Programación de Objetos Distribuidos
Trabajo Práctico 05

Ejercicio 1

Se quiere implementar un servicio de crowdfunding. Este servicio tiene 2 tipos de clientes:

- El *initiator* de un proyecto quiere realizarlo pero no tiene el dinero para ello, por lo cual utiliza el **CrowdfundingInitiatiorService** para crear un proyecto y pedir el apoyo de los clientes *backers*. Entonces crea el proyecto con: un nombre, un monto entero (*goal*) y una lista de premios (*rewards*) donde cada premio tiene un nombre y un valor entero (aporte). El proyecto entonces recibe el aporte de los backers y si llega al monto meta se confirma y se otorgan los premios. Por otro lado el *initiator* puede decidir cancelar el proyecto (si el mismo no está confirmado aún). Si se cancela el proyecto se devuelven los aportes realizados.
- El *backer* utiliza el **CrowdfundingBackerService** para ver el listado de los proyectos a los que puede aportar (no cancelados) y realizar el aporte (*pledge*). El método de aporte es asincrónico y las respuestas del mismo se envían mediante a la interfaz **CrowdfundingBackerResponseHandler** para indicar si el aporte se acepta y está pendiente de confirmación el proyecto (*pledgeReceived*) ó, si se acepta y se otorga el premio (*prizeGranted*) ó, si se rechaza por alguna razón (*errorOnSupport*). La misma interfaz se usa si se requiere para notificarle al backer que se le devuelve el aporte (*pledgeReturned*), porque el proyecto se canceló, o que se le otorga un premio que estaba pendiente porque el proyecto se confirmó (*prizeGranted*).

Aclaraciones:

- Un proyecto solo deja de recibir aportes si está cancelado.
- Se garantiza (por lo cual no es necesario validar) que en la creación de un proyecto todos los premios tienen montos diferentes. Aunque no se garantiza que venga ordenado.
- El aporte debe ser igual a uno de los montos de los premios si no se rechaza.
- Un cliente puede realizar diferentes aportes al mismo proyecto para obtener varios premios (o varias veces el mismo premio).
- Todos los errores al realizar un aporte se reportan por medio de **CrowdfundingClientHandler#errorOnSupport** no por excepciones.

El servicio se basa en las siguientes interfaces:

```
/** servicio para el manejo de proyectos a través de un servicio RMI sincrónico */
public interface CrowdfundingInitiatiorService extends Remote {

  /** @return true si pudo crear el proyecto, false si no */
  boolean createProject(String projectName, int goal, List<Reward> prizes) throws
RemoteException;
```

```
  /** @return true si el proyecto pudo ser cancelado */
  boolean cancelProject(String projectName) throws RemoteException;
}
```

```
/** Servicio para poder dar apoyo a proyectos utilizando RMI */
public interface CrowdfundingBackerService extends Remote {

    /** @return la lista de proyectos a los que se puede aportar */
    public List<Project> listProjects() throws RemoteException;

    /**
     * Método asíncrono, las respuestas se envían mediante el parámetro
     * {@link CrowdfundingBackerResponseHandler}
     */
    public void pledge(Project project, int amount, CrowdfundingBackerResponseHandler
supporter)
            throws RemoteException;
}
```

```
/** servicio de respuesta para un backer de un proyecto */
public interface CrowdfundingBackerResponseHandler extends Remote {

    /**
     * se invoca con cualquier error ocurrido durante el llamado a
     * {@link CrowdfundingBackerService#pledge}
     */
    public void errorOnSupport(String projectName, int amount, String
errorMessage) throws RemoteException;

    /**
     * se invoca al llamar a {@link CrowdfundingBackerService#pledge} si se
     * aceptó el aporte pero aún no se otorga el premio porque el proyecto no
     * está aún confirmado.
     */
    public void pledgeReceived(String projectName, int amount) throws
RemoteException;

    /**
     * Se invoca cuando un premio es otorgado ya sea por aportar a un proyecto
     * confirmado o porque se confirmó el proyecto.
     */
    public void prizeGranted(String projectName, int amount, String reward)
throws RemoteException;

    /**
     * Se invoca cuando un proyecto fue cancelado para notificar la devolución
     * del aporte.
     */
    public void pledgeReturned(String projectName, int amount) throws
RemoteException;
}
```

Se requiere que se implemente un solo servant que implemente ambas interfaces de servicio remotas (**CrowdfundingInitiatiorService** y **CrowdfundingBackerService**). Utilizando RMI con activatable y para que funcione en un ambiente concurrente y Thread Safe. Además, se requiere serializar en un archivo el listado de los proyectos para garantizar persistencia del catálogo (sin incluir premios o aportes).