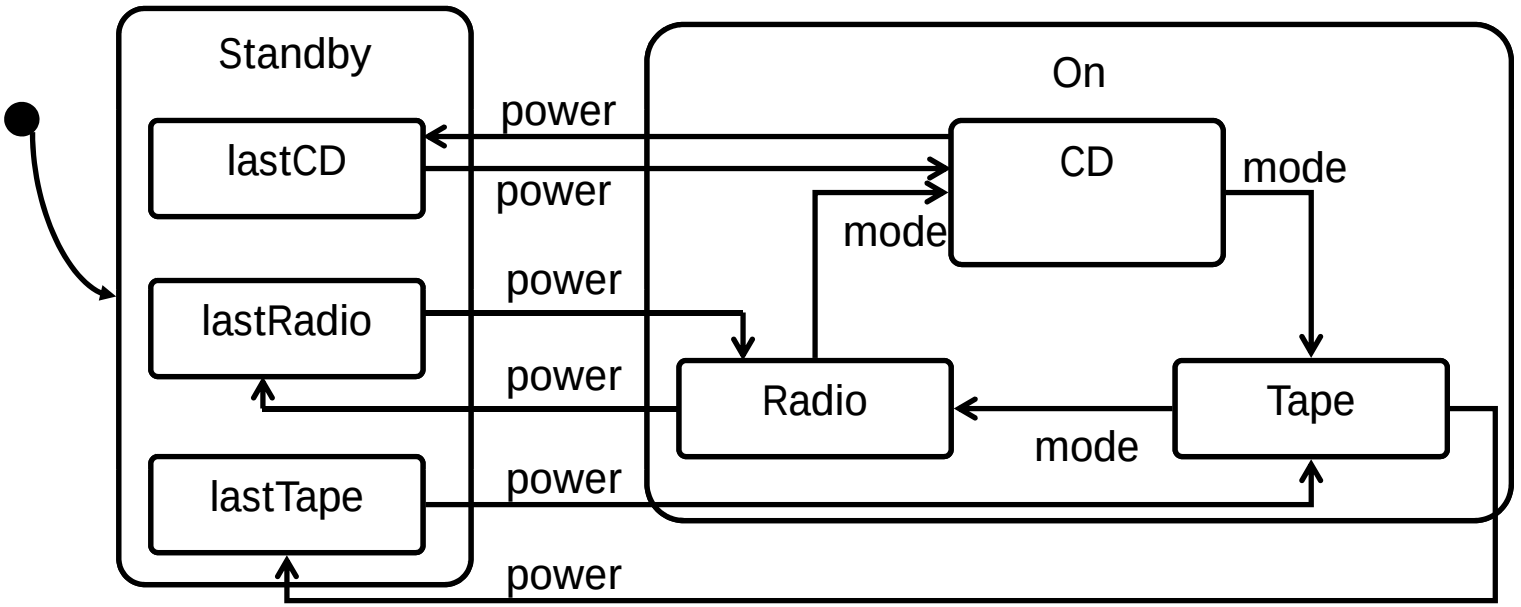
Estado Histórico. Ejercicio. Solución



Modelar el mismo sistema sin usar estado histórico.

Máquinas de Estados

Estado Histórico. Ejercicio. Solución (ii)



Ejercicio

Examen Junio 2008.

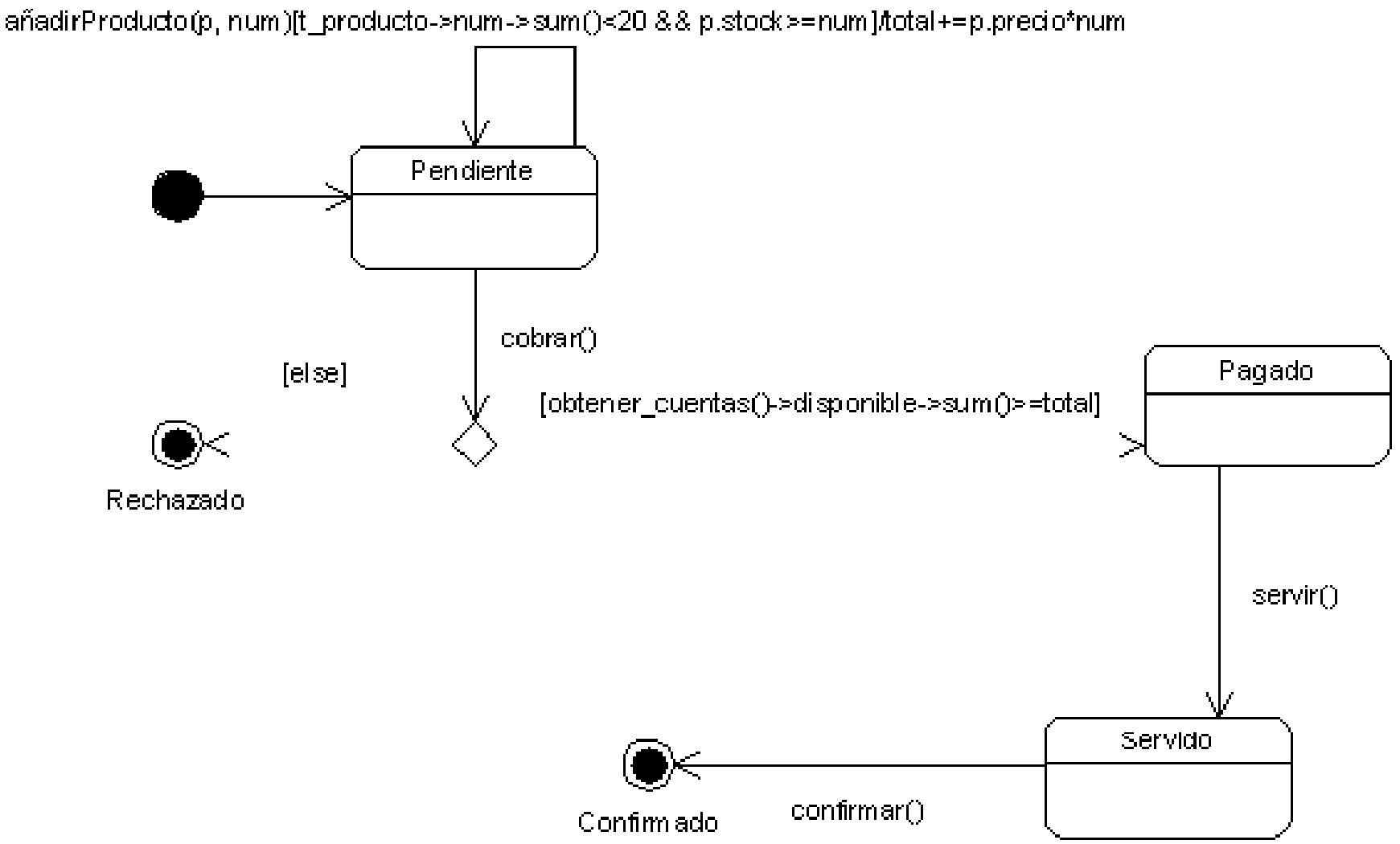
Realiza el diseño de una aplicación para la gestión de pedidos. La aplicación deberá manejar clientes (se guarda su nombre, dirección, teléfono y e-mail), que pueden realizar pedidos de productos, de los cuales se anota la cantidad en stock. Un cliente puede tener una o varias cuentas para el pago de los pedidos. Cada cuenta está asociada a una tarjeta de crédito, y tiene una cierta cantidad disponible de dinero, que el cliente debe aumentar periódicamente para poder realizar nuevos pedidos.

