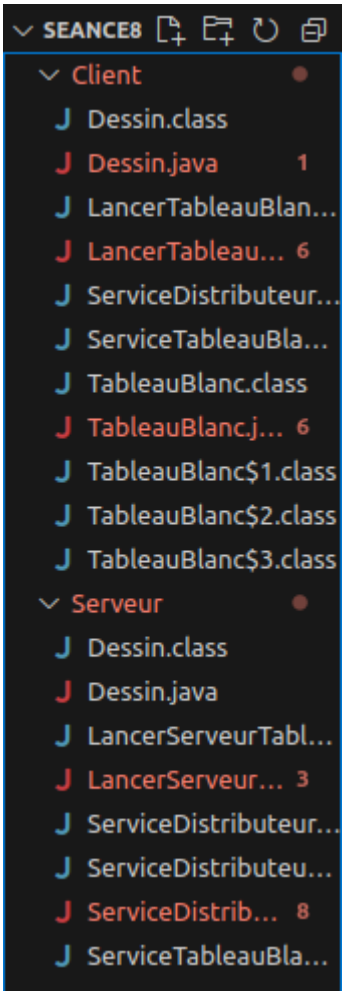
Séance 8 - Un tableau blanc collaboratif

Code première partie :

Insérez dans votre compte-rendu les lignes modifiées du programme statique et l'arborescence de votre répertoire avec les fichiers contenus



COTE SERVICE CENTRAL :

```
import java.net.ConnectException;
import java.rmi.RemoteException;
import java.util.ArrayList;
public class ServiceDistributeurTableauBlanc implements
ServiceDistributeur {
   private ArrayList<ServiceTableauBlanc> tableaux;
   private ArrayList<Dessin> dessins; // Pour stocker tous les dessins
ayant été réalisés jusqu'à là, pour renvoyer aux nouveaux clients.

   public ServiceDistributeurTableauBlanc(){
```

```
        this.tableaux = new ArrayList<>();
        this.dessins = new ArrayList<>();
    }

    public void enregistrerClient(ServiceTableauBlanc c) throws
RemoteException{
        this.tableaux.add(c);
        // Avec cette boucle, lorsqu'un nouveau client s'ajoute à la
liste du serveur central, alors le service central lui retourne tous
les points qu'il a déjà enregistrés au précédent.
        for(Dessin d : this.dessins){
            c.afficherMessage(d);
        }
    }

    public void distribuerMessage(Dessin d) throws RemoteException{
        // A chaque fois que le serveur reçoit un dessin, alors il en
informe tous ses clients
        // Si le client ne répond pas, alors il le supprime de la liste

        this.dessins.add(d);

        for(ServiceTableauBlanc s : this.tableaux){
            try{
                s.afficherMessage(d);
            }catch(RemoteException e){ // si jamais une exception de
type "ConnectException" est déclenchée lorsqu'un client est déconnecté
mais toujours présent dans la liste
            // alors on l'enlève de la liste, comme ça, le serveur ne se
bloque pas.
                this.tableaux.remove(s);
                e.printStackTrace();
            }

        }
    }
}
```

Au niveau du main, lancement classique comme n’importe quel service.

COTE SERVICE CLIENT :

```
import java.awt.Color;
import java.io.Serializable;

public class Dessin implements Serializable{
   public int x, y;
   public Color c;

   public String toString(){
       return "Dessin : ("+this.x+", "+this.y+") couleur = "+this.c ;
   }
}
import java.rmi.server.UnicastRemoteObject;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.StubNotFoundException;

public class LancerTableauBlanc{
   public static void main (String args[]) {

       try{

          // on récupère d'abord le service, c'est à dire
          Registry reg = LocateRegistry.getRegistry(args[0],
Integer.parseInt(args[1]));
          ServiceDistributeur tBS = (ServiceDistributeur)
reg.lookup("distributeur");

          TableauBlanc tB = new TableauBlanc(tBS); // On crée notre
tableau *

          ServiceTableauBlanc rdTB = (ServiceTableauBlanc)
UnicastRemoteObject.exportObject(tB, 0);
           // j'exporte ma liste de contacts
           // et la passe en param de nouveauService
           tBS.enregistrerClient(rdTB);

           // puis je dois constater qu'un contact a été ajouté à ma
liste

       }catch(NotBoundException e){
```

```
            e.printStackTrace();
        }catch(StubNotFoundException e){
            System.err.println("Stub not of correct class");
        }catch(RemoteException e){
            e.printStackTrace();
        }


    }
}
```

