Diseño de Aplicaciones II

Diseño de Frameworks

Qué es un Framework ?

- Definiciones:
  - Es una aplicación reusable y semicompleta que se puede especializar para producir aplicaciones especificas
  - Es un conjunto de clases que comprenden un diseño abstracto para soluciones de una familia de problemas relacionados
  - Es un esqueleto de aplicación adaptable por el desarrollador

Clasificación por alcance

- **De infraestructura**
  - Servicios de bajo nivel como comunicaciones y acceso a APIS del sistema operativo
- **“Middleware” de integración**
  - Permiten integrar componentes y sistemas
- **De aplicación en dominios particulares**
  - Seguros, finanzas, control industrial, etc.

Clasificación por diseño

- **Caja Blanca (White Box)**
  - Son extensibles mediante la utilización de herencia y polimorfismo (Template Method)
  - El usuario del framework define subclases y redefine determinados métodos
  - Ventajas
    - Son flexibles y adaptables
  - Desventajas
    - Requieren que el usuario del framework tenga cierto conocimiento de la implementación del framework

Clasificación por diseño

- **Caja Negra (Black Box)**
  - Definen un conjunto de interfaces que permiten asociar al framework componentes intercambiables que deben respetar determinada interfaz
  - Ventajas:
    - Fáciles de utilizar
  - Desventajas:
    - Difíciles de implementar

Principales beneficios Frameworks

- Reducir el tiempo y costo de desarrollo
- Disminuir el numero de defectos en las aplicaciones
- Capturar el conocimiento de un dominio
- Contar con diseños reusables

Principales problemas Frameworks

- **Difíciles de entender**
  - Problema de documentación para usuarios
  - Necesidad de conocer los mecanismos internos del framework
- **Difíciles de desarrollar**
  - Difíciles de evolucionar ya que los cambios tienen mucho impacto en las aplicaciones que utilizan el framework
- **Difíciles de combinar**

Frameworks vs. Bibliotecas de Clases

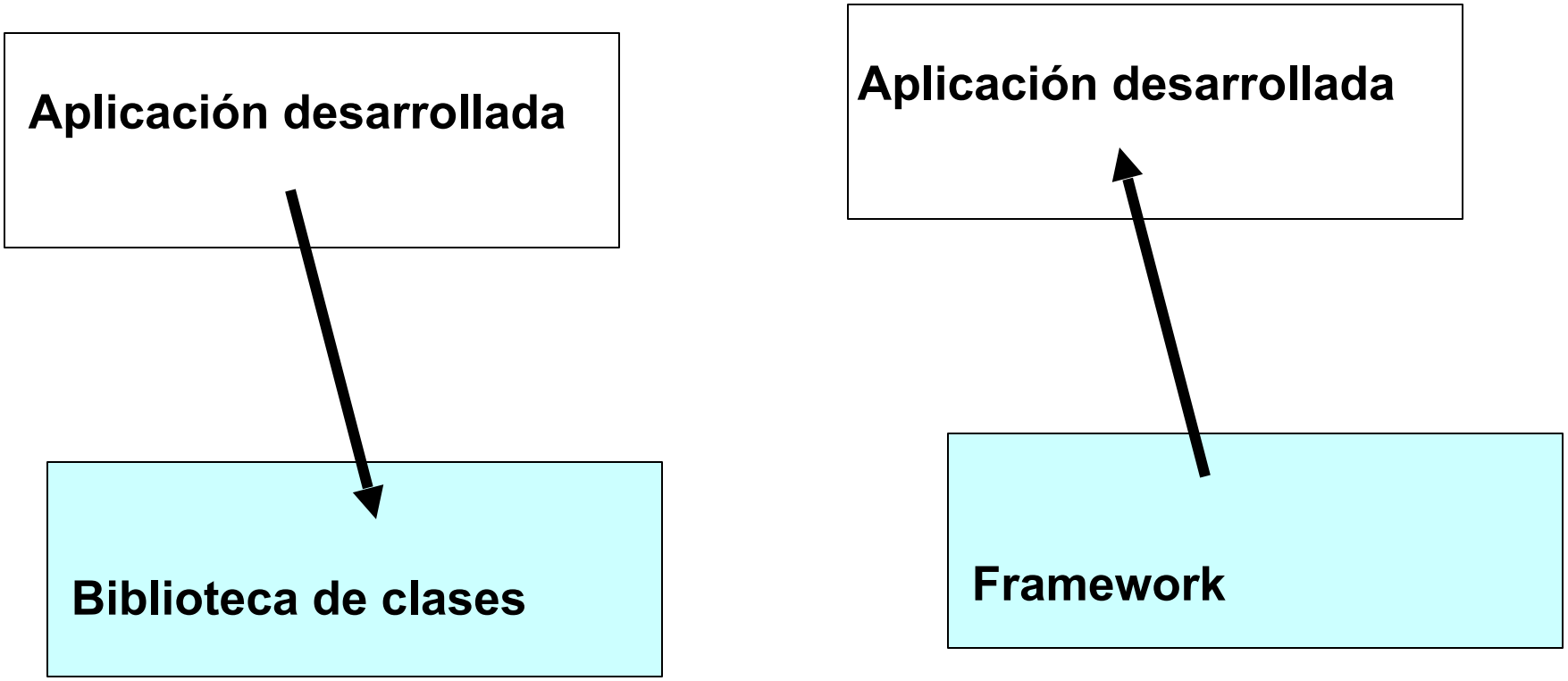
- Las bibliotecas de clases son independientes del dominio, mientras que los frameworks son desarrollados para determinados dominios o problemas particulares

Frameworks vs. Bibliotecas de Clases

- El uso de una biblioteca de clases se caracteriza por que el flujo de control en un solo sentido (de la aplicación a la biblioteca)

- En la utilización de un Framework el flujo de control es principalmente en el sentido Framework - Aplicación (Inversión de control)

Frameworks vs. Bibliotecas de Clases



Frameworks

- Normalmente un framework tiene asociado:
  - Una o más bibliotecas de clases que proveen componentes utilizados por el framework
  - Utilitarios como generadores de código, debuggers que permiten automatizar la generación de código particular utilizado por el framework
    - Ej. Classwizard, resource editor

Frameworks y Patterns

- Un **framework** involucra un diseño y su implementación (son programas)
- Los **patterns** representan problemas abstractos y recurrentes de diseño junto a su solución (el código asociado al pattern simplemente ejemplifica)
- En el diseño de un framework se utilizan patterns para diseñar y documentar el framework

Principales elementos utilizados en el diseño y desarrollo de Frameworks

- **Conceptos:**
  - Herencia
  - Polimorfismo
  - Clases Abstractas
  - Composición y delegación

Principales elementos utilizados en el diseño y desarrollo de Frameworks

- **Análisis, diseño y método:**
  - Patterns
  - Análisis de dominio
  - Refactoring
  - Proceso de desarrollo para frameworks
  - Documentación de frameworks
  - Uso de frameworks

Clases Abstractas

- Normalmente una clase abstracta difiere la implementación de algunos de sus métodos a sus subclases (abstract, virtual)
- Generalmente una clase abstracta surge de la generalización de clases concretas (Generalización)
- Una vez creada una clase abstracta nuevas clases concretas se pueden derivar de ella (Especialización)
- El concepto de **clase abstracta es central** en la implementación de frameworks orientados a objetos

Polimorfismo

- “Es un concepto de la teoría de tipos en el cual un nombre (variable) puede representar objetos de diferentes clases mientras estas estén relacionadas por una superclase en común. Cualquier objeto representado por esta variable es capaz de responder a un conjunto común de operaciones en forma diferente” [Booch]
- En el contexto de la Orientación a Objetos significa que el generador de un mensaje no necesita conocer la clase a la que pertenece el receptor, el cual puede pertenecer a cualquier clase

Herencia vs. Composición

- **Herencia**
  - Quiebra la encapsulación (reuso de caja blanca)
  - Definición estática
  - Fácil de usar y de cambiar la implementación en forma estática
- **Composición**
  - Permite reuso de caja negra
  - Definición en tiempo de ejecución

Delegación

- Es la forma de implementar las asociaciones
- Consiste en que dos objetos colaboren para satisfacer un pedido
  - Un objeto recibe un pedido y lo **delega** a otro para que lo resuelva

Utilización de Patrones (Gamma)

- La utilización de patrones de diseño simplifica el desarrollo de un framework

Creación	Estructura	Comportamiento
Factory Method	Adapter	**Template Method**
Abstract Factory	**Bridge**	Chain of Responsability
Prototype	**Composite**	Command
Singleton	Decorator	Interpreter
Builder	**Facade**	**Iterator**
	Flyweight	**Mediator**
	Proxy	Memento
		Observer
		State
		Strategy
		Visitor

Desarrollo de Frameworks

- Hot Spot
  - Son aquellos aspectos de un dominio de aplicación que deben mantenerse flexibles (código que es **similar** - pero cambia - en todas las aplicaciones del dominio)
  - Determinan aquellos lugares donde el framework debe permitir adaptación
- Frozen Spots
  - Aspectos del framework que no se diseñaron para permitir adaptación

Desarrollo de Frameworks

- En la implementación de frameworks se distinguen los siguientes tipos de métodos:
- Métodos **Template**
  - Se basan en
- Métodos **Hook,** que pueden ser:
  - Métodos abstractos
  - Métodos comunes
  - Template methods

