

TRAVAIL PRATIQUE 2

Le pool de hockey

En attendant le retour de la LNH, le travail que vous aurez à faire devra représenter certains éléments du fonctionnement d'un « pool » de hockey de la LNH. Un pool de hockey consiste en un choix de joueurs actifs de la Ligue Nationale de Hockey selon certaines règles établies. Chaque but, passe ou blanchissage d'un joueur donne des points au participant qui a choisi ce joueur. Habituellement, un pool dure pour toute la saison de hockey; le nôtre sera ponctuel et consistera au choix des joueurs et aux points présents de ceux-ci.

Les règles du Pool

Voici les principales règles à mettre en œuvre dans ce travail :

- * Le participant doit choisir 11 joueurs : 7 joueurs d'avant (centre et ailiers), 2 défenseurs et 2 gardiens de but
- * Le participant doit respecter un plafond salarial : la somme des salaires de ses 11 joueurs ne doit pas dépasser 40 000 000 \$
- * Les points accordés seront les suivants :

<i>Pour un Avant et un Défenseur</i>	<i>Points accordés dans le pool</i>
Pour un but	1 point
Pour une passe	1 point
<i>Pour un gardien</i>	<i>Points accordés dans le pool</i>
Pour une victoire	1 point
Pour un blanchissage	3 points

Les classes

1. Commencez par établir un schéma des classes dont vous aurez besoin pour représenter les différents joueurs de la LNH et les variables de celles-ci. Vos classes doivent **être organisées de manière à ce qu'un héritage soit présent** (superclasse, sous-classes)

2. Vous devez créer maintenant une classe `ListeJoueurs` dont la seule variable d'instance sera un `Vector`. Ce `Vector` devra contenir environ 20 joueurs de la LNH parmi lesquels les participants devront choisir leurs 11 joueurs.

- * Vous créez donc 20 joueurs à partir des classes que vous avez créées au numéro 1. → servez-vous de statistiques présentes sur www.nhl.com ou la section sports de www.src.ca (il doit s'agir de réels joueurs pour le salaire vous pouvez regarder le site www.nhlnumbers.com)
- * Vous ajoutez ces objets à votre objet `Vector`.

- * Codez également une méthode `rechercherJoueur` qui permettra, à partir du nom du joueur passé en paramètre, de retourner l'objet `Joueur` correspondant à ce nom, s'il existe.

3. Vous devez maintenant coder une classe `Participant`, représentant un participant au pool de hockey. C'est lui qui devra choisir ses joueurs en fonction des règles établies du pool énoncées au début du document.

Rappelez-vous qu'en programmation orientée-objet, mieux vaut réaliser plusieurs méthodes, chacune faisant une action précise, quitte à les regrouper dans une méthode les combinant. Cette procédure vous aidera lors du codage de l'interface graphique.

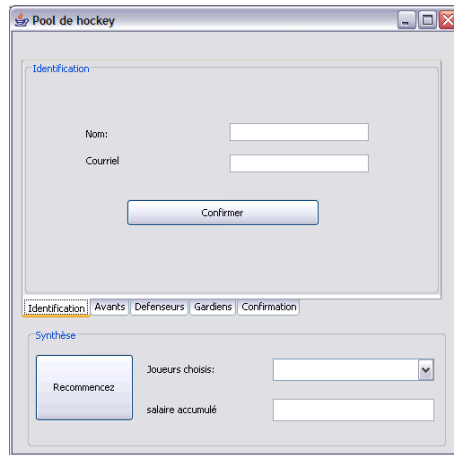
Tenez en compte les faits suivants :

- * le participant sera identifié à l'aide de son nom et de son adresse courriel
- * la classe devra elle aussi contenir un `Vector` contenant les joueurs au fur et à mesure qu'ils sont choisis par le participant
- * Évidemment, outre les méthodes habituelles, la classe devra contenir une méthode permettant de choisir un joueur présent dans la liste de joueurs disponibles (#2) à partir de son nom. Avant de choisir le joueur, vous devrez vérifier 3 points principaux...lesquels ?? pensez-y !
- * Vous devez également produire :
 - une méthode calculant le salaire total des choix du participant
 - une méthode calculant les points de pool attribués aux choix du participant

4. **LE GUI (Graphical User Interface)**

Vous devez maintenant prouver le fonctionnement de votre modèle de classes à l'aide d'une interface graphique qui est partiellement réalisée pour vous.