La classe TableauBlanc, possède donc un nouvel attribut correspondant au serveur central

```
public class TableauBlanc implements ServiceTableauBlanc{
    JPanel     canevas;
    Color      couleur_courante    = Color.black;
    Color[]    couleurs_possibles = { Color.black, Color.white, Color.
    ServiceDistributeur tB;

    public void afficherMessage(Dessin d) { // Dessiner un dessin sur
        Graphics g = canevas.getGraphics();
        g.setColor( d.c );
        g.fillRect( d.x - 5, d.y - 5, 10, 10 );
    }

    public JPanel CreerPaletteCouleur() { // Pallette des couleurs po
```

Et cet attribut est utilisé lorsqu'un evenement de la souris est capté par le listener.

```
        canevas.addMouseMotionListener( new MouseMotionListener() {  // Gestion
            public void mouseDragged(MouseEvent arg0) {
                Dessin d = new Dessin();
                d.x = arg0.getX();
                d.y = arg0.getY();
                d.c = couleur_courante;
                try{
                    tB.distribuerMessage(d); // on envoie au serveur central les
                }catch(RemoteException e){
                    e.printStackTrace();
                }

                afficherMessage(d);                                  // Voir la
            }
            public void mouseMoved(MouseEvent arg0) { }
        });
    }
}
```

Car grâce à lui, on peut notifier le serveur central des modifications faites par le client.

Quelques améliorations pour finir

- **Comment gérer proprement la fin d'exécution d'un participant ?**
  On peut donc faire en sorte que le service central comporte une liste de clients. A chaque fois qu'un client fait une modification, il l'envoie au service central qui lui en informe tous les autres clients. Donc à ce moment, lorsque le service central envoie des requêtes séquentiellement à chaque client, il suffit qu'il regarde s'il obtient une réponse du client. Si ce n'est pas le cas, ça veut dire que le client n'est plus connecté, et il suffit donc de le supprimer de sa liste de clients.
- **Comment faire en sorte que le dessin d'un nouveau venu soit identique à ceux des participants déjà présents ?**
  Pour ça, dès le lancement du serveur, il suffit que le service central possède une liste contenant tous les dessins effectués. C'est à dire qu'à chaque fois qu'il reçoit une modification effectuée par un client, alors il l'ajoute à sa liste de dessins. Lorsqu'un nouveau client apparaît après 30 minutes, le service central, dans sa méthode *enregistrerClient* a juste besoin de lui retourner tous les dessins qu'il a enregistrés jusqu'à présent, grâce à la méthode : *afficherMessage(Dessin d)* du nouveau client.

```
public ServiceDistributeurTableauBlanc(){
    this.tableaux = new ArrayList<>();
    this.dessins = new ArrayList<>();
}

public void enregistrerClient(ServiceTableauBlanc c) throws RemoteException{
    this.tableaux.add(c);
    // Avec cette boucle, lorsqu'un nouveau client s'ajoute à la liste du se
    for(Dessin d : this.dessins){
        c.afficherMessage(d);
    }
}

public void distribuerMessage(Dessin d) throws RemoteException{
    // A chaque fois que le serveur reçoit un dessin, alors il en informe to
    // Si le client ne répond pas, alors il le supprime de la liste

    this.dessins.add(d);

    for(ServiceTableauBlanc s : this.tableaux){
        try{
            s.afficherMessage(d);
        }catch(RemoteException e){ // si jamais une exception de type "Conne
        // alors on l'enlève de la liste, comme ça, le serveur ne se bloque
            this.tableaux.remove(s);
            e.printStackTrace();
        }

    }
```