Desarrollo de Frameworks

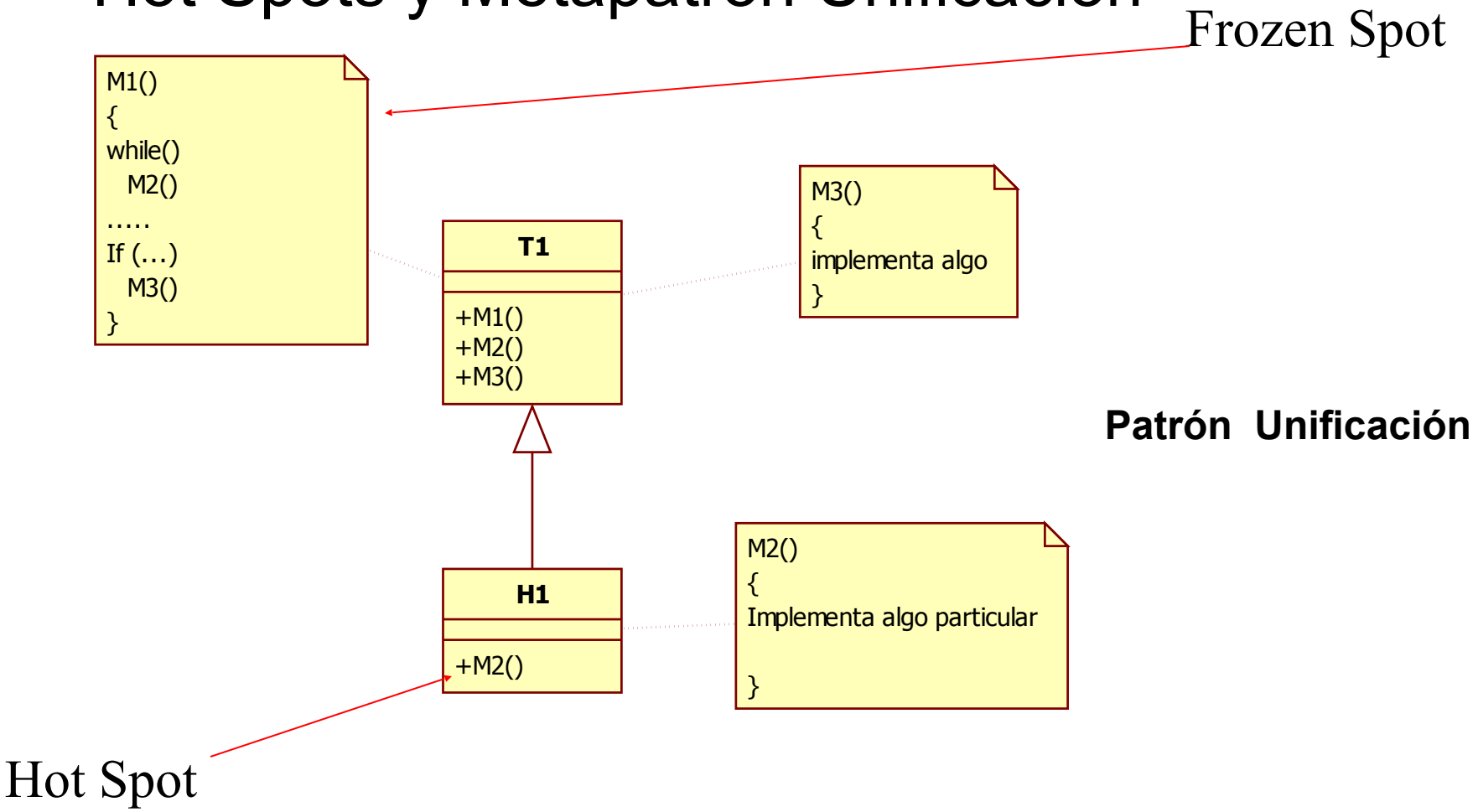
- Métodos **Template**
  - Implementan los **Frozen Spots**
- Métodos **Hook**
  - Implementan los **Hot Spots**

Metapatrones

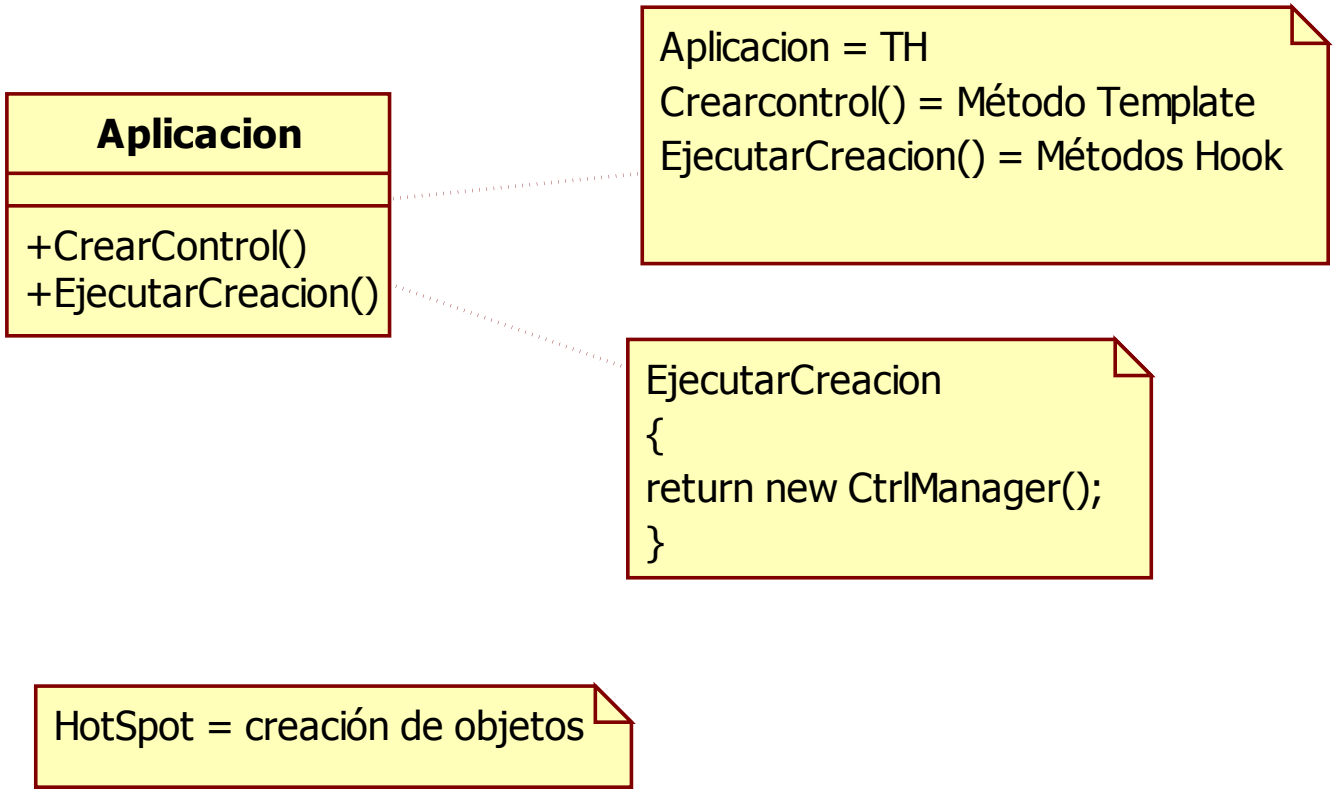
- Existen 5 patrones que en general se pueden aplicar en el diseño de frameworks
  - Unificación
  - Conexión 1 a 1
  - Conexión 1 a N
  - Conexión recursiva 1 a 1
  - Conexión recursiva 1 a N