Pour que la gestion des événements-bouton soit la plus simple possible, j'ai adopté un GUI avec des onglets (`JTabbedPane`) (voir page suivante)



Ce que vous avez faire :

A) Déclarer un objet Participant et un objet ListeJoueurs. Vous pouvez créer l'objet ListeJoueurs dans le constructeur du GUI (FramePoolHockey).

B) Coder la méthode remplirCombo :

Cette méthode servira à remplir les 3 JComboBox (listes déroulantes) présentes dans les onglets Avants, Défenseurs, Gardiens. À l'aide de votre liste de joueurs disponibles, ajouter **le nom** de chaque joueur dans le bon JComboBox.

Type de Joueurs	Nom du JComboBox
Avant	comboAvants
Défenseur	comboDéfenseurs
Gardien	comboGardiens

Méthodes importantes JComboBox

- pour ajouter un item à un JComboBox : addItem

ex : comboAvants.addItem(" bonjour");

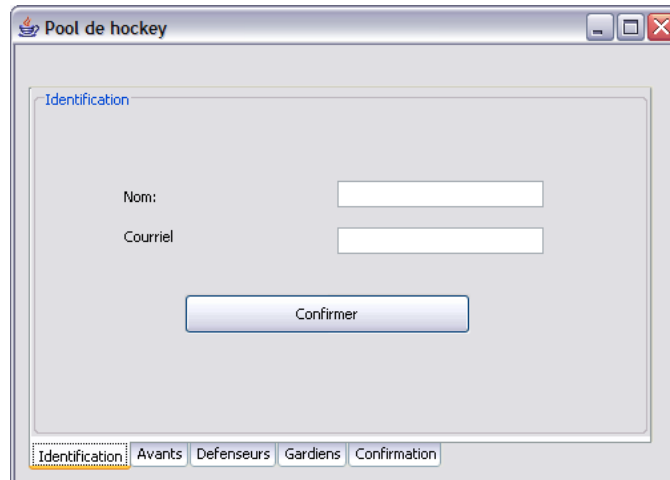
- pour savoir quel item est sélectionné dans le JComboBox : getSelectedItem

ex : (String)comboAvants.getSelectedItem(); → retourne la String sélectionnée dans le JComboBox

- pour enlever tout du JComboBox : removeAllItems

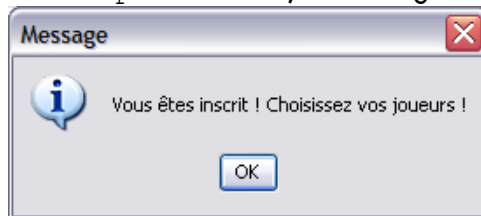
ex : comboAvants.removeAllItems → vide le JComboBox

C) L'onglet Identification (panelIdentification) :



Cet onglet permet au participant de s'identifier
à faire :

- * gérer le boutonConfirmerNom : créer l'objet Participant déclaré à l'aide du contenu des champs texte
- * Afficher, à l'aide d'un JOptionPane, le message suivant :



où le faire :

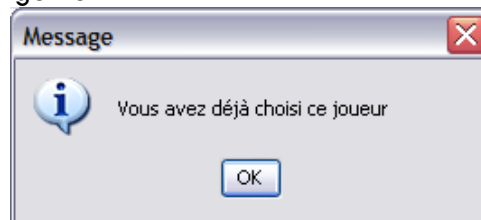
- * à l'endroit indiqué dans le code en commentaires (gestion de l'événement boutonConfirmerNom, coder le if)

D) L'onglet Avants

Le participant doit maintenant choisir 7 joueurs d'attaque, en cliquant sur Ajouter pour chacun.

