Programación de Objetos Distribuidos
Soluciones Trabajo Práctico 05

Ejercicio 1

```
public class Project implements Serializable {

	private static final long serialVersionUID = -6976014559806070824L;

	private String name;
	private int goal;
	private List<Reward> rewards;
	private Status status;

	public Project(String name, int goal, List<Reward> rewards) {
		this.name = name;
		this.goal = goal;
		this.rewards = rewards;
		this.status = Status.UNCONFIRMED;
	}

	public String getName() {
		return name;
	}

	public int getGoal() {
		return goal;
	}

	public List<Reward> getRewards() {
		return rewards;
	}

	public Status getStatus() {
		return status;
	}

	public void setStatus(Status status) {
		this.status = status;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((name == null) ? 0 : name.hashCode());
		return result;
	}
```

```
	@Override
	public boolean equals(Object obj) {
		return this == obj || Optional.ofNullable(obj).filter(o -> o instanceof
Project).map(o -> (Project) o)
				.filter(r -> Objects.equals(this.name,
r.name)).isPresent();
	}

	@Override
	public String toString() {
		return getName() + " # (" + getStatus() + ")";
	}
}
```

```
public class Reward implements Serializable {

	private static final long serialVersionUID = 4113516841478782277L;

	private String name;
	private int amount;

	public Reward(String name, int amount) {
		this.name = name;
		this.amount = amount;
	}

	public String getName() {
		return name;
	}

	public int getAmount() {
		return amount;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((name == null) ? 0 : name.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		return this == obj || Optional.ofNullable(obj).filter(o -> o instanceof
Reward).map(o -> (Reward) o)
				.filter(r -> Objects.equals(this.name,
r.name)).isPresent();
	}

}
```

```
public enum Status {
	UNCONFIRMED, CONFIRMED, CANCELLED;
}
```

```
public class Pledge implements Serializable {

	private static final long serialVersionUID = 68321592870468416866L;

	private CrowdfundingBackerResponseHandler handler;
	private int amount;
	private Reward reward;

	public Pledge(CrowdfundingBackerResponseHandler handler, int amount,
                                                                Reward reward) {
		this.handler = handler;
		this.amount = amount;
		this.reward = reward;
	}

	public CrowdfundingBackerResponseHandler getHandler() {
		return handler;
	}

	public int getAmount() {
		return amount;
	}

	public Reward getReward() {
		return reward;
	}

}
```

```
/**
 * Clase que sirve para serializar a la variable de instancia que contiene a los
 * proyectos. De esta forma, incluso habiéndose "caído" el RMID, al instanciar
 * un nuevo stub, se pueden recuperar los proyectos que existieron en instancias
 * anteriores. A modo de ejemplo sólo se persisten los proyectos sin los
 * aportes. Podría persistirse toda la información.
 */
public class CrowdfundingData implements Serializable {

	private static final long serialVersionUID = 47635577394873624456L;

	private Map<String, Project> projects;

	public void setProjects(Map<String, Project> projects) {
		this.projects = projects;
	}

	public Map<String, Project> getProjects() {
```

```
                return projects;
        }

}
```

```
public class CrowdfundingBackerResponseHandlerImpl implements
                                CrowdfundingBackerResponseHandler {

        public CrowdfundingBackerResponseHandlerImpl() throws RemoteException {
                 UnicastRemoteObject.exportObject(this, 0);
        }

        @Override
        public void errorOnSupport(String projectName, int amount, String errorMessage)
                                                                        throws RemoteException {
                System.out.println("Error: " + errorMessage + ", al aportar $" + amount +
" al proyecto " + projectName);
        }

        @Override
        public void pledgeReceived(String projectName, int amount)
                                                                throws RemoteException   {
                System.out.println("Se recibieron $" + amount + " para el proyecto "
                                                                        + projectName);
        }

        @Override
        public void prizeGranted(String projectName, int amount, String reward)
                                                                throws RemoteException {
                System.out.println("Se confirmaron $" + amount + " para el proyecto "
                                                                          + projectName);
                System.out.println("Obtuviste el premio " + reward);
        }

        @Override
        public void pledgeReturned(String projectName, int amount)
                                                                throws RemoteException {
                System.out.println("Se retornaron $" + amount + " para el proyecto "
                                                                        + projectName);
        }

}
```

```
public class Servant implements CrowdfundingInitiatiorService, CrowdfundingBackerService
{

        private Map<String, Project> projects;
        private Map<String, List<Pledge>> pledges;

        private CrowdfundingData crowdfundingData;
        private File storage;
```

```
	private static final Logger logger = Logger.getLogger(Servant.class.getName());

	public Servant(ActivationID id, MarshalledObject<File> data)
					throws ClassNotFoundException, IOException {
		storage = data.get();
		if (storage.exists()) {
			loadData(); //Carga del archivo con el listado de los proyectos.