Patrón Unificación

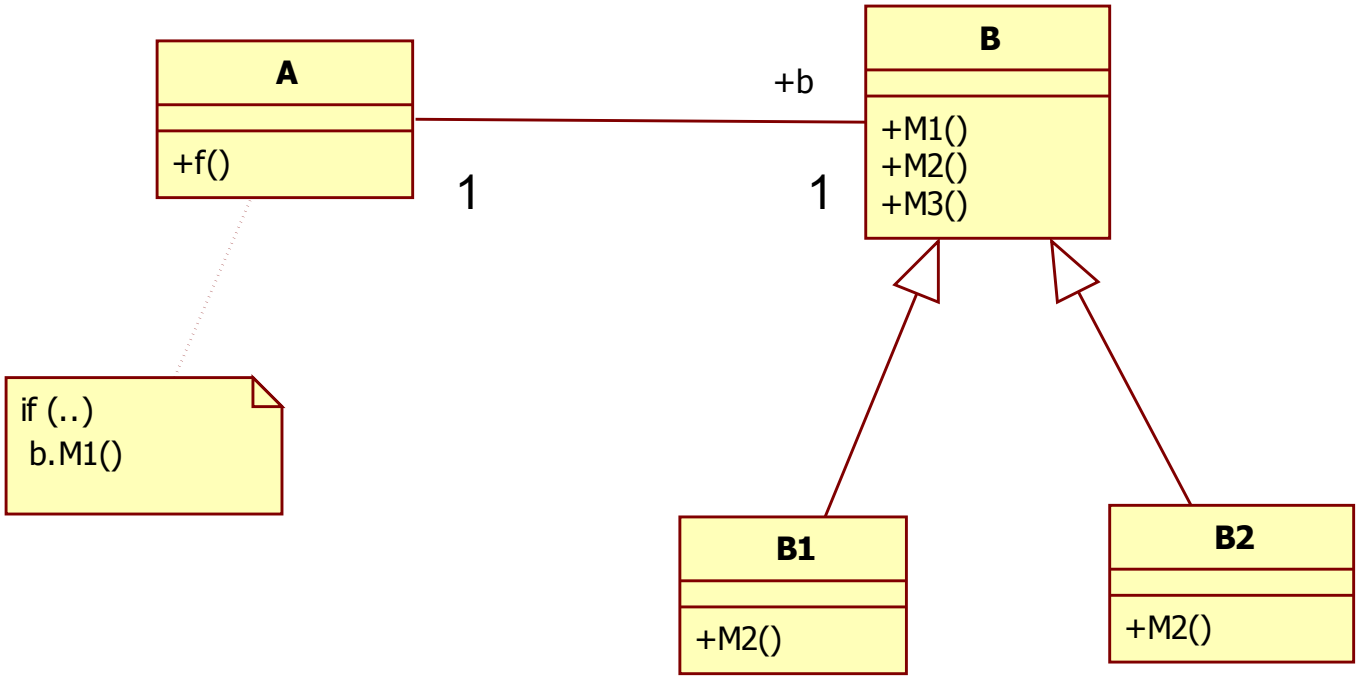
- Hot Spots y Metapatrón Unificación



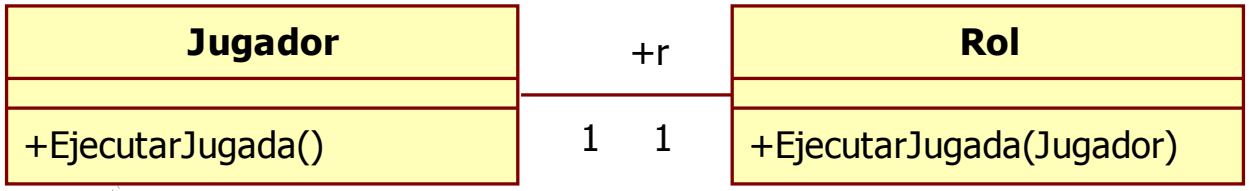
Patrón Unificación



Patrón Conexión 1 a 1



Patrón Conexión 1 a 1

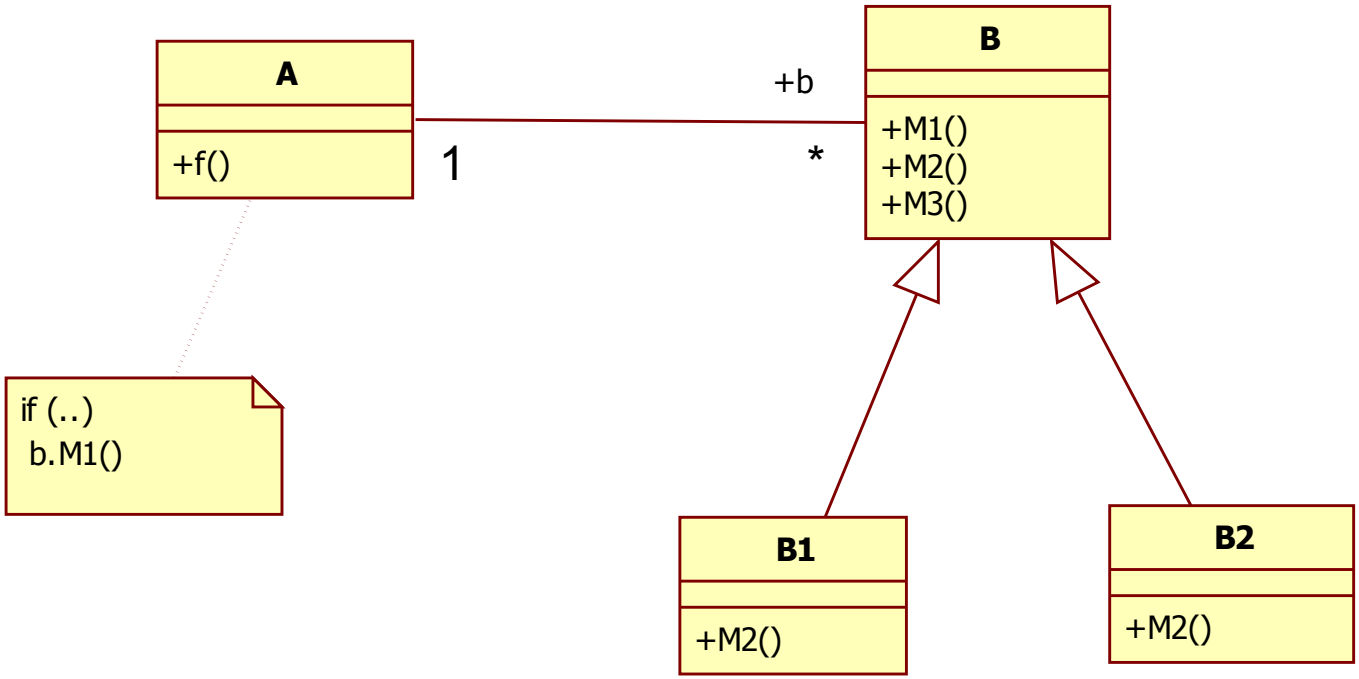



```
EjecutarJugada()
{
 r.EjecutarJugada(this)
}
```

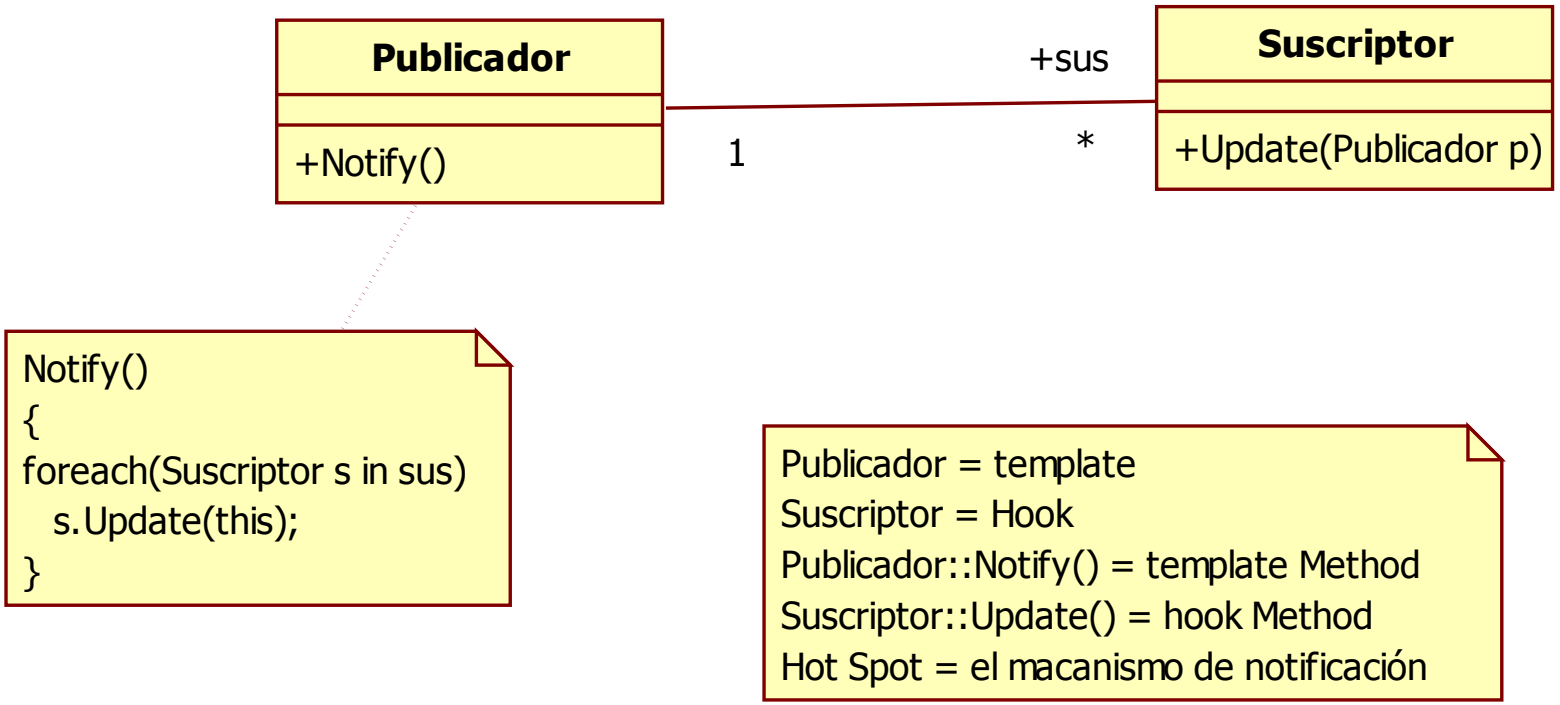
Jugador = Template
Rol = Hook
Jugador::EjecutarJugada = Template Method
Rol:EjecutarJugada() = Hook Mehtod

