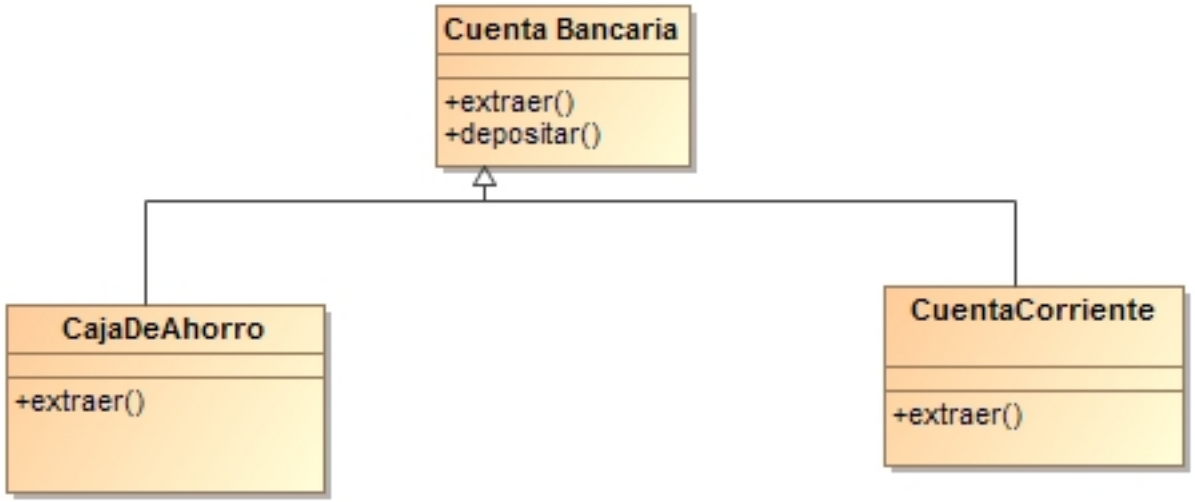
Problema

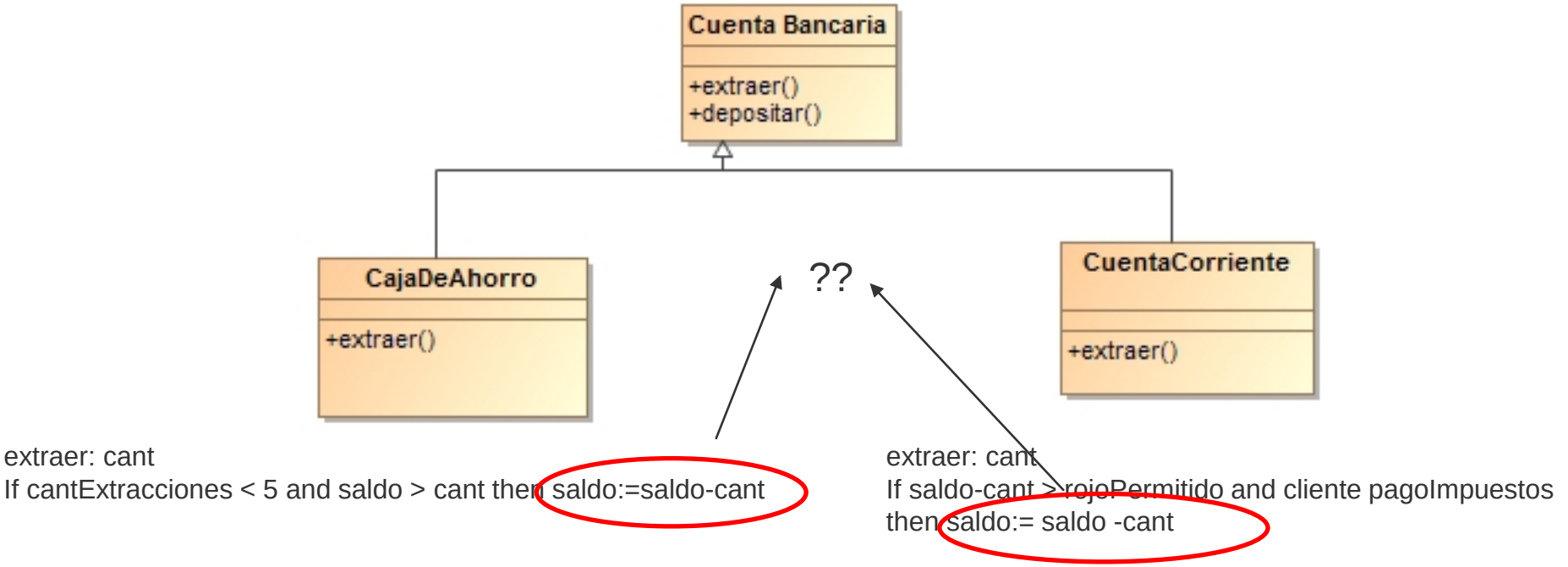
- ✓ Supongamos una jerarquia de cuentas bancarias y una operación con “variantes” de acuerdo a la cuenta



