Práctica ORB 1. Invocación remota a un objeto de una clase

Objetivos específicos que persigue la realización del trabajo.

- Transformación de un programa que realiza una llamada a un objeto local en otro que realiza la llamada de forma transparente al mismo objeto situado en un nodo remoto.

Fecha	Grupo	Tiempo resolución	Tiempo de resolución

Apellidos y nombre	Grupo	Calificación

Enunciado

La clase Agenda

Un repositorio es un almacén de datos. El tipo y la organización de los datos depende del servicio de almacenamiento que se quiera ofrecer. Supongamos que se cuenta con un interfaz `IRepositorio` que define la semántica de un servicio de almacenamiento en forma de tuplas <clave, valor>. Donde la clave es una tira de caracteres y el valor asociado es un número entero.

Este servicio de almacenamiento está definido por dos operaciones, como se puede ver a continuación en esta interfaz:

```
/*************************************************************/
// FICHERO: interfaz IRepositorio
/*************************************************************/
interface IRepositorio {

        // Se asocia un valor entero a una cadena
        public void asociar(String key, int d);

        // Se obtiene el valor entero asociado previamente a la
        // cadena especificada.
        public int obtener(String key);
}
```

El método `asociar` permite almacenar una tupla <Tira, valor>. El método `obtener` devuelve el valor asociado a una clave previamente almacenada. En este servicio no pueden almacenar tuplas con claves iguales, pero sí es posible que dos tuplas con claves distintas tengan asociadas el mismo valor.

Un sistema de almacenamiento que se ajusta a la semántica definida por `IRepositorio` es una agenda clásica. Para ello, se crea la clase `Agenda` que implementará la interfaz `IRepositorio`. Internamente la clase `Agenda` usa una tabla hash para mantener las tuplas de información. El uso de la tabla

hash es por simplificar la solución, pero se podría haber elegido una base de datos o un fichero.

```
/*************************************************************/
// FICHERO: Agenda
/*************************************************************/
import java.util.Hashtable;

public class Agenda implements IRepositorio {
     private Hashtable<String, Integer> ht = new Hashtable<String, Integer>();

    public void asociar(String s, int v) {
    //Inserta una nueva tupla en la tabla hash
          ht.put(s, new Integer(v));
    }

    public int obtener(String s) {
    //Obtiene el valor asociado a la clave s
          return ((Integer) ht.get(s)).intValue();
    }
}
```

Uso local de la agenda

Como ejemplo de uso de la clase Agenda, se muestra un programa cliente que crea una agenda telefónica, sobre la que se inserta y se recupera un contacto.

```
/*************************************************************/
// FICHERO: ClienteUnaAgenda
/*************************************************************/
public class ClienteUnaAgenda{

        public static void main(String[] args) {

                Agenda agendaTelefonica = new Agenda();
                agendaTelefonica.asociar("Juan", 66756677);
                System.out.println("Telefono Juan = " + agendaTelefonica.obtener("Juan"));
        }
    }
```

Observe que el objeto agendaTelefonica se almacena en el espacio de direccionamiento en memoria asignado al cliente. Cuando el cliente termina el programa, el objeto agendaTelefonica desaparece, perdiéndose los contactos.

Uso remoto de la agenda

Para conseguir un sistema tolerante a fallos se quiere implementar el servicio de agendas en un nodo distinto del nodo del cliente, de tal forma que exista un proceso servidor que soporte la clase Agenda y el objeto

agendaTelefonica y que atienda las peticiones que provengan de cliente.

Además, se quiere seguir usando el mismo (exacto) fichero cliente que en el apartado del acceso local, para no perder la transparencia de acceso. Pero si la clase Agenda, tal y como está definida en el apartado local, va a residir ahora en el servidor, entonces es necesario que en el lado del cliente se cree otra clase Agenda que se encargue de redireccionar las peticiones al servidor.