HotSpot = como se ejecuta el pedido de Jugador

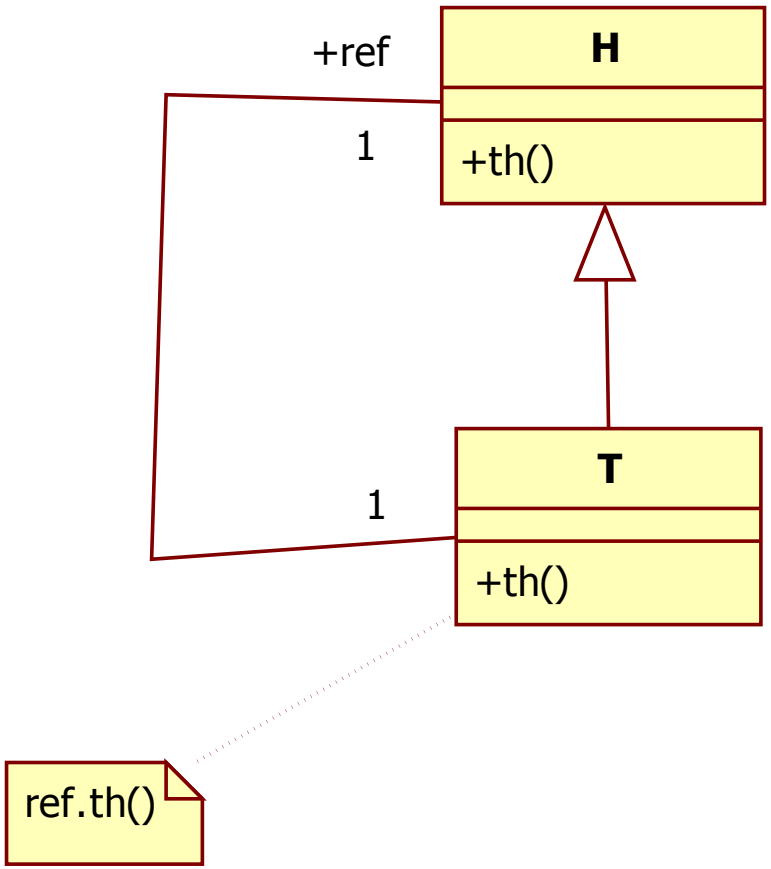
Patrón Conexión 1 a N



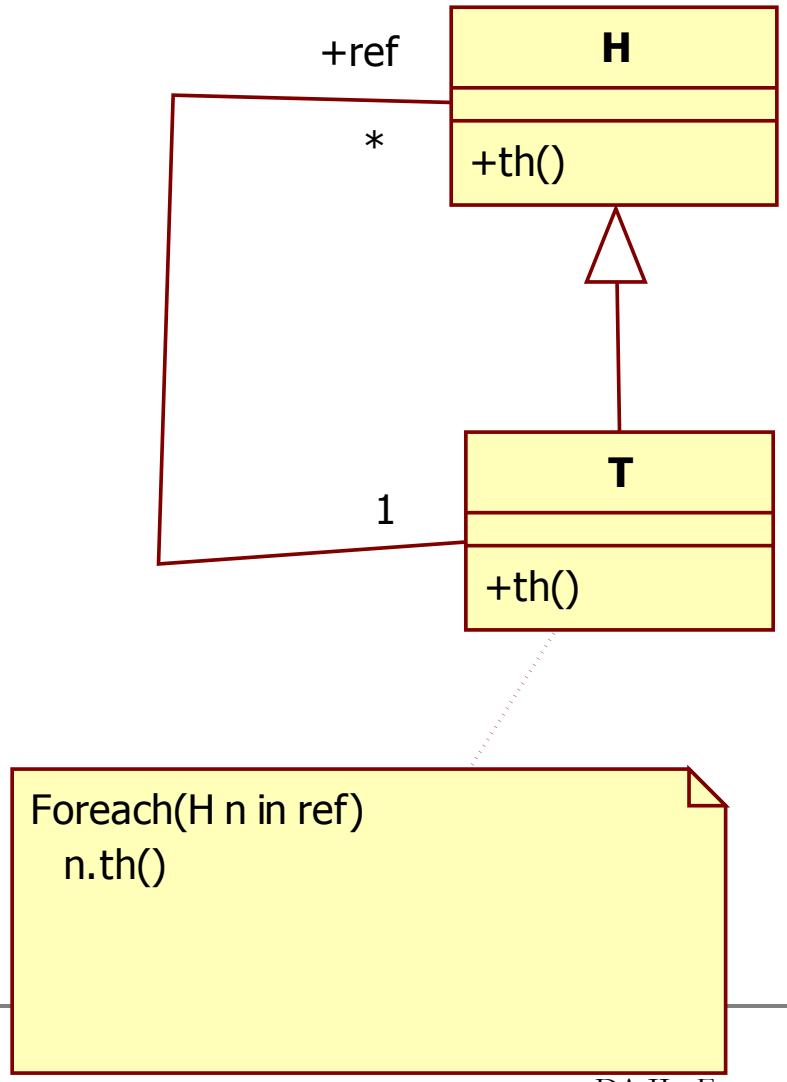
Patrón Conexión 1 a N



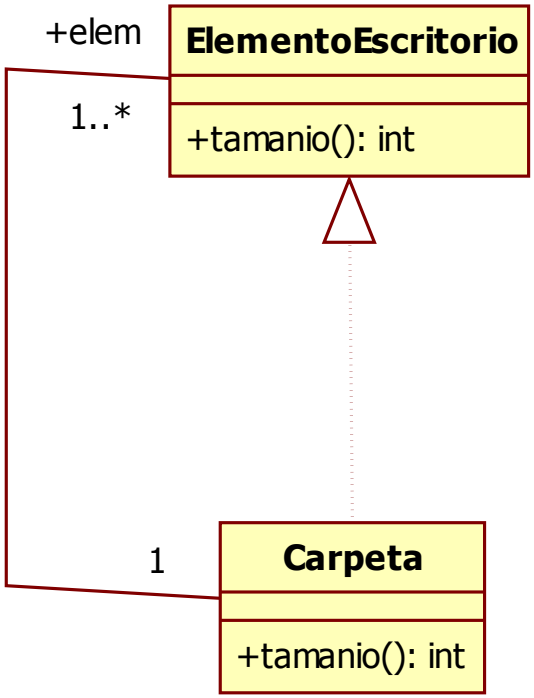
Patrón Conexión recursiva 1 a 1



Patrón Conexión recursiva 1 a N



Patrón Conexión recursiva 1 a N



ElementoEscritorio = Hook
Carpeta = Template
ElementoEscritorio::Tamaio() = TemplateHook
Carpeta:tamanio() = TEmplateHook

Hot Spot = Caculo del tamanio

Patrones y Metapatrones

- Patrones de diseño basados en Unificación
  - Template Method
  - Bridge

Patrones y Metapatrones

- Patrones de diseño basados en Conexión
  - Builder
  - Abstract Factory
  - Command
  - Interpreter
  - Observer
  - State
  - Strategy

Patrones y Metapatrones

- Patrones de diseño basados en Conexión Recursiva
  - Composite
  - Decorator
  - Chain of Responsability

Proceso de Desarrollo de Frameworks

- **Proceso basado en la experiencia**
- **Proceso basado en análisis de dominio**

Desarrollo de Frameworks - Basado en la experiencia

1. Desarrollar 3 aplicaciones utilizando el método de desarrollo tradicional
2. Identificar los elementos comunes a las tres aplicaciones y extraerlo para el framework
3. Reimplementar las aplicaciones utilizando el framework
4. Mejorar el framework
5. Desarrollar nuevas aplicaciones al tiempo que se evoluciona el framework

- Se basa en un ciclo de vida iterativo

Desarrollo de Frameworks - Análisis de Dominio

- Definición
  - El intento de identificar los objetos, operaciones y relaciones que **expertos del dominio** perciben como importantes sobre el dominio
- Busca:
  - Identificar las clases y objetos que son comunes a todas las aplicaciones de un dominio
  - Identificar el vocabulario común utilizado en el dominio

Desarrollo de Frameworks - Análisis de Dominio

- Pasos para realizar un Análisis de Dominio:
  1. Construir un modelo inicial genérico del dominio para discutir con los expertos del dominio
  2. Examinar varios sistemas existentes y representar las abstracciones del dominio en modelos
  3. Consultando con expertos del dominio, identificar las similitudes entre los sistemas
  4. Refinar el modelo genérico para que contemple los sistemas existentes

Desarrollo de Frameworks - Análisis de Dominio

- Análisis de Requerimientos Vs. Análisis de Dominio
  - El **análisis de requerimientos** se utiliza para identificar **los requerimientos funcionales y no funcionales de una aplicación**
  - El **análisis de dominio** se utiliza para identificar todos **los elementos comunes a las aplicaciones de un dominio**

Desarrollo de Frameworks - Análisis de Dominio

- Basado en Análisis de Dominio
  1. Realizar el **análisis de dominio**
  2. Identificar “Hot Spots”
  3. Desarrollar el framework
  4. Desarrollar una o más aplicaciones para probar el framework
  5. Modificar el framework
  6. Evolucionar el framework
- Se basa en un ciclo de vida iterativo

Refactoring

- Consiste en modificar el código para mejorar la estructura (diseño) y desempeño del sistema sin impactar la correctitud y el comportamiento del mismo
- Es necesario realizarla, en forma disciplinada, al final de cada iteración

Aspectos importantes del Refactoring

- Un buen diseño permite que un sistema sea: más fácil de mantener, extensible y perdurable
- Es una realidad empírica del desarrollo de software que:
  - Cuando se codifica un diseño se descubren aspectos que el diseño original no contempló
  - Las presiones normales de un proyecto hacen que el desarrollo no sea tan disciplinado como se espera
  - Los desarrolladores tiene distintos niveles de experiencia por lo que la calidad del código no es uniforme

Aspectos importantes del Refactoring

- La reestructuración del código y del diseño es la mejor forma de mantener y mejorar el diseño del sistema
- Las técnicas de reestructuración **son aplicables a cualquier desarrollo y no exclusivamente al desarrollo de frameworks**
- Las actividades de reestructuración se deben realizar durante o entre las iteraciones del ciclo de vida del desarrollo

Documentación de Frameworks

- La documentación de un framework debe estar orientada a:
  - Usuarios del Framework
    - Descripción general y ejemplos
    - Tutorial sobre el uso del framework
    - Documentación de diseño que permita modificar el framework
  - Desarrolladores del Framework
    - Documentación de análisis, diseño, implementación y prueba

Documentación de Frameworks (Usuarios)

- Propósito del Framework
- Descripción del Uso del Framework
  - Instrucciones detalladas (estilo receta con ejemplos) de cómo utilizar el Framework
- Documentación de Diseño
  - Descripción de las clases del framework
  - Descripción de los mecanismos de interacción entre las clases
- Ejemplos