El Metodo Extraer

- ✓ En la Clase Caja de Ahorro tiene que controlar el saldo contra 0 y la cantidad de extracciones
- ✓ En la Clase Cuenta Corriente tiene que controlar el saldo contra un “rojo” permitido y la situacion impositiva del cliente
- ✓ Otras sub-clases podrían implementar reglas de extracción diferentes

Solucion



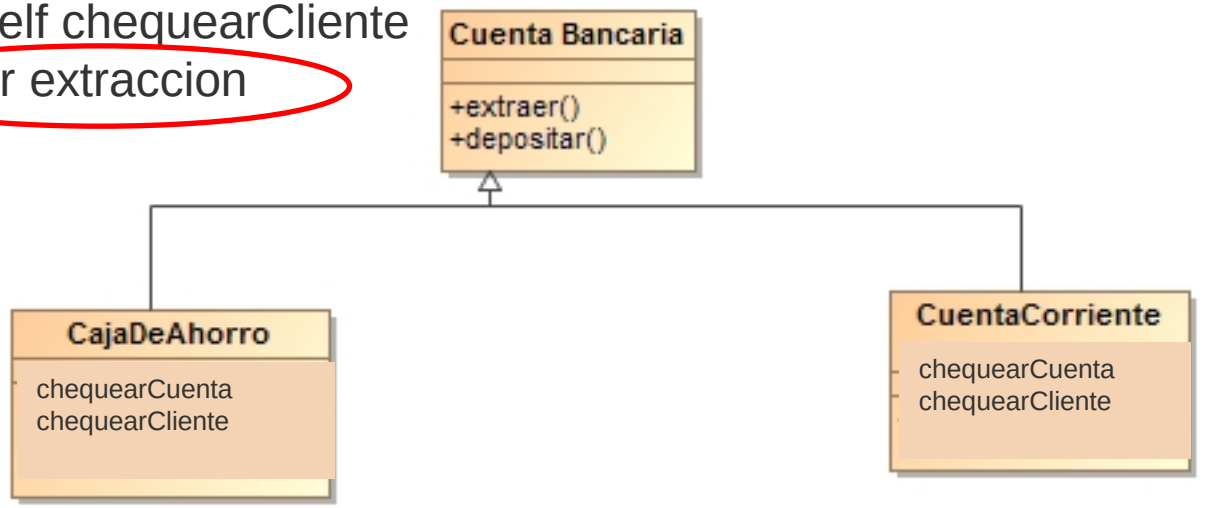
Problemas con esta Solucion

Que pasa cuando aparecen reglas “nuevas” que se aplican a diferentes tipos de cuentas? O chequeos sobre las personas?