à faire :

- * utiliser la / les méthode(s) codée(s) dans la classe `Participant` pour savoir si le joueur choisi est accepté
 - si oui : il est ajouté dans le `Vector` du `Participant`
 - si oui : vous ajoutez son nom dans le `JComboBox` `comboJoueursChoisis` situé dans le `panelSynthèse`
 - si oui : vous affichez le salaire total des joueurs choisis par le participant dans le `champTexte` `champSalaire`
 - si non : affichez un `JOptionPane` à l'utilisateur l'avertissant du problème du genre :



où le faire :

- à l'endroit indiqué dans le code en commentaires (gestion de l'événement `boutonAjouterAvant`, codez le if)

E) L'onglet Defenseurs

À faire : même chose qu'en D) sauf cette fois-ci avec les défenseurs

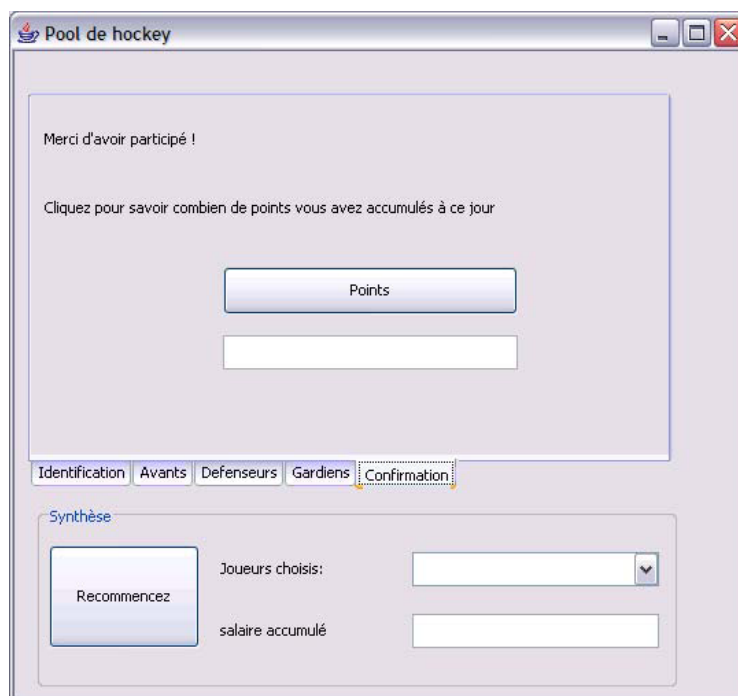
Où le faire: à l'endroit indiqué dans le code en commentaires (gestion de l'événement `boutonAjouterDefenseurs`, coder le if)

F) L'onglet Gardiens

À faire : même chose qu'en D) sauf cette fois-ci avec les gardiens

Où le faire: à l'endroit indiqué dans le code en commentaires (gestion de l'événement `boutonAjouterGardiens`, coder le if)

G) L'onglet Confirmation



À faire:

- gérer le `boutonPoints` : afficher dans le champ texte `champPoints` le résultat de votre méthode de la classe `Participant` calculant le nombre de points de l'équipe que s'est créée le participant.

Où le faire:

- à l'endroit indiqué dans le code en commentaires (gestion de l'événement boutonPoints, coder le if)

H) Le bouton boutonRecommencer (à l'endroit indiqué dans le code en commentaires)

- vider le Vector de joueurs choisis de l'objet Participant
- vider le comboBox comboJoueursChoisis
- vider le champSalaire

ANNEXE – méthodes à utiliser avec l'interface graphique

Méthodes importantes JComboBox

- pour ajouter un item à un JComboBox : addItem

Ex: `comboAvants.addItem("bonjour");`

- pour savoir quel item est sélectionné dans le JComboBox : `getSelectedItem`

Ex: `(String) comboAvants.getSelectedItem();` → retourne la String sélectionnée dans le JComboBox

- pour enlever tout du JComboBox: `removeAllItems`

Ex: `comboAvants.removeAllItems()` → vide le JComboBox

Méthode importante JOptionPane : pour créer un "messageBox"

- `JOptionPane.showMessageDialog (null, "un message de Éric");`

Méthodes importantes JTextField: champs texte

- Pour obtenir le texte écrit dans le champ texte : `getText()`

Ex.: `String salaire = champSalaire.getText();`

- Pour écrire dans le champ texte: `setText()`

Ex.: `champSalaire.setText("à écrire");`

Méthode importante : pour passer d'une String à un int ou vice-versa

- String à int : `Integer.parseInt (uneString);`
Ex.: `int calcul = Integer.parseInt (champSalaire.getText());`
- int à String : `String.valueOf (int, double, long, etc.)`
Ex.: `champTexte.setText(String.valueOf(unDouble);`