Resumen

- Frameworks
  - Definiciones
  - Clasificaciones
    - Alcance
    - Diseño
  - Beneficios y problemas del desarrollo y uso de frameworks
  - Frameworks Vs. Bibliotecas de Clases
  - Frameworks y Patrones
  - Conceptos de Diseño OO utilizados en Frameworks

Resumen

- Frameworks
  - Desarrollo de Frameworks
    - Basado en la Experiencia
    - Basado en Análisis de Dominio
      - Análisis de Dominio
      - Análisis de Dominio vs. Análisis de Requerimientos
      - Hot Spots, Frozen Spots y Patrón Template Method
  - Refactoring
    - Patrones GRASP
    - Abstracción de superclases

Resumen

- Frameworks
  - Refactoring
    - Abstracción de Composición
      - Migración a Componente
      - Migración a Compuesto
      - Conversión de Herencia
    - Eliminación de Condicionales
    - Conversión de segmentos de código en métodos
    - Extensión o envoltorio
    - Objeto Método
  - Documentación de un Framework

Bibliografía

- R.Johnson. Frameworks = (Components + Patterns).(CACM Oct.1997)
- M.Fayad, D Schmidt. Object Oriented Application Frameworks (CACM Oct 1997)
- R.Johnson, B. Foote. Designing Reusable Classes. 1991
- R.Johnson. Documenting Frameworks using Patterns, OOPSLA 1992
- Análisis de Dominio
  - G, Booch. Object Oriented Analysis and Design. Pg. 157
- Refactoring
  - W. Opdyke, R. Johnson. Creating Superclases by Refactoring
  - R. Johnson, W. Opdyke. Refactoring and Aggregation
- Patterns
  - GOF
  - F. Buschmann, et. Al. Pattern Oriented Architectures.

Framework Architectures! Pattern usage for success - *John Earles, Castek*

Framework Architectures! Pattern usage for success

- Artículo que trata sobre la estructura interna de los frameworks bien diseñados y sus componentes
- Ya que no se puede eliminar los cambios en los requerimientos, la única solución es tratar de mitigar los efectos
- Las áreas propensas a cambiar deben ser tomadas en cuenta como “hot spots” y tratarlas apropiadamente en el diseño e implementación
- Las siguientes técnicas representan patrones orientados a objetos y buenas prácticas para aplicar a la hora de implementar esos “hot spots”

Framework Architectures! Pattern usage for success

- ***Accessors and Mutators***
  - Todos los atributos deberían ser privados y ser accedidos a través de “get” y modificados con “set”
  - Todos los usos deben de ser a través de esos métodos, incluso dentro de la misma clase
  - Los métodos “get” y “set” son la interfase de los atributos. Mediante esta interfase, un cambio en los atributos, tiene un impacto mínimo en los clientes ya existentes.

Framework Architectures! Pattern usage for success

- ***Interface abstraction***
  - Uno de los mecanismos para mitigar los cambios es la separación temprana de las interfases de la implementación
  - Las interfases proveen un nivel de abstracción. La abstracción introduce una capa que separa al que llama (el cliente) de los cambios en el llamado (el que provee la implementación)
  - Proveen una forma fácil de que métodos acepten parámetros con distintas implementaciones, manteniéndose representativo del tipo de dato

Framework Architectures! Pattern usage for success

- ***Strategy Pattern***
  - Los patrones de abstracción son una manera de protegerse de los cambios de todo tipo
  - Nunca se debe hacer “hardcode” una sección de código que muestre variabilidad
  - Estas áreas siempre se pueden abstraer, encapsular utilizando el patrón estrategia

Framework Architectures! Pattern usage for success

- ***Implementation Inheritance***
  - La herencia tiene un gran acoplamiento entre la subclase y la superclase
  - Una vez que se instancia una subclase, se obtiene el comportamiento de esa clase y no se puede cambiar, es estático. En cambio, la composición habilita la flexibilidad en tiempo de ejecución
  - Una instancia de un objeto es capaz de recrear una o más de sus partes compuestas, y cambiar así su comportamiento a medida que ejecuta. Este tipo de habilidad tiene un gran incremento en la flexibilidad, adaptabilidad, y resistencia al cambio
  - Por esta razón, es que los Frameworks deben apoyarse en la composición y en las interfases antes que en la herencia de implementación

Framework Architectures! Pattern usage for success

- ***Factory Pattern***
  - Encapsula, abstrae
  - La fábrica contiene toda la lógica para crear el objeto correcto de una jerarquía de herencia
- ***Templates and Hooks***
  - One or more abstract types joined together by a “uses a” relationship with the composite forwarding messages to its appropriate part
  - The composite method being labeled the ***template*** and the part method labeled the ***hook***
  - Template Method Pattern

Framework Architectures! Pattern usage for success

- ***Role Object Pattern***
  - Una clase tiene una colección (composición) de roles que puede realizar
  - Cada rol puede mantener una referencia a su entidad central, y por lo tanto es capaz de "decorar" el objeto con su comportamiento adicional
  - El uso del concepto de rol mantiene a la entidad central privada del conocimiento, pero proveyendo un mecanismo para acceder ese comportamiento cuando sea necesario

Framework Architectures! Pattern usage for success

- ***Metadata***
  - Capturando la especificación de los datos, se puede automatizar la creación de tipos nuevos en el momento de ejecución sin tener que modificar nada de código
  - El uso de metadatos provee el nivel más alto de flexibilidad para un framework pero es mas difícil de implementar

UML-F

UML-F

- Es común que hoy en día los usuarios de un framework tengan que navegarlo para identificar los puntos variables
- Es importante que los desarrolladores de un framework provean documentación que describa la parte del sistema que debería ser adaptada para obtener una instancia válida (puntos variables del framework)

UML-F

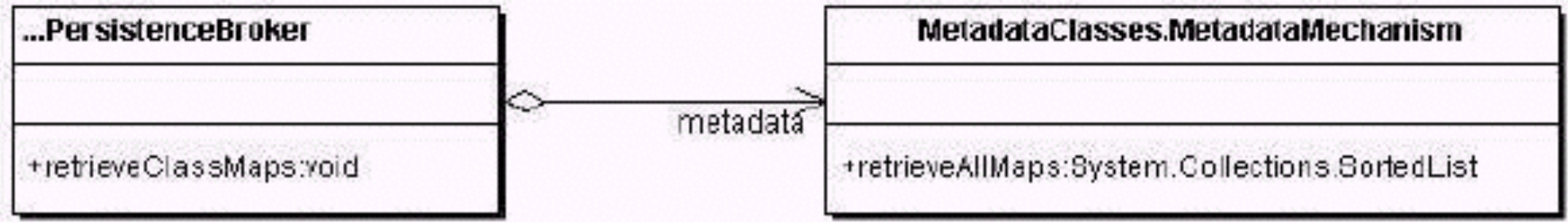
- No hay indicadores en los diagramas de UML de cuales son los puntos de variación y las restricciones de instanciación
- UML provee mecanismos de extensión que nos permiten definir etiquetas y marcas apropiadas para los elementos del modelo
- Se agregan extensiones al modelo para permitir la mejor comunicación del diseño de un framework

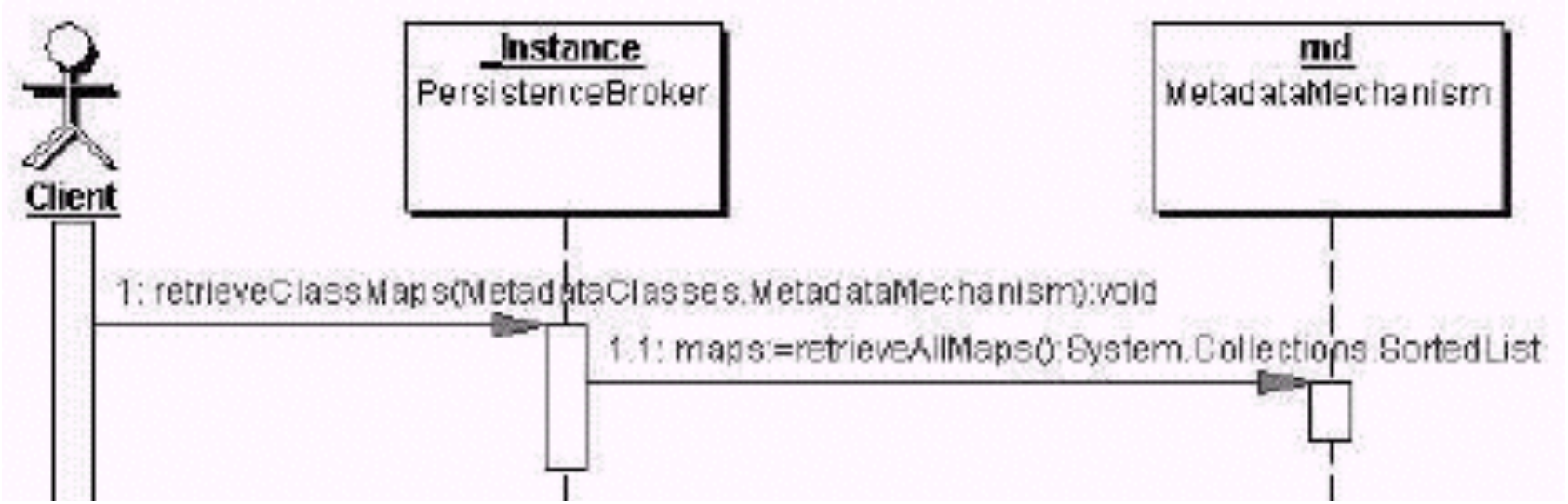
UML-F

- Utilizando los mecanismos de extensibilidad de UML se agregan elementos al lenguaje que forman las bases para un nuevo perfil de UML llamado **UML-F**
- El objetivo es usar un grupo pequeño de extensiones que capturen la semántica de los tipos más comunes de puntos de variación en los OO Frameworks