Template Method

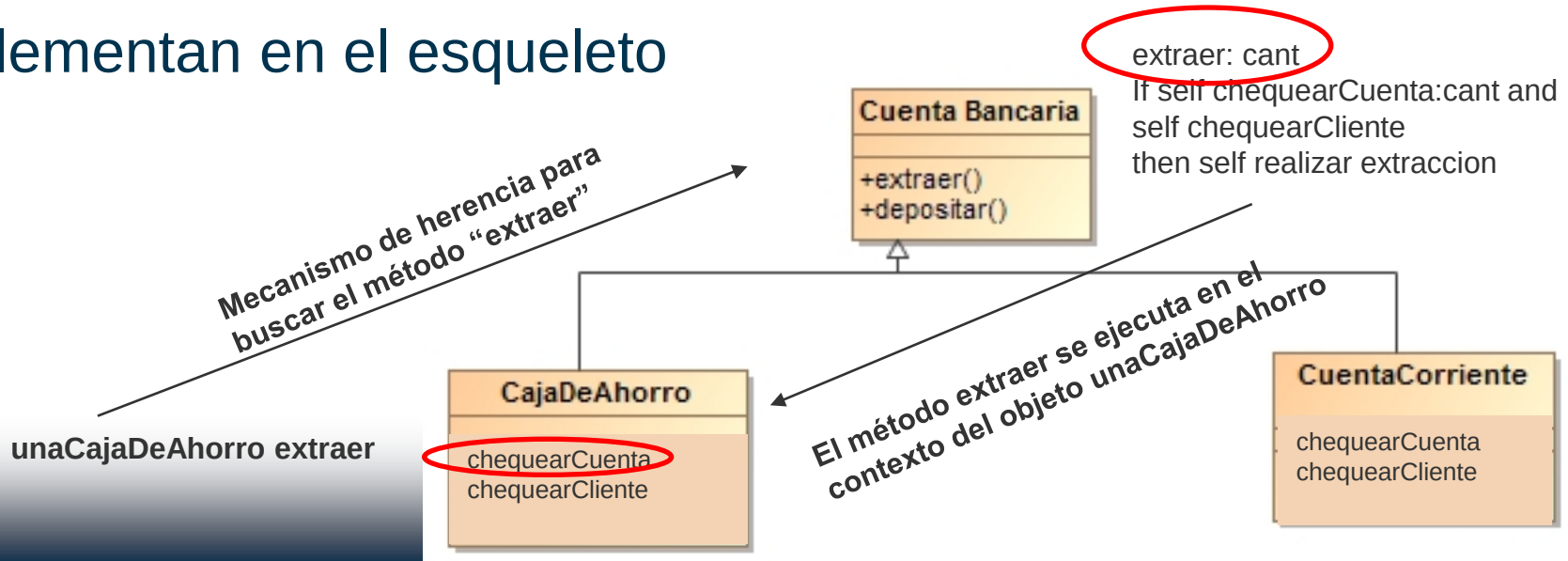
- ✓ Re-escribimos el algoritmo en términos más genéricos
- ✓ Lo escribimos con forma de esqueleto en la clase abstracta
- ✓ Las (sub) clases concretas implementan las diferencias como métodos

```
extraer: cant
If self chequearCuenta:cant and self chequearCliente
                then self realizar extraccion
```



Como funciona esta solución?

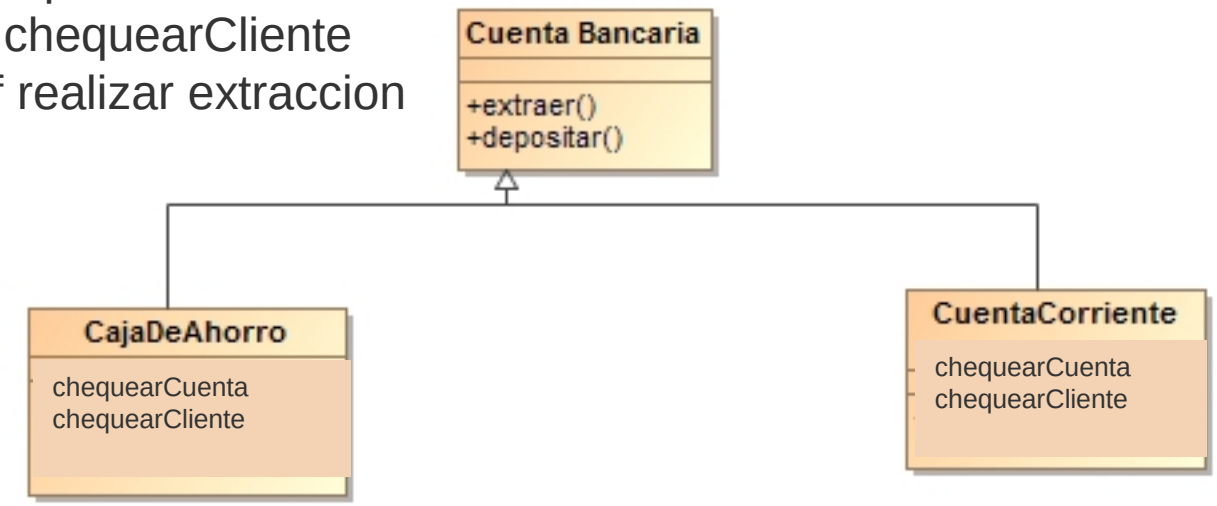
- ✓ Cuando un objeto CajaDeAhorro (o CuentaCorriente) recibe el mensaje *extraer,* como no lo tiene definido usa el método de la super-clase (CuentaBancaria)
- ✓ La ejecución de ese método ocurre en el contexto del objeto recepto (CajaDeAhorro), entonces la expresión *self chequearCuenta* provoca que el mensaje se envía a ese objeto, el cual responde ejecutando su propio método
- ✓ Observen que es “como si” desde la super-clase se usaran métodos de la sub, aunque no es así. El código esta en la super pero se ejecuta en la sub
- ✓ Las nuevas reglas se implementan en el esqueleto



