Une pause-café bien méritée pour terminer !

1. En mode déconnecté pour commencer …

Ne serait-ce pas temps de déguster un bon café 😉 ?
L'ŒIL projette de se transformer en coffee-bar et de proposer la carte des cafés ci-contre.

Café't'œil

Cafés

Colombia	0,89 €
Deca	1,05 €
Expresso	1,99 €

Suppléments

Chantilly	0,10 €
Caramel	0,15 €
Chocolat	0,20 €

En plus des cafés classiques tels que l'Expresso, le Colombia ou le Déca, ce coffee-bar souhaiterait vous permettre d'ajouter, si vous le souhaitez, un ou plusieurs ingrédients supplémentaires à votre boisson comme du caramel, du chocolat et même pourquoi pas couronner le tout de mousse Chantilly…
Mais bien sûr, qui dit ingrédient supplémentaire dit coût supplémentaire et pour *bien* gérer ce coffee-bar, l'ŒIL a absolument besoin d'un système de prise de commandes qui s'adapte à ses besoins et facture donc individuellement chacun des suppléments 😉

Il est à noter qu'un ingrédient supplémentaire est, ***non seulement***, ajouté à une boisson (de base) telle qu'Expresso, Colombia ou Déca, ***mais aussi*** qu'un ingrédient supplémentaire est lui-même (considéré comme) une boisson (puisqu'il sera mélangé au café pour constituer au final une boisson originale qui sera dégustée avec un bon cookie).

Pour répondre au besoin de ce coffee-bar, notre équipe de développeurs expérimentés a écrit le code suivant :

```
public interface Boisson {
	public abstract String description();
	public abstract double prix();
}


public class Espresso implements Boisson {

	@Override
	public String description() {
		return "Espresso";
	}

	@Override
	public double prix() {
		return 1.99;
	}
}
```

```java
public class Colombia implements Boisson {

	@Override
	public String description() {
		return "Pur Colombia";
	}

	@Override
	public double prix() {
		return .89;
	}
}


public class Deca implements Boisson {

	@Override
	public String description() {
		return "Café déca";
	}

	@Override
	public double prix() {
		return 1.05;
	}
}



public abstract class IngredientSupplementaire implements Boisson {
	protected Boisson boisson;
}



public class Caramel extends IngredientSupplementaire {
	public Caramel(Boisson boisson) {
		this.boisson = boisson;
	}



	@Override
	public String description() {
		return boisson.description() + ", Caramel";
	}

	@Override
	public double prix() {
		return boisson.prix() + .15;
	}
}
```

```java
public class Chocolat extends IngredientSupplementaire {
	public Chocolat(Boisson boisson) {
		this.boisson = boisson;
	}

	@Override
	public String description() {
		return boisson.description() + ", Chocolat";
	}

	@Override
	public double prix() {
		return boisson.prix() + .20;
	}
}
```

```java
public class Chantilly extends IngredientSupplementaire {
	public Chantilly(Boisson boisson) {
		this.boisson = boisson;
	}

	@Override
	public String description() {
		return boisson.description() + ", Chantilly";
	}

	@Override
	public double prix() {
		return boisson.prix() + .10;
	}
}
```

a. Le code précédent a été implémenté à partir d'un diagramme de classes réalisé durant la phase de conception par un architecte (logiciel).
Sur une feuille de papier, procédez à une rétro-conception du code précédent pour retrouver le **diagramme de classes** proposé par l'architecte pour résoudre les besoins du coffee-bar de l'ŒIL.

b. Notez, sur le modèle précédent, la relation appropriée (**EST-UN** ou **A-UN** ou **IMPLÉMENTE)** sur chaque relation du diagramme de classes.

