UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Modelado y programación.

Rosa Victoria Villa Padilla

[2025-1] (03/10/2024)

Práctica 4:

Proxy y Singleton.

Equipo: Christian.

- Leon Navarrete Adam Edmundo
- Rubio Resendiz Marco Antonio
- Valencia Pérez Guillermo Emanuel

Ciudad Universitaria, Cd. Mx. 2024

Anotaciones sobre la estructura de la práctica.

La práctica consistió en desarrollar un software de solicitudes de impresión de documentos para una empresa.

Elección de patrones de diseño.

Proxy.

Empleamos el patrón proxy para lo referente a la relación Cliente-Servidor que debe de haber en el uso del programa. Para ello se estructuró una interfaz **ImpresoraInterface** que define el comportamiento de una impresora (imprime y gestiona los permisos para esta acción).

La clase servidora recibe el nombre de **ImpresoraServidor**, que consiste en una clase que maneja en general las solicitudes de impresión de documentos, para esto se empleó el patrón de diseño Singleton para asegurar que solo exista una única instancia de la clase y se empleó una cola concurrente "ConcurrentLinkedQueue" para asegurar una sincronización entre los posibles hilos que se generan con el programa.

La clase que representa al proxy se llama **ImpresoraProxy**, que en general sobrecarga los métodos de una implementación de ImpresoraInterface, su único propósito es ser capaz de comunicarse con otra impresora.

Además, el programa cuenta con una clase que sirve de intermediaria con las dos clases mencionadas con anterioridad. Esta clase recibe el nombre de **ClienteRemoto** y solo cuenta con un método que busca la instancia de ImpresoraServidor para establecer la comunicación con una instancia de ImpresoraProxy. Esta clase se escribió con el objetivo de desacoplar la clase proxy de la clase servidora y establecer una comunicación entre ellas de una forma más orgánica.

Singleton.

Se usó el patrón de diseño singleton para la única impresora de la empresa, la clase en cuestión es **ImpresoraServidor** y en general es la clase que maneja todas las solicitudes de impresión por parte de los empleados. Se utilizó este patrón para que todas las solicitudes que se realicen sean destinadas a la misma instancia de esta clase.

Respecto al uso del proyecto.

El programa fue desarrollado en la versión de java 11.0.22 y se empleó linux (debian y ubuntu) durante su desarrollo. Para ejecutarlo en linux (para distribuciones basadas en debian) es necesario usar los siguientes comandos en terminal:

Cabe mencionar que para usar el programa es necesario que se utilicen dos terminales, una para encender el servidor donde se recibirán todas las solicitudes de los empleados y otra para hacer uso del programa como un empleado de la empresa (para mandar las solicitudes). Es necesario que el servidor esté encendido para que se puedan realizar solicitudes de impresión.

Para ejecutar el programa:

Para compilar todas las clases y generar los archivos .class

```
$ javac -d bin $(find src -name "*.java")
```

Para encender el servidor (Terminal 1):

```
$ java -cp bin mx.unam.ciencias.modelado.practica4.singleton.ImpresoraServidor
```

Para ejecutar el programa como empleado, se realiza una simulación. (Terminal 2):

```
$ java -cp bin mx.unam.ciencias.modelado.practica4.Main
```

Importante: Una vez ejecutado el Main, el programa lanza la siguiente excepción:

```
$ Error de comunicación remota: Comunicación interrumpida.
$ Error unmarshaling return header; nested exception is:
$    java.io.EOFException
```

Esto sucede debido a que, una vez ejecutadas las simulaciones de prueba, el servidor de la impresora procesa una cola de solicitudes (de longitud máxima 2), realiza las impresiones y cierra el servidor, por ello que la comunicación entre el proxy y el servidor se interrumpe y el programa termina. Esto se hizo con el único propósito de que el programa terminara después de realizar la simulación.