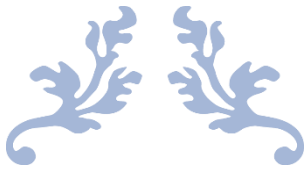


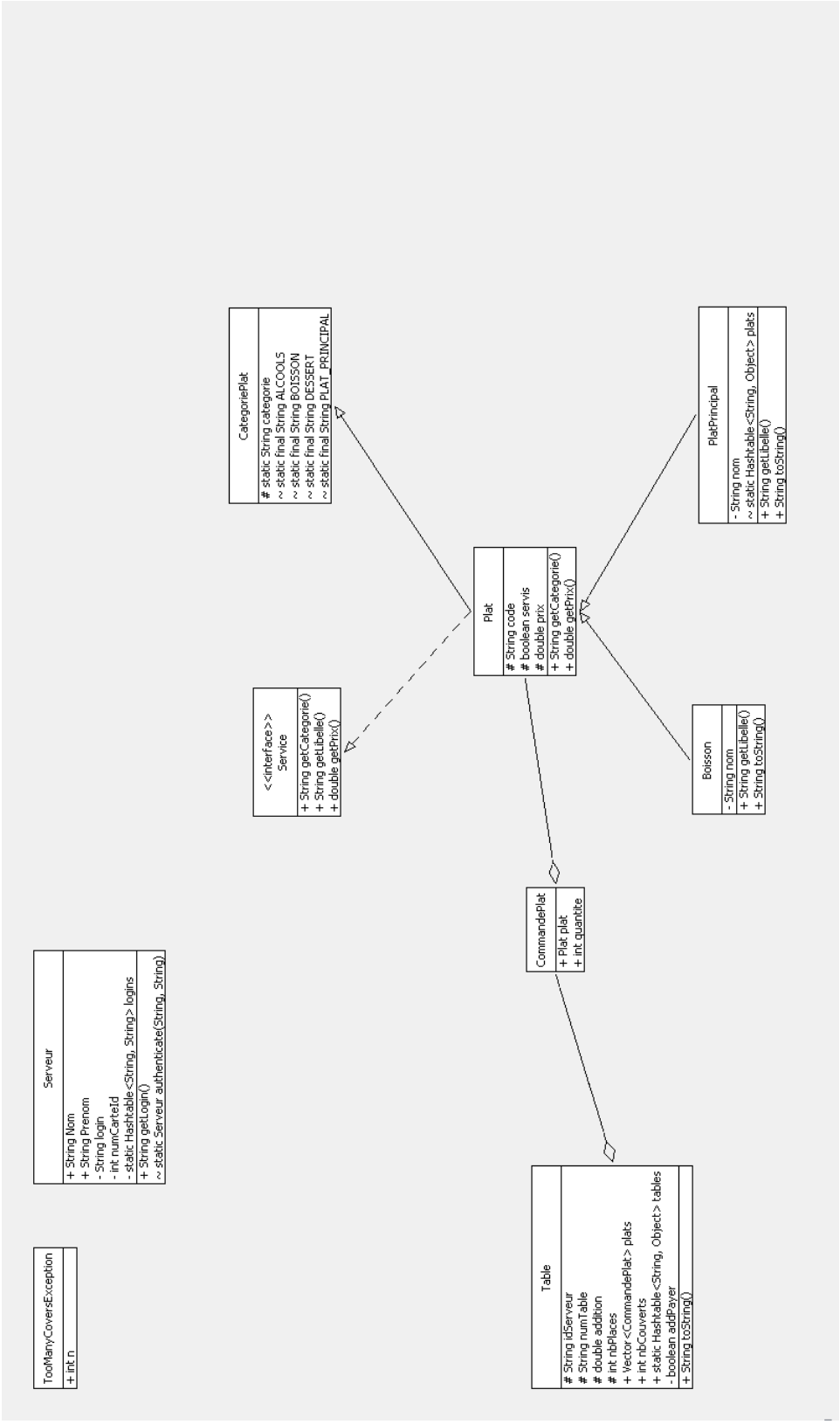
LE GOURMET AUDACIEUX

Laboratoire Java – M. MADANI

03 MAI 2018

DANIEL GARCIA LECLOUX – ALESSANDRO ALOISIO
2222

Evaluation n°1 : Schéma UML des classes utilisées dans l'application



Explication et code correspondant à un changement de table dans l'application

```
private void tableBoxItemStateChanged(java.awt.event.ItemEvent evt) {//GEN-
FIRST:event_tableBoxItemStateChanged

        boolean continuer = true;

        // afficher le nombre de places dispo de la table
        if(evt.getStateChange() == 1){

            // Demander si on change de serveur
            if(tableCourant >= 0) {

                final JTextField userNameField = new JTextField(10);

                userNameField.setText(serveur.getLogin());
                userNameField.enable(false);

                JPanel pane = new JPanel(new GridBagLayout());
                GridBagConstraints gbc = new GridBagConstraints(0, 0, 1, 1, 1.0, 1.0,
                        GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL,
                        new Insets(2, 2, 2, 2), 0, 0);
                pane.add(new JLabel("Serveur en cours :"), gbc);

                gbc.gridx = 1;
                gbc.gridy = 0;
                gbc.anchor = GridBagConstraints.EAST;
                pane.add(userNameField, gbc);

                int reply = JOptionPane.showConfirmDialog(null, pane, "Restaurant \"Le
gourmet audacieux\"",
                        JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);

                if (reply == JOptionPane.OK_OPTION) {
                    // On continue avec
                    String userName = userNameField.getText();
                }else{
                    continuer = false;
                    loginField.setText("");
                    passwordField.setText("");

                    this.setVisible(true);

                    tempTable = new Table(evt.getItem().toString(),
Integer.parseInt(placesLabel.getText()));
                }

            }

            // si table occupée
            int i = 0;
            for(Table item : table){
                // Si la table est occupée et que le serveur est le même qui a pris la
commande
                if(item.numTable.equals(evt.getItem()) &&
serveur.getLogin().equals(table.elementAt(i).idServeur)){
                    System.out.println("Table occupee : " + item + ", index = "+i);

                    continuer = false;
```

```
                    tableCourant = i;
                    tableCourantCode = item.numTable;


couvertsLabel.setText(Integer.toString(table.get(tableCourant).nbCouverts));
                    platsAttente.setModel(createListData(false));
                    listPlats.setModel(createListData(true));

                    break;
                }
                i++;
            }

            // si table disponible
            if(Table.tables.containsKey(evt.getItem()) && continuer) {


placesLabel.setText(Table.tables.get((String)evt.getItem()).toString());