		} else {
			logger.log(Level.INFO,
				"No existe un archivo con el listado de los proyectos.");
			crowdfundingData = new CrowdfundingData();
			projects = new HashMap<>();
			pledges = new HashMap<>();
		}
		Activatable.exportObject(this, id, 0);
	}

	@Override
	public List<Project> listProjects() throws RemoteException {
		synchronized (projects) {
			return new ArrayList<>(projects.values());
		}
	}

	@Override
	public void pledge(Project project, int amount, CrowdfundingBackerResponseHandler
supporter)
			throws RemoteException {
		if (project == null || supporter == null) {
			throw new IllegalArgumentException();
		}
		try {
			pledgeAux(project, amount, supporter);
		} catch (InexistentProjectException ex) {
			supporter.errorOnSupport(project.getName(), amount,
ex.getLocalizedMessage());
		} catch (InvalidPledgeAmountException ex) {
			supporter.errorOnSupport(project.getName(), amount,
ex.getLocalizedMessage());
		}
	}

	public void pledgeAux(Project project, int amount,
				CrowdfundingBackerResponseHandler supporter)
						throws RemoteException {
		if (amount <= 0) {
			throw new InvalidPledgeAmountException();
		}
		String projectName = project.getName();
		synchronized (projects) {
			if (!projects.containsKey(project.getName())) {
				throw new InexistentProjectException();
```

```
                        }
                }
                Project destinationProject = projects.get(projectName);
                Reward reward = destinationProject.getRewards().stream().filter(r ->
r.getAmount() == amount).findFirst()
                                .orElseThrow(InvalidPledgeAmountException::new);
                Pledge pledge = new Pledge(supporter, amount, reward);
                List<Pledge> projectPledges;
                synchronized (destinationProject) {
                        switch (destinationProject.getStatus()) {
                        case UNCONFIRMED:
                                projectPledges = pledges.get(projectName);
                                projectPledges.add(pledge);
                                pledges.put(projectName, projectPledges);
                                if (projectHasReachedGoal(destinationProject,
projectPledges)) {
                                        destinationProject.setStatus(Status.CONFIRMED);
                                        logger.log(Level.INFO, "El proyecto " + projectName
+ " ha sido confirmado.");
                                        notifyConfirmation(projectName, projectPledges,
reward);
                                }
                                try {
                                        updateData(); //Actualización del archivo con el
listado de proyectos.
                                } catch (Exception ex) {
                                        logger.log(Level.WARNING, ex.getMessage(), ex);
                                }
                                break;
                        case CONFIRMED:
                                projectPledges = pledges.get(projectName);
                                projectPledges.add(pledge);
                                pledges.put(projectName, projectPledges);
                                supporter.prizeGranted(projectName, amount,
reward.getName());
                                break;
                        case CANCELLED:
                                supporter.pledgeReturned(projectName, amount);
                                break;
                        default:
                                throw new IllegalArgumentException();
                        }
                }
        }

        @Override
        public boolean createProject(String projectName, int goal, List<Reward> prizes)
throws RemoteException {
                if (projectName == null || projectName.isEmpty() || goal <= 0 || prizes ==
null) {
                        throw new IllegalArgumentException();
                }
                if (prizes.isEmpty()) {
```

```
                        throw new ProjectWithoutPrizesException();
                }
                synchronized (projects) {
                        if (projects.containsKey(projectName)) {
                                throw new AlreadyExistsProjectException();
                        }
                }
                synchronized (this) {
                        Project project = new Project(projectName, goal, prizes);
                        projects.put(projectName, project);
                        pledges.put(projectName, new ArrayList<>());
                        logger.log(Level.INFO, "El proyecto " + projectName + " ha sido
creado.");
                        try {
                                updateData(); //Actualización del archivo con el listado de
proyectos.