Un cliente puede empezar a realizar un pedido sólo si tiene alguna cuenta con dinero disponible. Al realizar un pedido, un cliente puede agruparlos en pedidos simples o compuestos. Los pedidos simples están asociados a una sola cuenta de pago y (por restricciones en la distribución) contienen un máximo de 20 unidades del mismo o distinto tipo de producto. A su vez, un pedido compuesto contiene dos o más pedidos, que pueden ser simples o compuestos. Como es de esperar, el sistema debe garantizar que todos los pedidos simples que componen un pedido compuesto se paguen con cuentas del mismo cliente. Además, sólo es posible realizar peticiones de productos en stock.

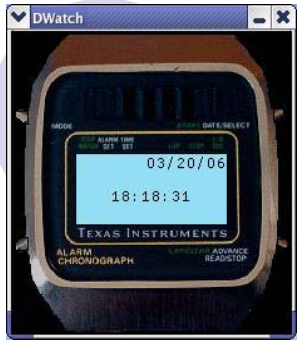
Existe una clase (de la cual debe haber una única instancia en la aplicación) responsable del cobro, orden de distribución y confirmación de los pedidos. El cobro de los pedidos se hace una vez al día, y el proceso consiste en comprobar todos los pedidos pendientes de cobro, y cobrarlos de la cuenta de pago correspondiente. Si una cuenta no tiene suficiente dinero, el pedido se rechaza (si es parte de un pedido compuesto, se rechaza el pedido entero). Una vez que el pedido está listo para servirse, se ordena su distribución, y una vez entregado, pasa a estar confirmado.

Se pide un diagrama de transición de estados para la clase Pedido

Solución


añadirProducto(p, num)[t_producto->num->sum()<20 && p.stock>=num]/total+=p.precio*num
Pendiente
cobrar()
[else]
[obtener_cuentas()->disponible->sum()>=total]
Pagado
Rechazado
servir()
Servido
Confirmado
confirmar()

