TP3: Tuples, Types Algébriques Simples

1 Préliminaires

Configuration En vous aidant de la Notice ScalaIDE :

1. Lancez l'éditeur ScalaIDE (Eclipse 4.7 Scala)
2. Importez le projet ScalaIDE `TP3.zip` disponible sur Moodle.

Partage de code avec votre binôme **Pour chaque TP**, vous devrez vous assurer que chaque membre du binôme dispose bien du code produit. Pour partager votre TP en fin de séance, réalisez un export du projet correspondant (voir Notice ScalaIDE). **Attention :** vérifiez que l'export a bien été réalisé en essayant de l'importer dans ScalaIDE.

Auto-évaluation des TP Pour vous aider à auto-évaluer votre travail, nous vous fournissons quelques tests unitaires. La Notice ScalaIDE décrit la procédure à suivre pour exécuter ces tests.

2 Jeu de cartes : package `fr.istic.si2.tp3.cartes`

Dans cet exercice, on souhaite modéliser des cartes à jouer, et programmer quelques fonctions sur les cartes. On utilise pour cela des types algébriques. Leur définition est fournie dans le fichier `jeuDeCartes.scala`.

Exercice 1 (Construction de valeurs)**.** *Complétez les définitions des* `val asPic`*,* `roiCoeur` *et* `septCarreau`*.*

Exercice 2. *Programmez la fonction* `estUneFigure`*, qui détermine si la carte prise en paramètre est une figure ou non. Exemples : Le roi de pique est une figure, le 7 de pique ou l'as de cœur ne sont pas des figures.*

Testez votre fonction sur quelques exemples en affichant le résultat qu'elle produit (à l'aide d'un simple `println` *dans le code de votre application).*

Exercice 3. *Programmez la fonction* `bataille`*. On rappelle qu'il y a une bataille entre deux cartes quand ces deux cartes ont la même valeur. Exemples : l'as de pique et l'as de coeur sont en bataille, le 7 de carreau et le 7 de trèfle sont en bataille ; en revanche, le 2 de pique et le valet de pique ne sont pas en bataille.*

Testez votre fonction sur quelques exemples en affichant le résultat qu'elle produit (à l'aide d'un simple `println` *dans le code de votre application).*

Exercice 4. *On souhaite maintenant savoir si deux cartes ont la même couleur, et dans ce cas-là, calculer cette couleur. On va donc utiliser le type* `Option[Couleur]`*. Programmez deux versions de la fonction déterminant la couleur commune de deux cartes (si elle existe) : sans et avec pattern matching.*

Testez votre fonction sur quelques exemples en affichant le résultat qu'elle produit (à l'aide d'un simple `println` *dans le code de votre application).*

Exercice 5. *Programmez la fonction* `changeCouleur`*.*

Testez votre fonction sur quelques exemples en affichant le résultat qu'elle produit (à l'aide d'un simple `println` *dans le code de votre application).*

On cherche maintenant à modéliser des mains [1] de trois cartes. Pour cela, on utilise un tuple, et on définit le type alias `Hand`.

Exercice 6. *On souhaite déterminer si une main contient une carte de couleur rouge* [2]*. Pour cela, on découpe le problème en deux sous-problèmes :*

- *le calcul des couleurs des cartes de la main (fonction* `couleurs`*)*
- *la fonction* `contientRouges`*, qui résout le problème principal*

Programmez ces deux fonctions.

Testez vos fonctions sur quelques exemples en affichant le résultat qu'elles produisent (à l'aide d'un simple `println` *dans le code de votre application).*

Exercice 7 (Tests unitaires)**.** *Vérifiez que vos fonctions passent les tests fournis pour ces exercices. Aidez-vous de la notice ScalaIDE au besoin. Si nécessaire, corrigez vos fonctions et re-lancez les tests.*

3 Addition de restaurant : package `fr.istic.si2.tp3.restaurant`

Un restaurant propose la carte indiquée dans la figure. Les formules proposées pour les repas sont les suivantes : *Petite Faim* (plat uniquement), *Entrée-Plat* ou *Plat-Dessert*, et enfin la formule *Complète* (entrée, plat et dessert).

Les prix de base des plats sont indiqués dans la carte ci-contre. Mais pour certaines formules, des réductions globales s'appliquent sur le total de la commande :

- aucune réduction pour la formule *Petite Faim*
- une réduction de 2€ pour la formule *Complète*
- une réduction de 1€ pour les autres formules

L'objectif de cet exercice est de programmer les fonctionnalités de prise de commande, et le calcul de l'addition pour la formule commandée.

Entrées	
Foie Gras	8€
Melon	4€
Crudités	5€
Plats	
Gratin	10€
Poisson du jour	11€
Couscous *Poulet · merguez*	12€
Couscous Végétarien *Legumes*	10€
Couscous Royal *Poulet · agneau · merguez*	14€
Desserts	
Café	1€
Salade de fruits	4€
Glace	5€
Glace chocolat	6€
Glace chantilly	7€

L'ensemble de la carte ainsi que les formules proposées sont modélisées par plusieurs types algébriques. Tout cela est fourni dans le fichier `definitions.scala`. Lisez ces définitions, et comprenez-les. C'est aussi dans ce fichier que vous devrez programmer les fonctions marquées `TODO`. Pour la prise de commande, vous devrez **scrupuleusement** respecter le nom des plats tels qu'ils sont écrits sur la carte (majuscules, et accents).

Pour tester régulièrement vos fonctions, vous disposez des tests unitaires fournis dans le répertoire `tests`, dans le package `fr.istic.si2.tests.tp3.restaurant`. Vous pourrez également tester de manière interactive le résultat de vos fonctions, en écrivant des tests dans le fichier `testsInterfactifs.scala` du répertoire `src`.

Enfin, on fournit dans le fichier `main.scala` une boucle d'interaction qui effectue la prise de commande, et affiche l'addition de la commande, en utilisant les fonctions que vous aurez précédemment programmées. Vous ne pourrez exécuter cette boucle d'interaction qu'après avoir fourni une implémentation pour toutes les fonctions marquées `TODO`, sauf la dernière `changerOuAnnulerDessert`, pour laquelle on vous fournit un comportement par défaut.

1. On appelle « main » l'ensemble des cartes possédées par un joueur.
2. Les couleurs « rouges » sont : Coeur et Carreau.