Acerca del código extraer

- ✓ Ojo: Es solo un pseudocódigo
- ✓ “chequearCuenta” y “chequearCliente” devuelven true o false, si la condición de extracción se cumple

```
extraer: cant
If self chequearCuenta:cant
and self chequearCliente
then self realizar extraccion
```




```
chequearCuenta:cant
^ (saldo >cant)

chequearCliente
^true
```

```
chequearCuenta:cant
^ (saldo+rojo >cant)

chequearCliente
^cliente pagoImpuestos
```

- ✓ chequearCliente podría estar implementado con un default en CuentaBancaria

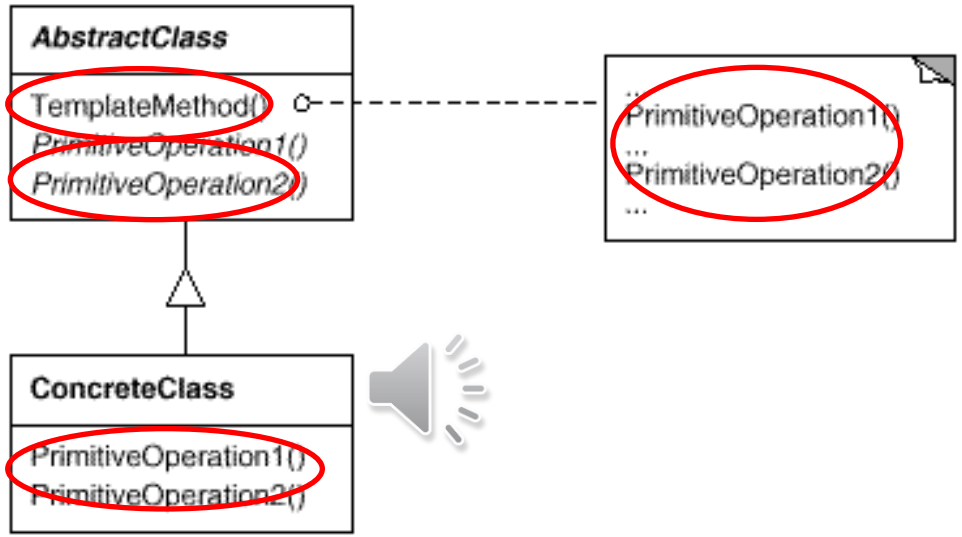
Template Method

- **Intent:**
  - Definir el esqueleto de un algoritmo en un metodo, difiriendo algunos pasos a las subclases. El template method permite que las subclases re-definan ciertos aspectos de un algoritmo sin cambiar su estructura

- **Aplicabilidad**
  - Para implementar las partes invariantes de un algoritmo una vez y dejar que las sub-clases implementen los aspectos que varian

Template Method

- Structure



- Las operaciones primitivas también se denominan métodos hook.
- Observen que yo puedo implementar una operación primitiva en una sub-clase la cual será invocada por un código pre-existente: el template method