Ejercicio

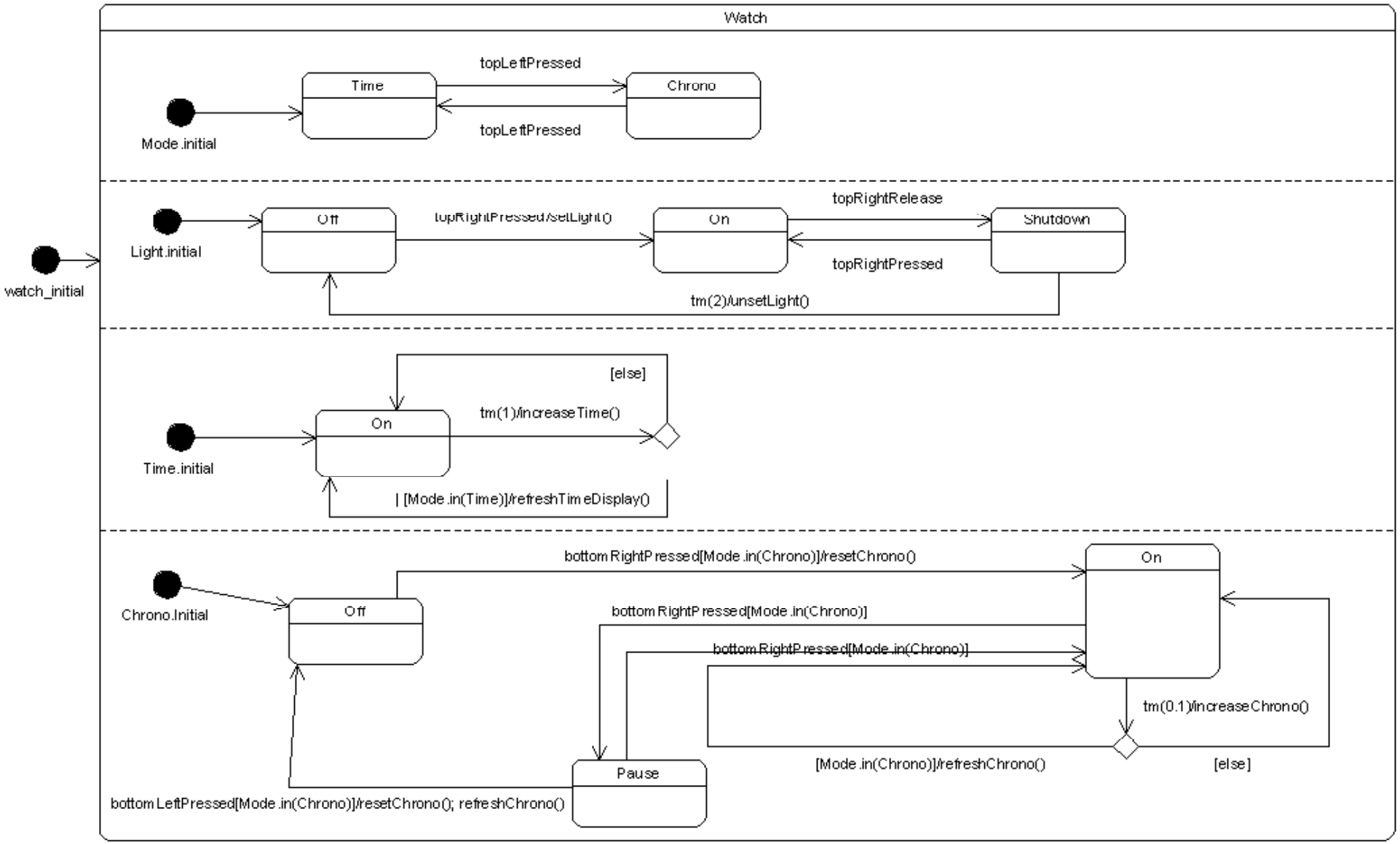


- Modelar el comportamiento reactivo de un reloj de pulsera.
- El valor del tiempo se debe actualizar cada segundo, incluso cuando no se muestra (p.ej. crono encendido).
- El botón de la parte superior derecha enciende la luz, que se mantiene encendida tanto como el botón está apretado, una vez que se suelta, la luz está encendida durante 2 segundos más y se apaga.
- El botón superior izquierdo alterna entre el modo de crono y de reloj. El sistema empieza en el modo reloj, en el que se muestra la hora en formato HH:MM:SS.
- En el modo crono, el tiempo discurrido se muestra en formato MM:SS:CC (CC son centésimas de segundo). Inicialmente el crono empieza en 00:00:00. El botón inferior derecho se usa para activar el crono. Éste se actualiza en incrementos de 1/100 segundos. Presionando el botón inferior derecho pausa o continua el crono (si el reloj está en modo crono). Pulsando el botón inferior izquierdo resetea el crono a 00:00:00 si el reloj está en modo crono y el crono ha sido pausado antes. El crono continua corriendo (si está corriendo) o mantiene su valor (si está en pausa) incluso cuando el reloj está en un modo de display distinto (por ejemplo, cuando se muestra la hora).

Ejercicio

- Interface provisto por el controlador:
  - ***getTime()*** : Devuelve la hora actual.
  - ***refreshTimeDisplay()*** : Repinta la hora en el visor con la hora interna actual. El visor no necesita limpiarse antes de llamar a esta función. Por ejemplo, si se está visualizando el crono, se borrará antes de pintar la hora.
  - ***refreshChronoDisplay()*** : ver ***refreshTimeDisplay().***
  - ***resetChrono()*** : Resetea el crono interno a 00:00:00.
  - ***increaseTime()*** : Incrementa la hora en un segundo. Los minutos y horas se modificarán adecuademente, (por ejemplo, si se llama a ***increaseTime ()*** a las 11:59:59, la nueva hora será 12:00:00).
  - ***increaseChrono ()*** : Incrementa el crono en 1/100 segundos.
  - ***setLight()*** : Enciende la luz del visor.
  - ***unsetLight()*** : Apaga la luz del visor.
- Eventos de botones recibidos:
  - **topRightPressed.**
  - **topRightReleased.**
  - **topLeftPressed.**
  - **topLeftReleased.**
  - **bottomRightPressed.**
  - **bottomRightReleased.**
  - **bottomLeftPressed.**
  - **bottomRightReleased.**

Posible Solución.