UML-F

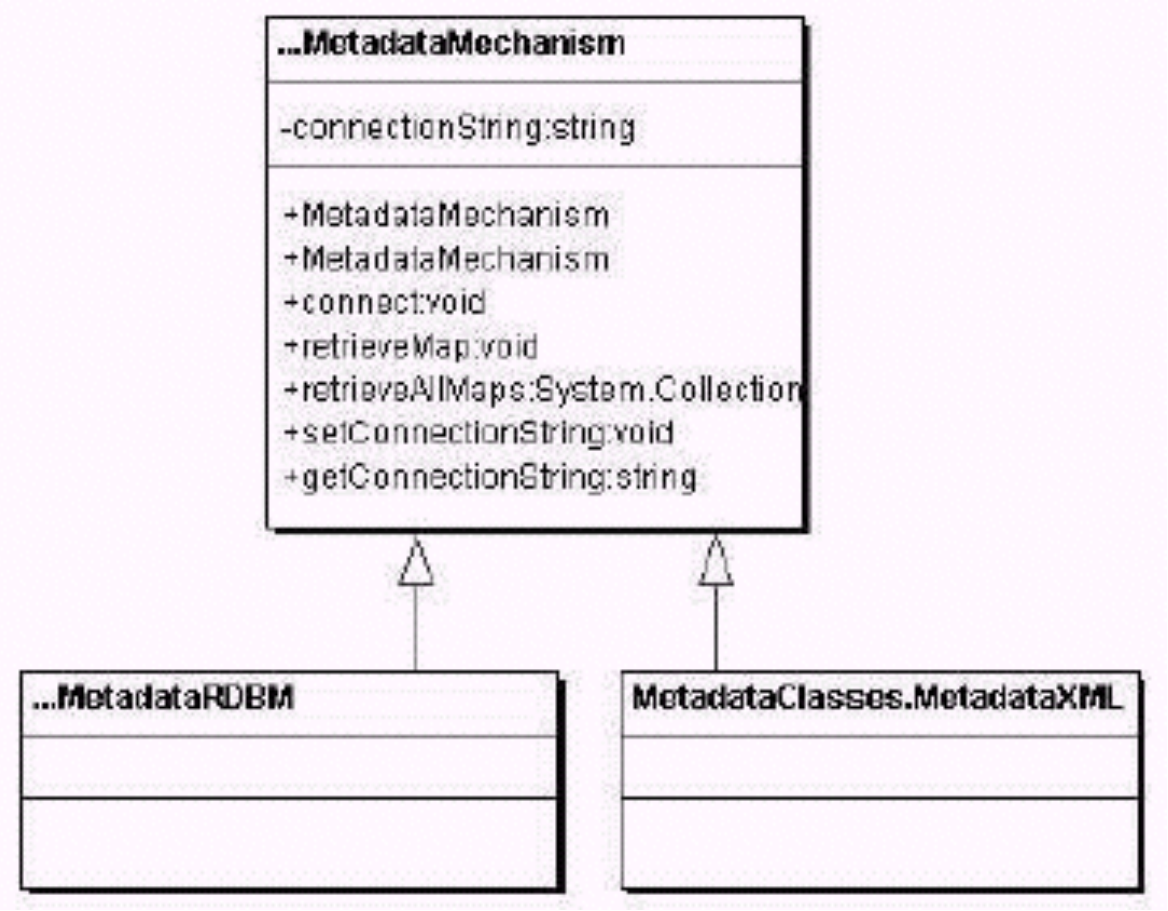
- Problema de ejemplo:





UML-F

- Problema de ejemplo:



UML-F

- UML provee 3 mecanismos de extensión:
  - ***Stereotypes***:
    - definición de extensiones al vocabulario de UML
      <<nombre>>
  - ***Tagged-values***:
    - extienden las propiedades de un elemento con cierto tipo de información
      {nombre=juan} o {incompleto}
  - ***Constraints***:
    - detallan como se pueden tratar los elementos de UML

UML-F

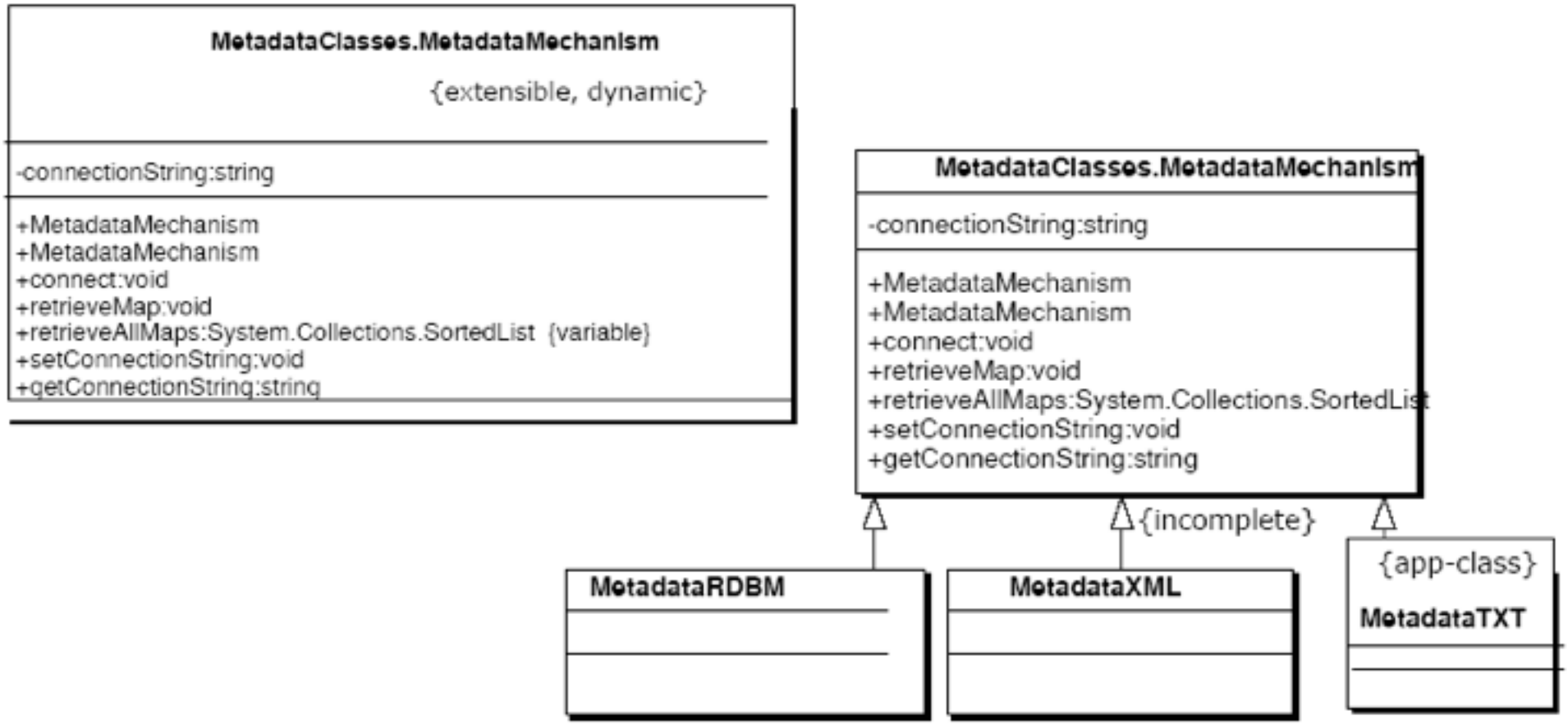
- Una vez definidas las extensiones es imprescindible definir su significado exacto
  - {variable}
    - Indica que un método puede variar su implementación dependiendo de la instanciación del framework. Tiene el propósito de mostrarle al usuario del framework que el método debe ser implementado con comportamiento específico para la aplicación, en cada instanciación del framework.
  - {extensible}
    - Indica que la interface de la clase puede ser extendida agregando nueva funcionalidad. Esto es opcional. Es importante notar que los cambios no tienen porque ser hechos directamente sobre la clase.

UML-F

- {incomplete}
  - Se aplica a relaciones de generalización, permitiendo definir nuevas subclases en instancias del framework.
- {app-class}
  - Indica clases en el diseño del framework que pertenecen a instancias particulares del mismo.
- {dynamic} {static}
  - Denota clases que requieren inicialización en tiempo de ejecución o no.

UML-F

- Solución de ejemplo:



UML-F

- También se agregan extensiones para hacer más claros los diagramas de interacción. Por ejemplo:
  - {optional}
    - Indica que una interacción no es imprescindible en la implementación de un método por parte de una instancia del framework

Documentación de Frameworks

- “Documenting Frameworks using Patterns” – Ralph E. Johnson
- “Documenting Frameworks” – Greg Butler, Pierre Denommée
- “Object-Oriented Frameworks. A survey of methodological issues” (capítulo 5) – Michael Mattsson

Documentación de Frameworks

- Los frameworks tienen como meta el rehúso mediante la personalización
- Un framework no es fácil de comprender:
  - Diseño abstracto
  - Diseño incompleto
  - Flexibilidad para los “hot spots”
  - Colaboraciones y dependencias indirectas y oscuras

Documentación de Frameworks

- Es difícil por el hecho de que distintos tipos de usuarios requieren distinta información:
  - Programadores de aplicaciones
  - Encargados de mantenimiento del framework
  - Diseñadores de otros frameworks
  - Verificadores

Documentación de Frameworks

- Para documentar un framework para asistir a los desarrolladores de aplicaciones se siguen estos tres puntos como guía:
  - Overview del framework
    - Se necesita un contexto para el framework, por eso una visión general del framework debe ser preparada, como una primer receta del “cookbook” y una presentación útil
  - Conjunto de aplicaciones de ejemplo
    - Diseñadas específicamente como herramientas de documentación. Estos ejemplos deben ser graduales, desde el mas simple hasta uno mas avanzado, y deben introducir incrementalmente un “hot spot” por vez
  - Cookbook
    - Las recetas deben usar los ejemplos de aplicaciones para hacer sus discusiones concretas. Habrá referencias cruzadas entre las distintas recetas, y entre las recetas y el código fuente

Documentación de Frameworks

- Estilos de documentación:
  - **Example Application**
    - El código fuente de las aplicaciones de ejemplo que fueron construidos usando el framework es comúnmente la primera y única documentación provista para desarrolladores. La documentación requiere un conjunto de ejemplos graduado. Cada uno debería ilustrar un nuevo “hot spot”, comenzando con el rehúso mas simple y común para ese “hot spot”, y eventualmente proveyendo una cobertura completa. La mayoría de los cookbooks giran entorno a un pequeño número de ejemplos simple.