                        } catch (Exception ex) {
                                logger.log(Level.WARNING, ex.getMessage(), ex);
                        }
                }
                return true;
        }

        @Override
        public boolean cancelProject(String projectName) throws RemoteException {
                if (projectName == null || projectName.isEmpty()) {
                        throw new IllegalArgumentException();
                }
                synchronized (projects) {
                        if (!projects.containsKey(projectName)) {
                                throw new InexistentProjectException();
                        }
                }
                Project project = projects.get(projectName);
                synchronized (project) {
                        if (project.getStatus().equals(Status.CANCELLED)) {
                                throw new AlreadyCancelledProjectException();
                        }
                        project.setStatus(Status.CANCELLED);
                        projects.put(projectName, project);
                }
                logger.log(Level.INFO, "El proyecto " + projectName + " ha sido
cancelado.");
                notifyCancellation(projectName, pledges.get(projectName));
                try {
                        updateData(); //Actualización del archivo con el listado de
proyectos.
                } catch (Exception ex) {
                        logger.log(Level.WARNING, ex.getMessage(), ex);
                }
                return true;
        }
```

```
	private void notifyCancellation(String projectName, List<Pledge> pledges) {
		pledges.parallelStream().forEach(p -> {
			try {
				p.getHandler().pledgeReturned(projectName, p.getAmount());
			} catch (RemoteException ex) {
				logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
			}
		});
	}

	private void notifyConfirmation(String projectName, List<Pledge> pledges, Reward
reward) {
		pledges.parallelStream().forEach(p -> {
			try {
				p.getHandler().prizeGranted(projectName, p.getAmount(),
reward.getName());
			} catch (RemoteException ex) {
				logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
			}
		});
	}

	private boolean projectHasReachedGoal(Project project, List<Pledge> pledges) {
		return pledges.parallelStream().mapToInt(p -> p.getAmount()).sum() >=
project.getGoal();
	}

	private synchronized void updateData() throws FileNotFoundException, IOException
{
		crowdfundingData.setProjects(projects);
		ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(storage));
		oos.writeObject(crowdfundingData);
		oos.close();
		logger.log(Level.INFO, "Se actualizó el archivo " +
storage.getAbsolutePath() + " con el listado de proyectos.");
	}

	private synchronized void loadData() throws FileNotFoundException, IOException,
ClassNotFoundException {
		logger.log(Level.INFO, "Se encontró el archivo " +
storage.getAbsolutePath() + " con el listado de los proyectos");
		ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(storage));
		crowdfundingData = (CrowdfundingData) ois.readObject();
		projects = crowdfundingData.getProjects();
		ois.close();
	}

}
```

```
public class Server {

	public static void main(String[] args) throws Exception {
		/**
		 * 1) Instalar un SecurityManager para la JVM.
		 */
		if (System.getSecurityManager() == null) {
			System.setSecurityManager(new SecurityManager());
		}

		/**
		 * 2) Registar un ActivationGroup en la instancia Activator que lanzó el
		 * RMID, o sea, RMID debe estar levantado.
		 */
		ActivationGroupID anAGroupID = registrarActivationGroup();

		/**
		 * 3) Generar un ActivationDesc con la info de creación para el objeto
		 * remoto, ActivationGroupID que lo manejara, nombre de la clase,
		 * codebase (classpath) y serialización.
		 */
		ActivationDesc remoteServiceDesc = new ActivationDesc(anAGroupID,
Servant.class.getCanonicalName(),
				Server.class.getClassLoader().getResource(".").toString(),
				new MarshalledObject<File>(new
File(Paths.get("").toAbsolutePath() + "/" + "projects.ser")));

		/**
		 * 4) Dicho ActivationDesc del servant se lo debe pasar a la única
		 * instancia Activator del RMID que está levantado para que cuando éste
		 * precise poner activo al servant se lo pase al ActivationGroup
		 * correspondiente. Para ello se registra a dicho ActivationDesc y se
		 * genera en esta sentencia el stub del servant (ya que no habrá
		 * creación explícita de la instancia servant codificada)
		 */
		Remote remoteServiceStub = Activatable.register(remoteServiceDesc);
		System.out.println("Activation descriptor registered: " +
remoteServiceStub);

		/**
		 * 5) Teniendo el stub ya podemos realizar el binding.
		 */
		Registry registry = LocateRegistry.getRegistry("localhost", 1099);
		registry.rebind(CrowdfundingInitiatiorService.class.getName(),
remoteServiceStub);
		registry.rebind(CrowdfundingBackerService.class.getName(),
remoteServiceStub);
		System.out.println("Service bound");

		/**
		 * 6) Salimos de la aplicación. No somos responsables de mantener
		 * objetos.