Indice

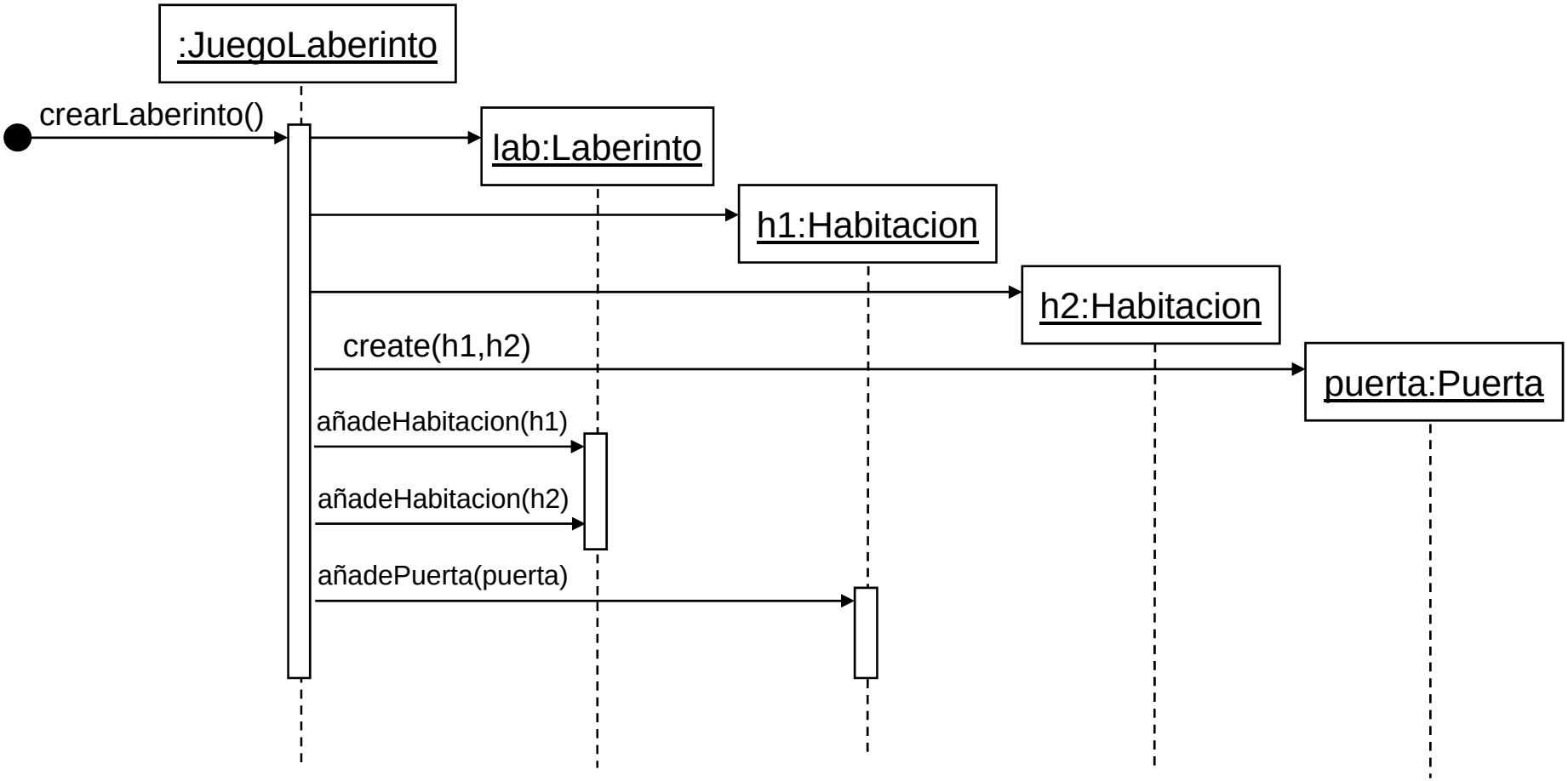
- Diagramas de clases
- Diagramas de Transición de Estados
- **Diagramas de Interacción.**

Ejercicio

Especificar el diagrama de secuencia de la operación “crearLaberinto”

```
public class JuegoLaberinto {
   public Laberinto crearLaberinto () {
      Laberinto lab = new Laberinto();
      Habitacion h1 = new Habitacion();
      Habitacion h2 = new Habitacion();
      Puerta puerta = new Puerta(h1, h2);
      lab.añadeHabitacion(h1);
      lab.añadeHabitacion(h2);
      h1.añadePuerta(puerta);
      return lab;
   }
}
```