c. Pour tester cette conception, un programme `main` permettant de réaliser les trois scénarii suivants a été implémenté :
   - Préparer un Expresso sans ingrédient supplémentaire
   - Préparer un Colombia avec Caramel, Chocolat et Chantilly
   - Préparer un Colombia avec un double chocolat

```
public class CafetOeil {

	public static void main(String args[]) {
		Boisson expressoSimple = new Espresso();
		System.out.println(expressoSimple.description()
				+ " : " + expressoSimple.prix()+ " €");

		Boisson colombiaAmeliore = new Colombia();
		colombiaAmeliore = new Caramel(colombiaAmeliore);
		colombiaAmeliore = new Chocolat(colombiaAmeliore);
		colombiaAmeliore = new Chantilly(colombiaAmeliore);
		System.out.println(colombiaAmeliore.description()
				+ " : " + colombiaAmeliore.prix()+ " €");

		Boisson autreColombiaAmeliore = new Colombia();
		autreColombiaAmeliore = new Chocolat(autreColombiaAmeliore);
		autreColombiaAmeliore = new Chocolat(autreColombiaAmeliore);
		System.out.println(autreColombiaAmeliore.description()
				+ " : " + autreColombiaAmeliore.prix()+ " €");
	}
}
```

Le résultat de l'exécution de ce programme est :

```
Espresso : 1.99 €
Pur Colombia, Caramel, Chocolat, Chantilly : 1.34 €
Pur Colombia, Chocolat, Chocolat : 1.29 €
```

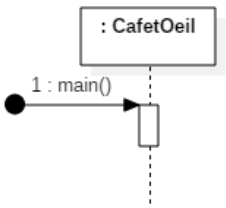
Sur une feuille de papier, Représentez sur un diagramme de séquences ***toutes*** les interactions (messages échangés entre les objets) lors de l'exécution du deuxième scénario de tests de la méthode `main` de la classe **`CafetOeil`**, c-a-d le diagramme de séquences permettant modéliser le code ci-dessous.

Pour cet exercice, on vous demande de représenter de manière explicite ***tous les messages retours*** relatifs aux interactions liées à la description et au coût.

```
public class CafetOeil {

   public static void main(String args[]) {
   //…
      Boisson colombiaAmeliore = new Colombia();
      colombiaAmeliore = new Caramel(colombiaAmeliore);
      colombiaAmeliore = new Chocolat(colombiaAmeliore);
      colombiaAmeliore = new Chantilly(colombiaAmeliore);
      System.out.println(colombiaAmeliore.description()
              + " : " + colombiaAmeliore.prix()+ " €");
 //…
   }
}
```

Ce diagramme de séquences commencera de la manière suivante.
A vous de le compléter avec les objets et messages appropriés.
On ne demande pas les stéréotypes de Jacobson.



Vous pourrez passer au mode connecté uniquement lorsque le diagramme de CLASSES ET le diagramme de SEQUENCES auront été réalisés sur une feuille de papier.

Montrez ces diagrammes à votre enseignant avant de continuer !!!

… même si la vérification de votre diagramme de séquence
ne se fera qu'en fin de TP comme indiqué en fin de cet énoncé 😉…

2. Récupérer le projet `cafetoeil` sur le remote (`Github`)

… suivre la procédure habituelle…

Sur le remote :
❑ Rendez-vous sur github à l'adresse suivante : https://github.com/iblasquez/cafetoeil
❑ Cliquez sur le bouton vert **Code** et copiez le lien **https** du dépôt en cliquant sur l'icône des 2 carrés à droite de l'adresse.

En local :
❑ Ouvrez **Git Bash**
❑ Placez-vous dans le **workspace `javabut1`** (les slashs des chemins sont inversés en linux et Windows)
❑ Clonez le dépôt **`cafetoeil`** via la **ligne de commande `git clone`** et de l'adresse **`https`** copiée.

Sous Eclipse :
❑ Lancez Eclipse avec le workspace **`javabut1`**
❑ Importez un ***projet Maven existant*** et cliquez sur **Next**.
❑ Cliquez sur **Browse** et sélectionnez le projet **`cafetoeil`** de manière à ce que dans le cadre **Projects** vous voyez bien un `pom.xml` Si tel est le cas, cliquez alors sur **Finish**