                table.add(nbTableOccupe, new Table(evt.getItem().toString(),
Integer.parseInt(placesLabel.getText())));
                tableCourant = nbTableOccupe;
                tableCourantCode = (String)evt.getItem().toString();
                nbTableOccupe++;

                table.get(tableCourant).idServeur = serveur.getLogin();


couvertsLabel.setText(Integer.toString(table.get(tableCourant).nbCouverts));
                platsAttente.setModel(createListData(false));
                listPlats.setModel(createListData(true));

            }
        }
    }
```

On vérifie qu'il y a bien une table occupée, si c'est le cas affiche une pop up pour demander si on garde le même serveur ou si on change.

Si on veut changer de serveur, on affiche la fenêtre de connexion et on sauvegarde la table sélectionnée dans une variable temporaire pour ensuite la récupérer dans la partie de connexion suivante :

```
else if(tempTable != null) {

            boolean continuer = true;

            placesLabel.setText(Table.tables.get(tempTable.numTable).toString());

            // si table occupée
            int i = 0;
            for(Table item : table){
                // Si la table est occupée et que le serveur est le même qui a pris la
commande
                if(item.numTable.equals(tempTable.numTable.toString()) &&
serveur.getLogin().equals(table.elementAt(i).idServeur)){
                    System.out.println("Table occupee : " + item + ", index = "+i);

                    tableCourant = i;
```

```
                    tableCourantCode = item.numTable;

                    continuer = false;
                    break;
                }
                i++;
            }

            if(continuer) {
                table.add(nbTableOccupe, new Table(tempTable.numTable.toString(),
Integer.parseInt(placesLabel.getText())));
                tableCourant = nbTableOccupe;
                tableCourantCode = tempTable.numTable.toString();
                nbTableOccupe++;
                table.get(tableCourant).idServeur = serveur.getLogin();
            }


couvertsLabel.setText(Integer.toString(table.get(tableCourant).nbCouverts));
            platsAttente.setModel(createListData(false));
            listPlats.setModel(createListData(true));

            tempTable = null;

        }
```

Cette partie de connexion permet de vérifier encore une fois si la table est disponible ou déjà occupée par un client.

On vérifie que la table sélectionnée est bien dans le vecteur de tables et que le serveur qui gère la table correspond à celui qui est connecté. TableCourant permet de récupérer les informations de la table.

On met à jour le nombre de couverts, les plats en attente ainsi que les plats déjà servis.

`placesLabel est le nombre maximum de places disponibles sur cette table et couvertsLabel est le nombre de plats principaux commandés.`

Si la table est disponible on réactualise le nombre max de couverts et on ajoute un nouvel objet table dans le vecteur de tables et on y instancie le serveur en cours.

On met à jour le nombre de couverts, les plats en attente ainsi que les plats déjà servis.