Solución

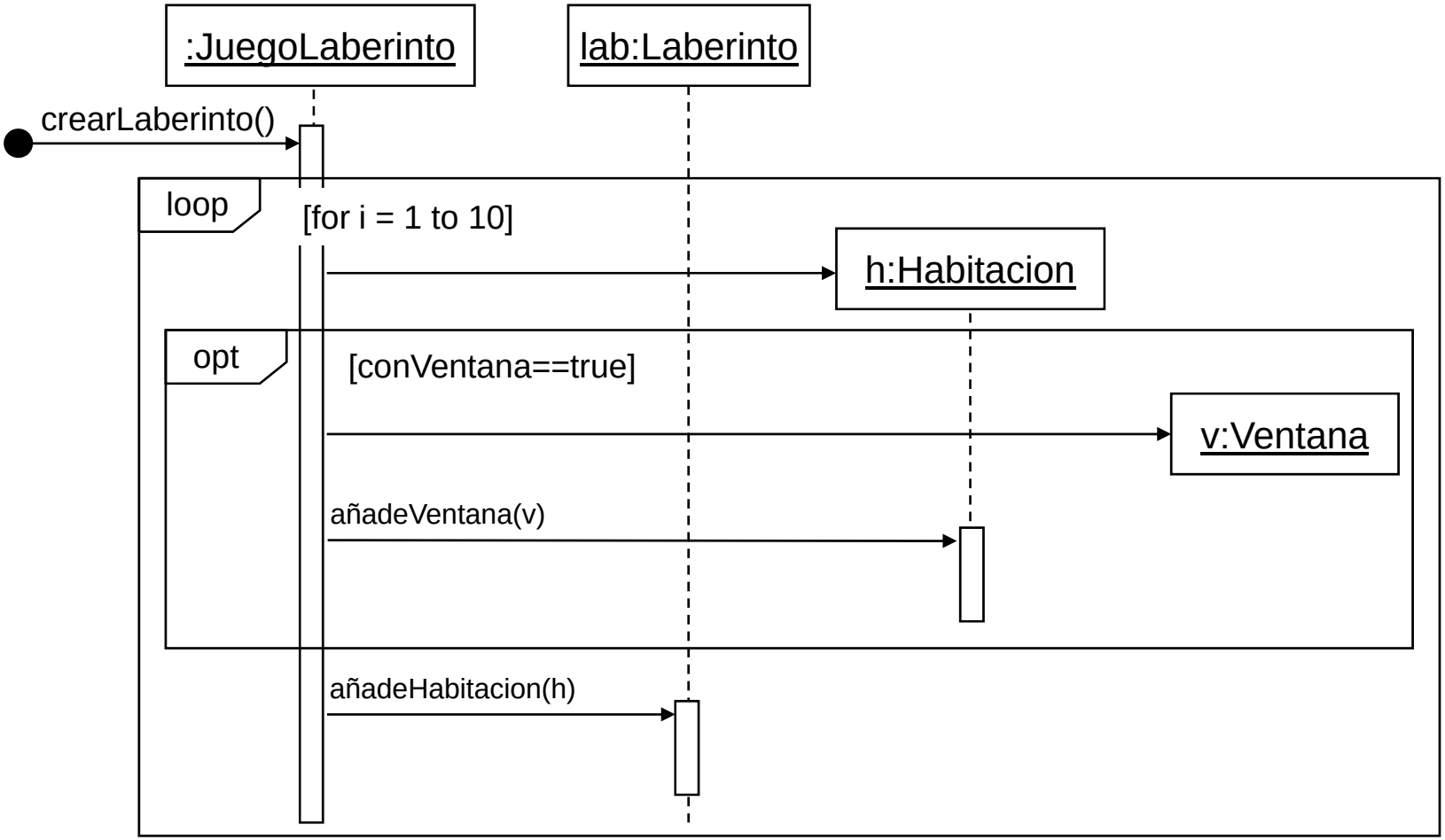

:JuegoLaberinto
crearLaberinto()
lab:Laberinto
h1:Habitacion
h2:Habitacion
create(h1,h2)
puerta:Puerta
añadeHabitacion(h1)
añadeHabitacion(h2)
añadePuerta(puerta)

Ejercicio

Especificar el diagrama de secuencia de la operación “crearLaberinto”

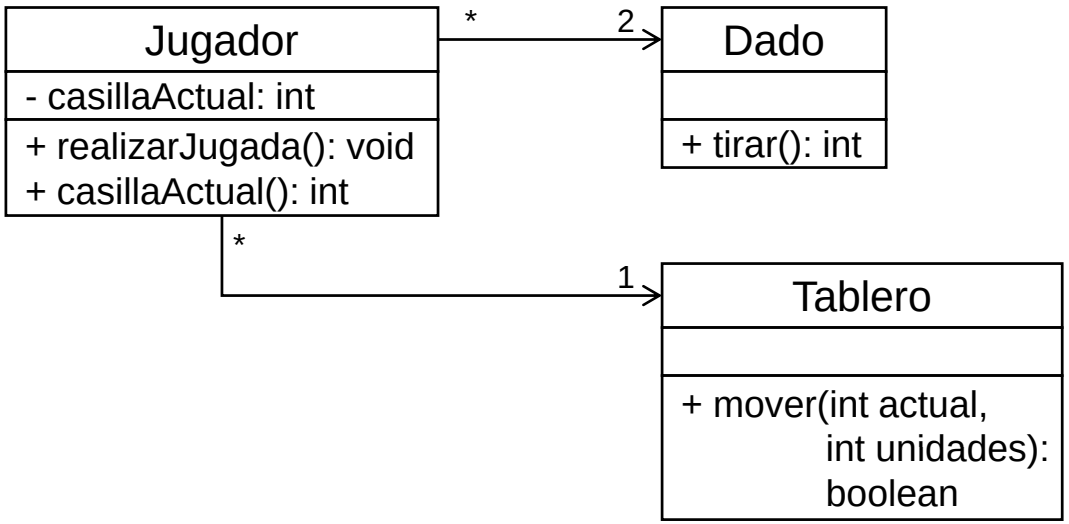
```
public class JuegoLaberinto {
   private Laberinto lab;
   private boolean conVentana;

   public JuegoLaberinto() {
      lab = new Laberinto();
      conVentana = true;
   }

   public void crearLaberinto () {
      Habitacion h;
      for (int i=0; i<10; i++) {
         h = new Habitacion();
         if (conVentana == true)
            h.añadeVentana(new Ventana());
         lab.añadeHabitacion(h);
      }
}
```