3. Familiariser-vous avec ce projet existant …

Pour vous familiariser avec ce projet :
❑ consulter **l'historique des versions** (**`Team→Show In History`**)
❑ **promenez-vous un peu dans le code** pour retrouver le code proposé dans la première partie de l'énoncé et comprendre comment il a été architecturé.
❑ procédez à une **rétro-conception du code existant (à l'aide de plantUML par exemple sous Eclipse) pour vous permettre de vérifier votre diagramme de classes.**
❑ **lancez les jeux d'essais** : les scénarios implémentés dans le `main` dans le paquetage **application**
❑ **consultez et lancez les tests automatisés :** a priori dans **`BoissonTest`**. Que constatez-vous ? 😉
❑ **lancer éventuellement SonarLint**

Le projet cloné `cafetoeil` est fonctionnel (c-a-d il marche !)
… Par contre, il n'a pas encore de tests automatisés
ce qui implique que son comportement n'est donc pas garanti en cas de modification du code 😉

4. Mise en place des tests automatisés pour garantir le comportement existant

a. Les bonnes pratiques de programmation et de conception recommandent d'écrire des tests automatisés En vous aidant, des jeux d'essais écrits dans la classe **`CafetOeil`** et des résultats de l'exécution de ces jeux d'essais sur la console, transformer les jeux d'essais du **`main`** en trois tests automatisés (qui testent bien sûr les mêmes scénarii).
Vous écrirez ces trois méthodes de tests dans la classe `BoissonTest` en veillant à respecter les bonnes pratiques d'écriture de test :
   - Un nom de méthode de test significatif pour chaque test montrant l'intention du test
   - Le respect du pattern 3A dans l'écriture de vos tests respect du pattern 3A.

     Rappel : Pattern 3A : **A** pour ……
     A pour ……
     A pour ……

b. Lancez la couverture de code par les tests.
Ajoutez un (ou plusieurs) test(s) pertinent(s) de manière à ce que le code métier existant (celui présent dans le paquetage **`model`** et ses sous-paquetages) soit couvert à 100% par les tests.

c. Lorsque que le code métier existant est couvert à 100%, effectuez un nouveau commit avec un message du genre « **`mise en place d'un harnais de tests avec BoissonTest`** »

5. Modification du code : extension de l'application avec un nouveau café et un nouveau supplément

Le coffee-bar fonctionne depuis quelques mois et les clients demandent fréquemment s'ils peuvent ajouter du **lait** à leur café. L'ŒIL souhaite donc ajouter le lait à sa carte de suppléments pour un tarif de 0,10 €. Certains clients aimerait aussi pouvoir commandé un café plus corsé que le Colombia. L'ŒIL décide donc de mettre à sa carte un nouveau café : le **Sumatra** pour un prix de 0,99 €.

a. Modifiez le diagramme de classes de la question 1 (sur papier en mode déconnecté) pour que votre système prenne désormais en compte, dans sa **conception**, le nouveau Café de type **Sumatra** et le nouveau supplément **lait**.

b. **Implémentez** cette extension de l'application dans votre projet `cafetoeil` au(x) bon(s) endroit(s) dans le projet bien sûr 😉

c. **Testez** votre implémentation avec un nouveau test automatisé dans la classe `BoissonTest`.
Vérifiez que la couverture du code métier par les tests est à 100%

d. Effectuez un **nouveau commit** avec un message du genre : « `ajout café Sumatra et supplément Lait` »

e. Que pensez-vous de la conception (diagramme de classes) proposée par l'architecte ?
Cette modification/extension de l'application a-t-elle été rapide/facile à mettre en place ?

6. Diagramme de séquences

Il est temps de vérifier ce diagramme de séquence !

a. Tout d'abord seul, à partir du code et avec l'aide de votre IDE, essayez de vérifier (et/ou compléter) le diagramme de classes que vous aivez réalisé sur papier en suivant l'exécution pas à pas de ce scénario avec des ***CTRL+clic*** 😉

b. Une fois que vous pensez avoir le bon diagramme de séquences, appelez votre enseignant de TP pour faire une vérification commune !

Profitez du temps restant pour terminer un ancien TP
ou avancer votre SAE si tous vos TP sont déjà terminés 😉

Image : https://tenor.com/view/coffee-coffee-emoji-coffee-time-coffee-love-coffee-morning-gif-22986225