Documentación de Frameworks

- Estilos de documentación:
  - **Recipe**
    - Una receta describe como llevar a cabo un ejemplo de rehúso típico durante el desarrollo de la aplicación. La información es presentada en lenguaje informal, capaz con algunas imágenes, y usualmente con código fuente de ejemplo. Aunque informal, una receta sigue generalmente una estructura, como secciones de propósito, pasos en la receta, referencias cruzadas a otras recetas, y ejemplos de código fuente.
  - **Cookbook**
    - Es una colección de recetas. Una guía del contenido de las recetas es provista generalmente, ya sea como una tabla de contenido o mediante la primera receta actuando como una visión general del cookbook.

Documentación de Frameworks

- Estilos de documentación:
  - **Pattern Language**
    - Johnson introduce un lenguaje de patrones informales, que puede ser usado para documentar un framework en lenguaje natural. Los patrones proveen un formato para cada receta, y una organización para el cookbook. La organización presenta primero las recetas para las formas de rehúso mas frecuentes, y los conceptos y detalles son retrasados lo mas posible. La primera receta es una visión general de los conceptos del framework y de las siguientes recetas.

Documentación de Frameworks

- Estilos de documentación:
  - **Motif**
    - Usan una plantilla para la descripción de temas que tienen un nombre y una intención, una situación de rehúso, los pasos involucrados en la personalización, y referencias cruzadas a temas, patrones de diseño, y contratos. Los patrones de diseño proveen información sobre la arquitectura interna, y los contratos proveen una descripción más rigurosa de las colaboraciones relevantes de los temas.
  - **Interface Contract**
    - Un contrato es una especificación de obligaciones. El "interface contract" de una clase provee una especificación de la interface de la clase y las invariantes de aislamiento de la clase.

Documentación de Frameworks

- Estilos de documentación:
  - **Interaction Contract**
    - Un “interaction contract” trata con el comportamiento cooperativo de unos cuantos participantes que interactúan para llegar a una meta en común. El contrato específica un conjunto de participantes comunicados en sus obligaciones contractuales: el tipo de impedimento dado por la firma de un método, la semántica de la interfase del método, y el impedimento en el comportamiento que capture las dependencias de comportamiento entre objetos.

Documentación de Frameworks

- Estilos de documentación:
  - **Design Pattern**
    - Un patrón de diseño presenta una solución a un problema de diseño que puede surgir en un contexto dado. La descripción de un patrón de diseño describe el problema y su contexto, la solución, y una discusión de las consecuencias al adoptar la solución. El problema puede ser ilustrado mediante un ejemplo concreto. La solución describe los objetos y clases que participan en el diseño, y sus responsabilidades y colaboraciones. Un diagrama de colaboración puede ser usado para representar la misma información. Ejemplos de la solución siendo utilizada en situaciones concretas pueden ser provistos. El análisis de los beneficios y balance al aplicar este patrón es una parte importante de la descripción del patrón de diseño.

Documentación de Frameworks

- Estilos de documentación:
  - **Framework Overview**
    - Mostrar el contexto de un framework es el primer paso para ayudar al desarrollador de aplicaciones a rehusar el framework. Un “framework overview” define la jerga del dominio y limita el alcance del framework: lo que es cubierto por el framework y lo que no, como también que es lo fijo y lo flexible en el framework. Una aplicación simple puede ser analizada, y una visión general de la documentación puede ser presentada. Un “framework overview” es por lo general la primera receta en un cookbook.

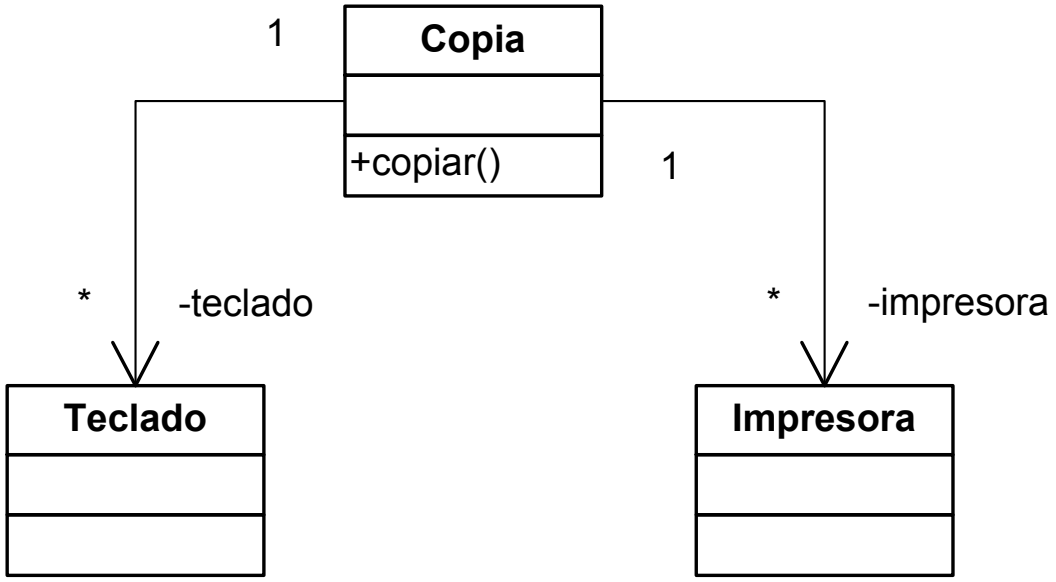
Documentación de Frameworks

- Estilos de documentación:
  - **Reference Manual**
    - Un manual de referencia para un sistema orientado a objetos consiste en una descripción de cada clase. Típicamente, la descripción de una clase presenta la responsabilidad o propósito de la clase, el rol de cada atributo, y alguna información sobre cada método. Una descripción de un método presenta la funcionalidad del método, su pre y pos condición, y una indicación que de atributos afecta su uso. Para la documentación de frameworks la descripción puede incluir material adicional concerniente del rol de una clase o método para proveer flexibilidad para un “hot spot”.

Ejercicios

Ejercicio 1

- Suponga que una parte de una aplicación que se está desarrollando plantea la necesidad de escribir un programa que lea una cierta cantidad de caracteres desde el teclado y los copie hacia un dispositivo de impresión. Una solución a este problema es la representada en el siguiente diagrama:



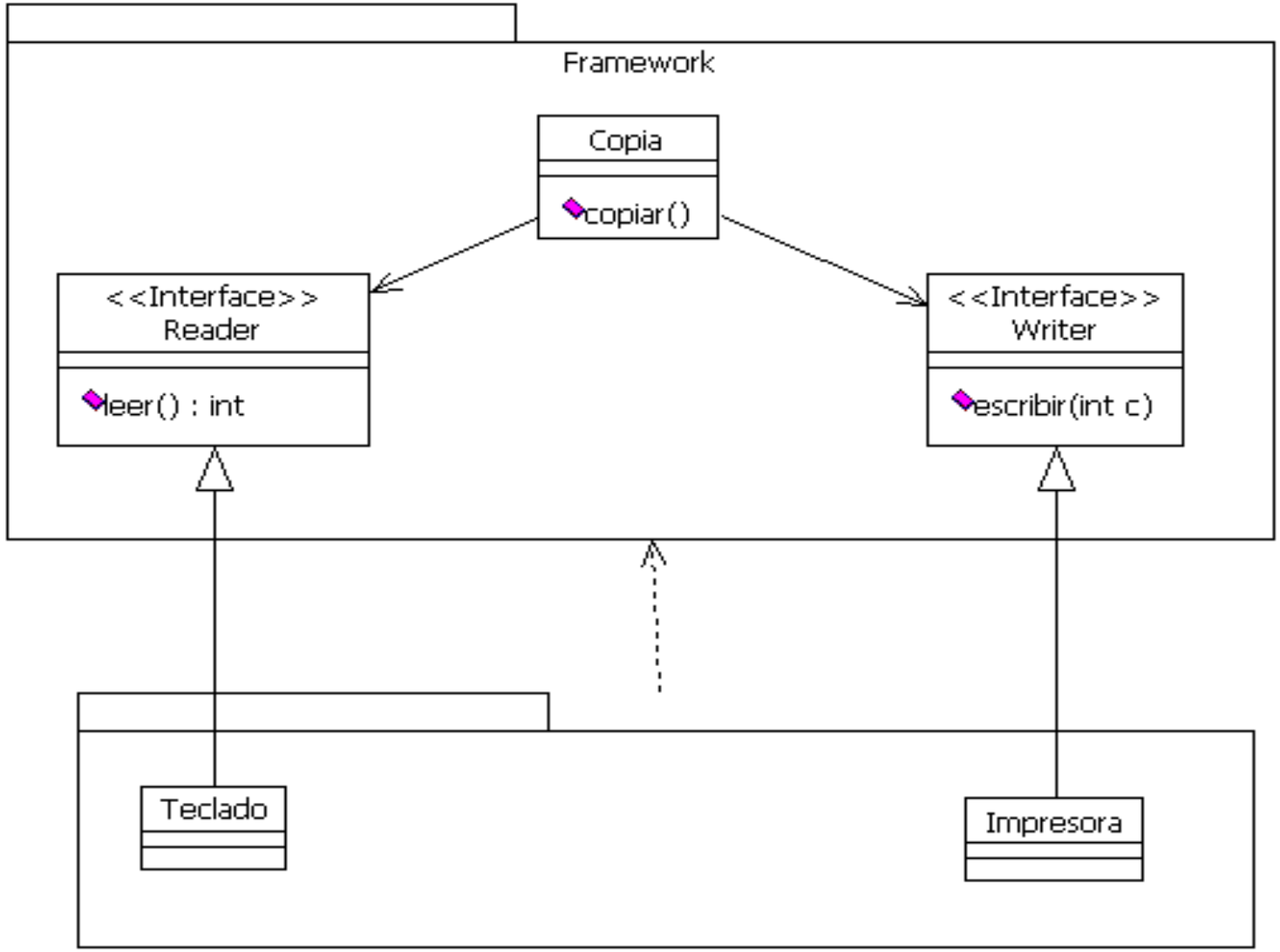
Ejercicio 1

- Supóngase que la implementación del método copiar() es como la que sigue:

```
public void copiar() {
        int c;
        while ((c = teclado.leerCaracterTeclado() ) != EOF())
                impresora.imprimirCaracter(c);
}
```

- **Se pide**:
  - Analizar las características de la solución en función de objetivos del diseño de frameworks como: la reutilización del diseño y la inversión de control
  - Plantee una solución alternativa a la presentada más arriba, teniendo en cuenta el punto anterior

Ejercicio 1 (solución)



Ejercicio 1 (solución)

- *copiar(Reader r, Writer w)*

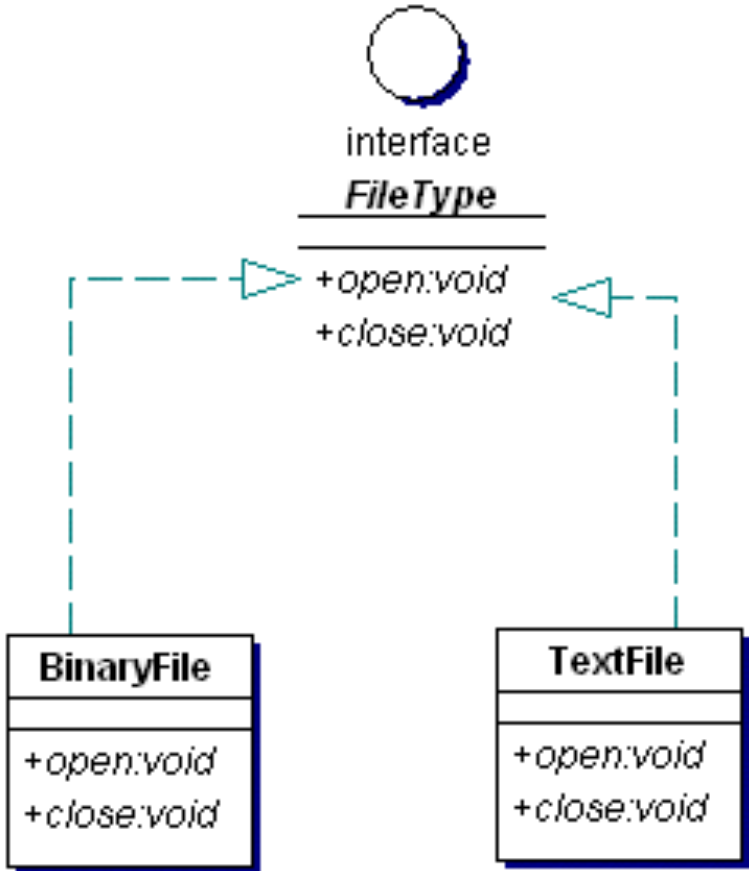
```
{
        int c;
        while( c = r.leer() )
        {
                w.escribir(c);
        }
}
```

- *En una clase Cliente:*

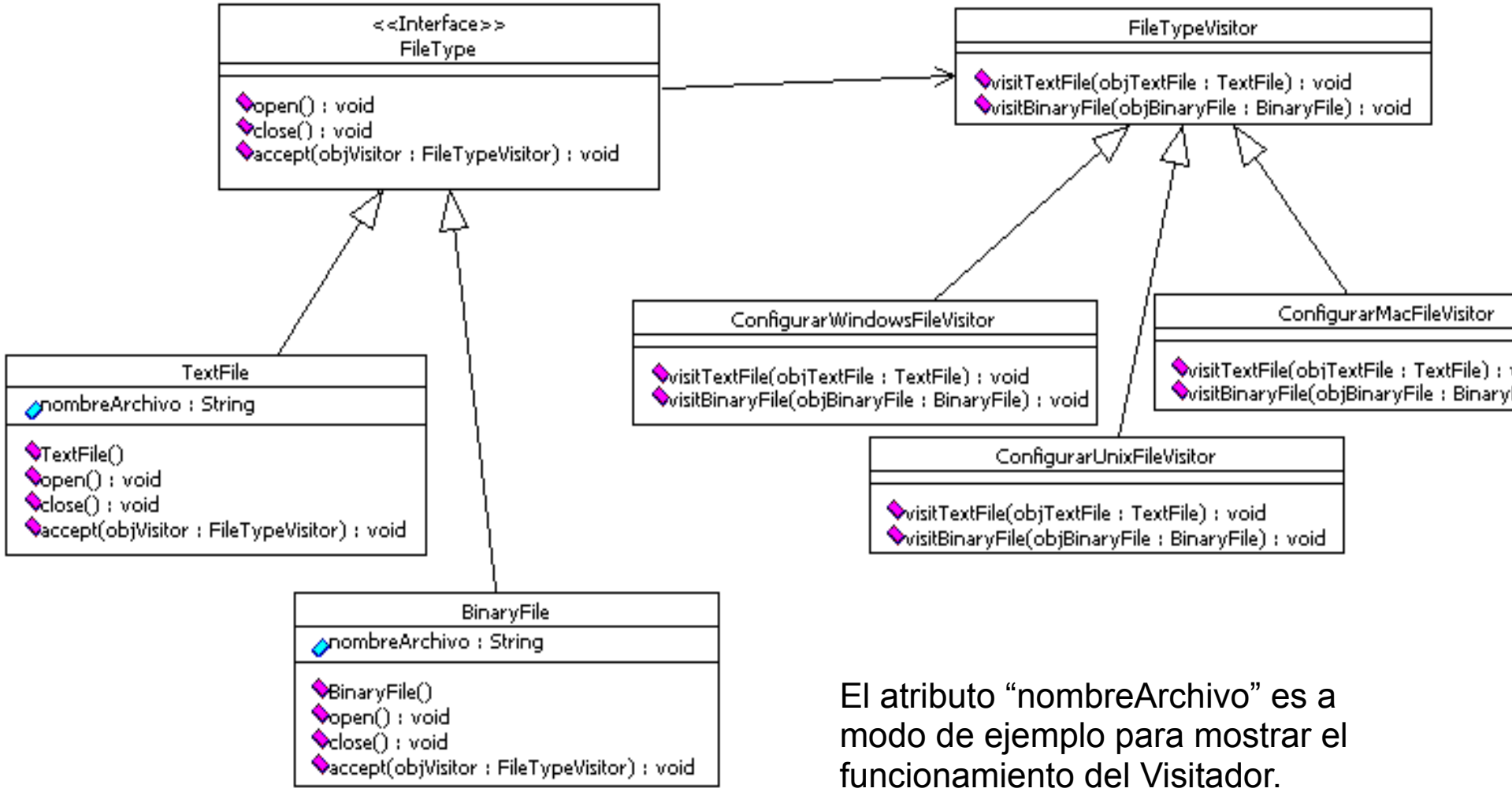
```
public void copiar(){
        Reader r = new Teclado();
        Writer w = new Impresora();
        Copia.copiar(r, w);
}
```

Ejercicio 2

- Observe las entidades de la figura. Ahora pensemos en agregar una operación **configurarUnixFile()**; Como podemos configurar la representación de nuestros **archivos** sin agregar el método a la interfase? Es decir, piense en que nos han advertido acerca de la extensión de los sistemas operativos sobre los cuales se deben configurar nuestros archivos y que posiblemente aparezcan operaciones ***configurarWindowsFile()***, ***configurarMacFile()***;



Ejercicio 2 (solución)



El atributo “nombreArchivo” es a modo de ejemplo para mostrar el funcionamiento del Visitador.

Ejercicio 3

- Se desea construir un framework para facilitar la prueba automática de una aplicación.
  Cada caso de prueba es representado por una clase.
  Cada clase probadora tiene un método que ejecuta la prueba.
  Antes y después de la ejecución de ese método de prueba se realiza una inicialización de los datos de prueba y luego del mismo la desinicialización.
  Tanto el método de prueba como el proceso de inicialización / desinicialización es particular de cada clase.
  En el framework se cuenta con una clase Runner, que utiliza una interfaz Test para recibir y ejecutar pruebas.
- ¿Cómo diseñaría su solución para asegurar la ejecución de los pasos de la prueba?

Ejercicio 3 (solución)

- Definir un Template Method para la ejecución de la prueba. Los pasos son inicilización(), runTest() y desinicializacion().