Solución

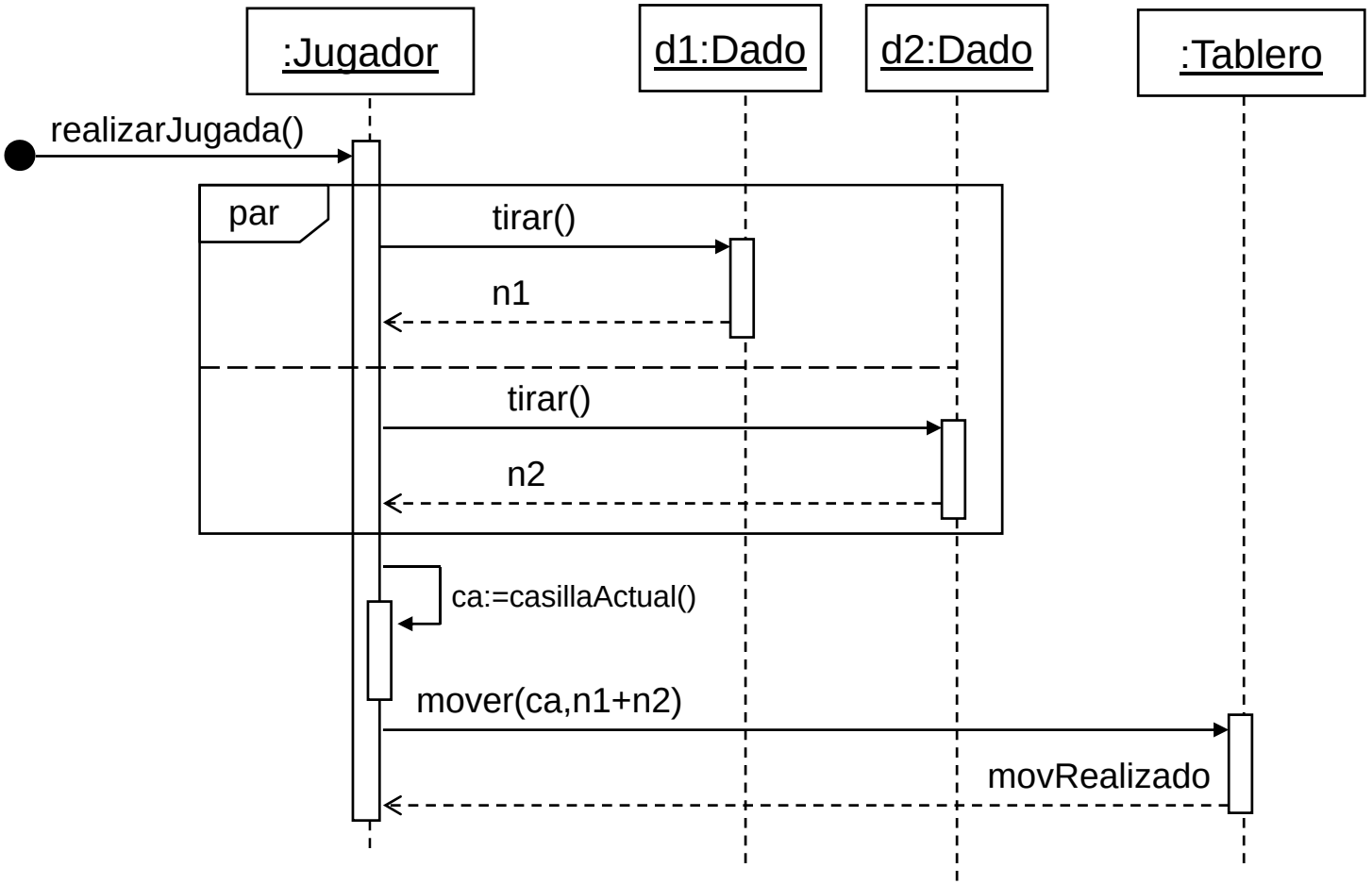

:JuegoLaberinto
lab:Laberinto
crearLaberinto()
loop
[for i = 1 to 10]
h:Habitacion
opt
[conVentana==true]
v:Ventana
añadeVentana(v)
añadeHabitacion(h)

Ejercicio

Especificar el diagrama de secuencia de la operación “realizarJugada” definida en la clase Jugador, para el juego del parchís