		 */
```

```
                System.exit(0);
        }

        public static ActivationGroupID registrarActivationGroup() throws
ActivationException, RemoteException {
                Properties pList = new Properties();
                pList.put("java.security.policy",
Server.class.getClassLoader().getResource("java.policy").toString());
                ActivationGroupDesc groupDesc = new ActivationGroupDesc(pList, null);
                System.out.println("groupDesc = " + groupDesc);
                ActivationGroupID groupID =
ActivationGroup.getSystem().registerGroup(groupDesc);
                System.out.println("groupID = " + groupID);
                return groupID;
        }

}
```

```
public class Client {

        private static final Logger logger = Logger.getLogger("Client");

        public static void main(String[] args)
                        throws MalformedURLException, RemoteException, NotBoundException,
InterruptedException {
                CrowdfundingInitiatiorService crowdfundingInitiatiorService =
(CrowdfundingInitiatiorService) Naming
                                .lookup("//localhost:1099/" +
CrowdfundingInitiatiorService.class.getName());
                CrowdfundingBackerService crowdfundingBackerService =
(CrowdfundingBackerService) Naming
                                .lookup("//localhost:1099/" +
CrowdfundingBackerService.class.getName());

                /**
                 * 1. Creación de un proyecto
                 */
                try {
                        List<Reward> project1Rewards = new ArrayList<>();
                        project1Rewards.add(new Reward("Premio #1", 250));
                        project1Rewards.add(new Reward("Premio #2", 750));

System.out.println(crowdfundingInitiatiorService.createProject("Proyecto #1", 1000,
project1Rewards));
                } catch (RemoteException ex) {
                        logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
                }

                /**
                 * 2. Listado de proyectos
                 */
                List<Project> projects = null;
```

```
		try {
			projects = crowdfundingBackerService.listProjects();
			projects.stream().forEach(p -> System.out.println(p));
		} catch (RemoteException ex) {
			logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
		}

		/**
		 * 3. Creación de un proyecto con un nombre ya existente
		 */
		try {
			List<Reward> project2Rewards = new ArrayList<>();
			project2Rewards.add(new Reward("Premio #3", 1000));
			crowdfundingInitiatiorService.createProject("Proyecto #1", 1000,
project2Rewards);
		} catch (RemoteException ex) {
			logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
		}

		/**
		 * 4. Aporte con un monto distinto a los montos de los premios del
		 * proyecto
		 */
		Project project1 = projects.get(0);
		try {
			CrowdfundingBackerResponseHandler handler = new
CrowdfundingBackerResponseHandlerImpl();
			crowdfundingBackerService.pledge(project1, 300, handler);
		} catch (RemoteException ex) {
			logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
		}

		/**
		 * 5. Aportes a un proyecto
		 */
		try {
			CrowdfundingBackerResponseHandler handler2 = new
CrowdfundingBackerResponseHandlerImpl();
			crowdfundingBackerService.pledge(project1, 250, handler2);
			CrowdfundingBackerResponseHandler handler3 = new
CrowdfundingBackerResponseHandlerImpl();
			crowdfundingBackerService.pledge(project1, 750, handler3);
		} catch (RemoteException ex) {
			logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
		}

		/**
		 * 6. Cancelar un proyecto
		 */
		try {
			crowdfundingInitiatiorService.cancelProject(project1.getName());
		} catch (RemoteException ex) {
			logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
```

```
		}

		/**
		 * 7. Aportar a un proyecto cancelado
		 */
		try {
			CrowdfundingBackerResponseHandler handler4 = new
CrowdfundingBackerResponseHandlerImpl();
			crowdfundingBackerService.pledge(project1, 750, handler4);
		} catch (RemoteException ex) {
			logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
		}

		/**
		 * 8. Cancelar un proyecto inexistente
		 */
		try {
			crowdfundingInitiatiorService.cancelProject("Proyecto Uno");
		} catch (RemoteException ex) {
			logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
		}

		/**
		 * 9. Cancelar un proyecto ya cancelado
		 */
		try {
			crowdfundingInitiatiorService.cancelProject(project1.getName());
		} catch (RemoteException ex) {
			logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
		}

		/**
		 * 10. Aportar sin un handler
		 */
		try {
			crowdfundingBackerService.pledge(project1, 750, null);
		} catch (RemoteException ex) {
			logger.log(Level.SEVERE, ex.getLocalizedMessage(), ex);
		}
	}

}
```