Así, la clase Agenda en el lado del cliente actúa como proxy de la clase Agenda en el servidor. El proxy de Agenda en el cliente se encargará de establecer las conexiones con el servidor para soportar la creación (new) como la modificación del contenido de la agenda (asociar y obtener) cuando se invoquen desde el main del cliente.

Organización de ficheros

Para facilitar el desarrollo de la agenda remota, cree dos carpetas “cliente” y “servidor”. Copie en cada una los ficheros que se indican a continuación:

Carpeta servidor:

- IRepositorio
- Agenda (con la implementación del apartado local).
- Servidor *

Carpeta cliente:

- IRepositorio
- ClienteAgenda
- Agenda* (con la implementación de los métodos vacíos y la tabla hash borrada). Esta clase ahora es el proxy de Agenda.

Los ficheros marcados con * son tarea de alumno.

Construcción del cliente. Proxy de Agenda

Puesto que el código main del cliente no se modifica, ni tampoco la interfaz del servicio, la construcción del cliente necesita solo implementar el proxy de Agenda.
El proxy de Agenda implementa la interfaz IRepositorio, pero ahora el constructor y los métodos asociar y obtener deben reescribirse para encaminar las peticiones al servidor. En el cliente, el constructor de la clase **Agenda** y los métodos **asociar** y **obtene**r crearán cada uno de ellos un socket TCP sobre el que pedirán la operación correspondiente en el servidor, es decir actuarán como proxies o representantes del servicio remoto en el cliente. El formato de los mensajes de petición y de respuesta de la operación asociar se describe a continuación.

Formato de mensajes del servicio de Agenda

Para que el proxy de Agenda y el servidor se entiendan, es necesario diseñar un protocolo de servicio. Por tanto, cada operación a solicitar tendrá un código de operación de tipo entero como se indica a continuación:

Código de operación	**Operación**
1	Crear objeto de Agenda
2	Asociar
3	Obtener
Integer	

Como ejemplo de formato, cuando en el main, el cliente invoca el método agendaTelefonica.asociar("Juan", 66756677), se llama al método asociar del proxy, el cual genera internamente el siguiente mensaje de petición para enviarlo al servidor:

Asociar. Mensaje de Petición		
código operación	clave	valor
2	"Juan"	66756677
Integer	String	Integer

Asociar. Mensaje de Respuesta
código resultado
true
Boolean

El envío del mensaje de petición se hará escribiendo cada campo del mensaje en el socket, usando los métodos de clase DataOutputStream, según el tipo de campo a escribir. (Las tiras de caracteres se escriben con el método writeUTF).
La lectura del mensaje de respuesta se realiza usando los métodos de la clase DataInputStream. Si se devuelve true, es que asociar se ha realizado con éxito, y se puede retornar al main del cliente.

Construcción del servidor

Es tarea del alumno escribir un servidor que atienda peticiones sobre objetos de tipo Agenda. Para ello creará un socket de servidor serverSocket, por el que escuchará las solicitudes de conexión. Cada aceptación de conexión devuelve un socket de tipo cliente que se usará para atender la petición correspondiente. Esta petición seguirá el formato del mensaje definido por el alumno en el protocolo de servicio de Agenda.

El servidor según la petición solicitada, creará un objeto de tipo Agenda para mantener realmente los contactos, o invocará los métodos asociar u

obtener sobre el objeto agendaTelefonica indicado. El servidor devuelve el resultado obtenido por la invocación de estos métodos por el socket del cliente.

Cuando el método del proxy de Agenda recibe el resultado proveniente del servidor, se cierra la conexión TCP y se devuelve el resultado al método main del cliente.

Se pide:

1.a) Escriba los diferentes tipos de mensajes a intercambiar y su formato (campos y tipo de los campos) para las operaciones de crear agenda y obtener, de forma similar a cómo se ha definido en asociar.
1.b) El código de los métodos del proxy de la clase Agenda para el lado del cliente.
1.c) El código del **servidor monohilo**, de tal forma que el servidor cree el objeto agendaTelefonica y los contactos del cliente se almacenen en dicho objeto remoto.