Solución

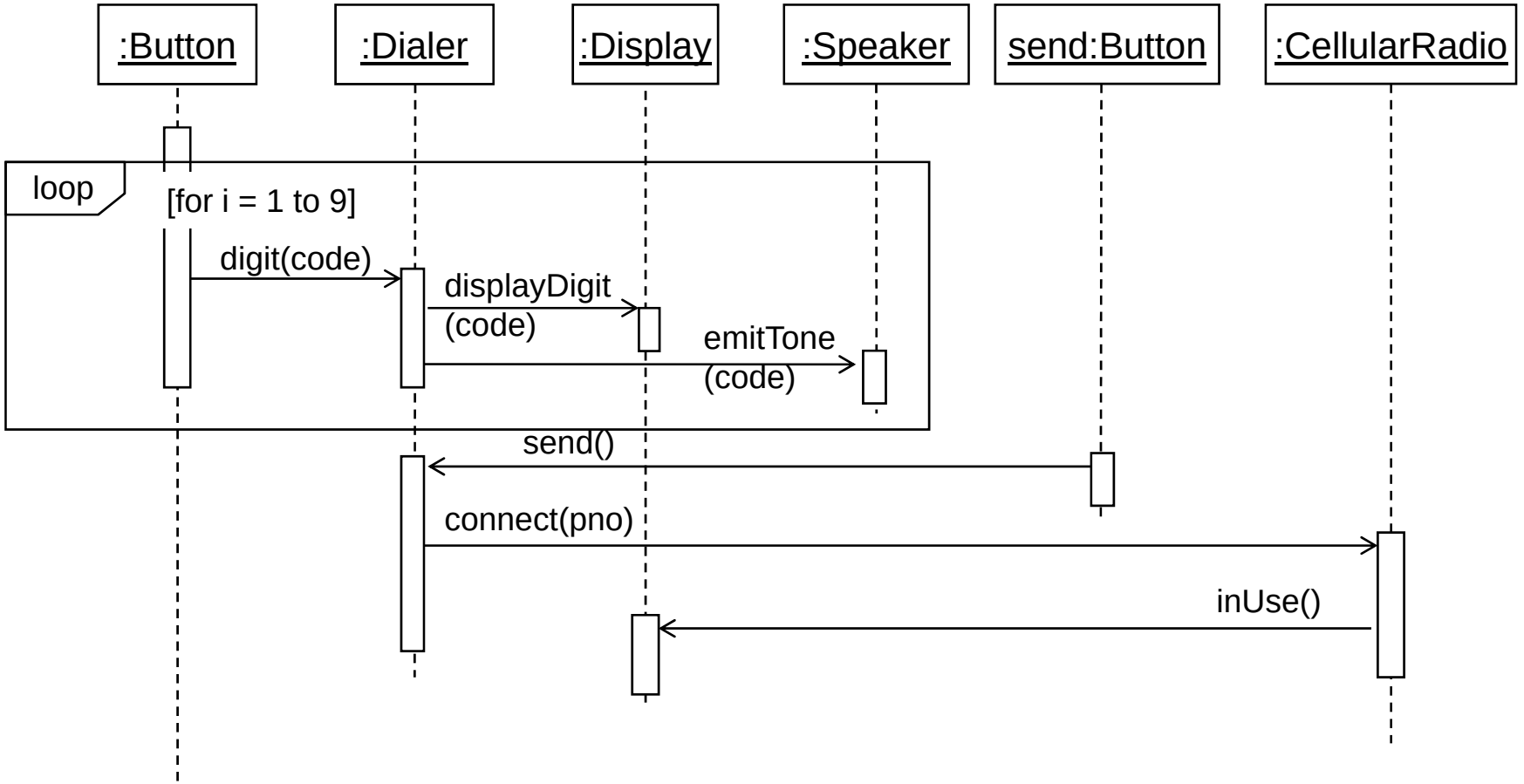

:Jugador
d1:Dado
d2:Dado
:Tablero
realizarJugada()
par
tirar()
n1
tirar()
n2
ca:=casillaActual()
mover(ca,n1+n2)
movRealizado

Ejercicio

Identificar las clases relevantes y realizar el diagrama de secuencia para el siguiente caso de uso, que corresponde a la realización de una llamada desde un teléfono móvil.

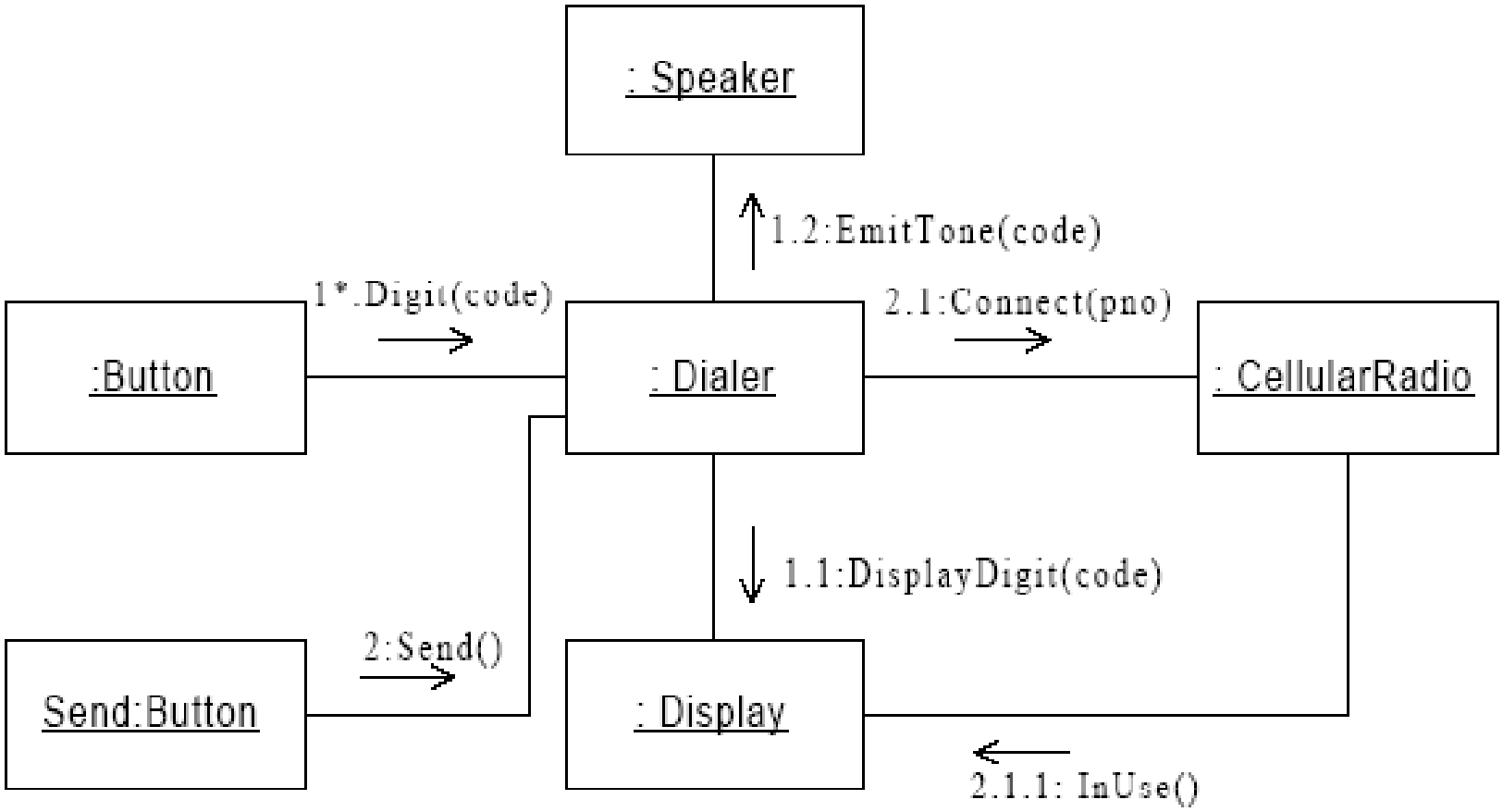
- El usuario pulsa los dígitos del número de teléfono
- Para cada dígito
  - la pantalla se actualiza para añadir el dígito marcado
  - se emite un tono por el receptor
- El usuario pulsa el botón “Enviar”
- El indicador “en uso” se ilumina en pantalla
- El móvil establece conexión con la red
- Los dígitos acumulados se mandan a la red
- Se establece la conexión con el número marcado

Solución



¿Diagrama de colaboración equivalente?

