El rectángulo grande rotulado **«Modelo de dominio»** es un *paquete* que agrupa las clases de interés (Directory, FileTransfer, FileSysUtil\_Rev). Fuera del paquete aparecen:

* CLOTest (programa de arranque).
* java.lang::Thread (clase estándar de Java que provee concurrencia).

**1. Anatomía de las clases (cajas)**

| **Símbolo** | **Significado** |
| --- | --- |
| **Nombre en negrita** | Identificador de la clase. |
| Compartimento superior | Atributos ▸ - = **privado**, + = **público**. |
| Compartimento inferior | Operaciones (métodos) y constructores. |
| *Nombre en cursiva / «Object»* | Estereotipo (p. ej. indica que la clase es un wrapper sencillo) o cursiva para concreto/abstracto. |

Ejemplo

markdown

CopiarEditar

Directory

-------------

- name : String

-------------

+ Directory(String)

+ Directory() -- constructor sobrecargado

**2. Relaciones y los símbolos usados**

| **Nº** | **Elemento visual** | **Tipo UML** | **Intención en este modelo** |
| --- | --- | --- | --- |
| **A** | **Línea continua** con extremos navegables, roles src, dest | **Asociación** | FileTransfer **tiene** (mantiene referencia a) dos instancias de Directory. |
| **B** | **Triángulo hueco** en línea continua | **Generalización / herencia** | FileTransfer **extiende** java.lang.Thread; hereda start(), run(), etc. |
| **C** | **Línea discontinua (flecha abierta)** + «instantiate» | **Dependencia** (creación) | Una clase **crea** instancias de otra con new. Ej.: CLOTest → Directory, CLOTest → FileTransfer. |
| **D** | **Línea discontinua (flecha abierta)** + «use» | **Dependencia** (uso) | Una clase **invoca** métodos de la otra sin conservar referencia larga. Ej.: Directory usa utilidades de FileSysUtil\_Rev. |
| **E** | **Línea discontinua** + «use, instantiate» | Dependencia doble | FileTransfer **crea** y **llama** a FileSysUtil\_Rev dentro de run(). |
| **F** | Etiquetas -src, -dest en la asociación | **Roles** | Nombran la función que cumple cada extremo dentro de la relación. |

No aparecen diamantes, así que **no** hay composición ni agregación explícita; solamente asociaciones simples y dependencias.

**3. Flujo lógico que reflejan las relaciones**

1. **CLOTest.main()**
   * Crea («instantiate») dos objetos Directory y cinco FileTransfer.
   * Arranca cada hilo invocando start() (heredado de Thread).
2. **FileTransfer.run()**
   * Internamente instancia («instantiate») un objeto FileSysUtil\_Rev.
   * Llama («use») a moveContents(src,dest), pasando las referencias a los dos Directory asociados (roles **src** y **dest**).
3. **FileSysUtil\_Rev**
   * Realiza la lógica de mover/copiar contenidos entre los dos directorios; sólo *usa* las instancias, no las guarda.

**4. Por qué estas relaciones y no otras**

* **Dependencia (use)** se prefiere para acoplamientos **ligeros** o de corta vida (método estático, variable local).
* **Asociación** implica un atributo miembro; se modela cuando la relación dura *todo el ciclo de vida del objeto* (FileTransfer mantiene los Directory).
* **Herencia** formaliza el polimorfismo (FileTransfer es-un Thread).
* El estereotipo **«instantiate»** deja claro *quién* hace el new, útil para documentar responsabilidades de creación (patrones de fábrica, etc.).

**5. Clave interpretativa**

Si sigues la flecha de dependencias en orden:

CLOTest ⇒ crea FileTransfer ⇒ posee Directory ⇒ delega a FileSysUtil\_Rev ⇒ ejecuta la operación crítica **concurrencia** garantizada por la herencia de Thread.

Todo el diseño se resume en: un *launcher* inicia hilos que mueven datos entre dos carpetas usando una utilidad, manteniendo bajo acoplamiento gracias a dependencias ligeras y encapsulando la concurrencia en la subclase de Thread.