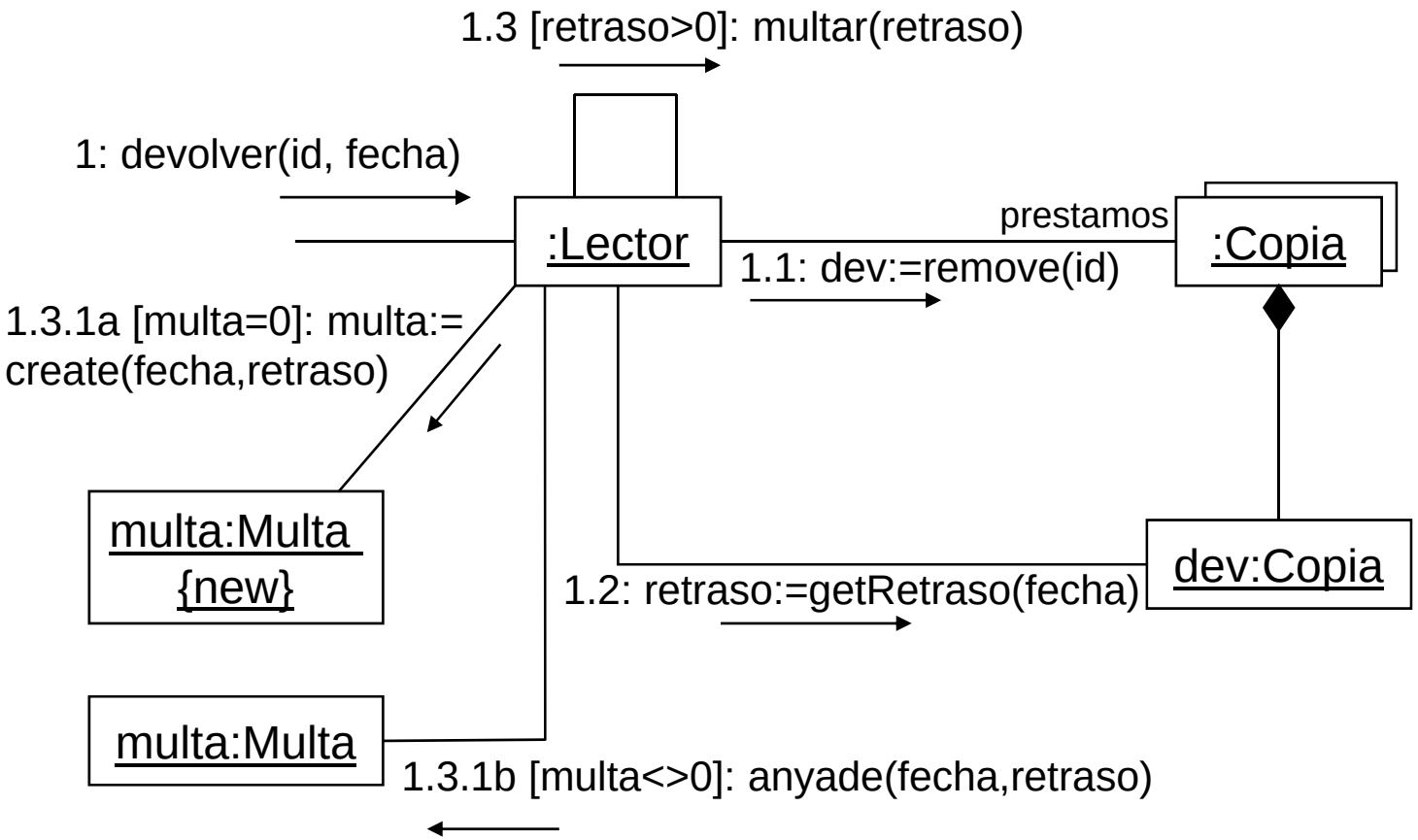
Solución


: Speaker
1.2:EmitTone(code)
1*.Digit(code)
2.1:Connect(pno)
:Button
: Dialer
: CellularRadio
1.1:DisplayDigit(code)
2:Send()
Send:Button
: Display
2.1.1: InUse()

Ejercicio: Biblioteca

- Una biblioteca tiene copias de libros. Estos últimos se caracterizan por su nombre, tipo (novela, teatro, poesía, ensayo), editorial, año y autor.
- Los autores se caracterizan por su nombre, nacionalidad y fecha de nacimiento.
- Cada copia tiene un identificador, y puede estar en la biblioteca, prestada, reservada, con retraso o en reparación.
- Los lectores pueden tener un máximo de 3 libros en préstamo.
- Cada libro se presta un máximo de 30 días, por cada día de retraso, se impone una “multa” de dos días sin posibilidad de coger un libro.
- Realiza el diagrama de colaboración para el método devolver()

Solucion


1.3 [retraso>0]: multar(retraso)
1: devolver(id, fecha)
:Lector
prestamos
:Copia
1.1: dev:=remove(id)
1.3.1a [multa=0]: multa:=
create(fecha,retraso)
multa:Multa
{new}
dev:Copia
1.2: retraso:=getRetraso(fecha)
multa:Multa
1.3.1b [multa<>0]: anyade(fecha,retraso)