TP-1 : Premiers pas en Smalltalk

Noury Bouraqadi

1 Présentation

L'objectif de ce TP est de vous familiariser avec l'environnement de programmation de Smalltalk tout en révisant les concepts d'objet et de classe. Cette familiarisation se fera à l'aide du Smalltalk libre Pharo.

2 Créez un projet Pharo

- Avec **PharoLauncher** créez une nouvelle image Pharo que vous nommerez : **PremiersPas**
- Affichez le dossier de l'image **PremiersPas** à partir du launcher :

  Clic-droit sur l'image > show in Folder.
- Vérifiez que le fichier **PremiersPas.changes** a été créé automatiquement, ainsi qu'un certain nombre d'autres fichiers groupés dans le même dossier.
- Retournez au Pharo Launcher et ouvrez la nouvelle image :

  Clic-droit sur l'image >launch

3 Lampe

Dans le package **Trafic**, définissez la classe **Lampe** qui dispose de deux variables d'instances : **estAllumee** et **couleur**. La variable d'instance **estAllumee** référence un booléen qui est **true** quand la lampe est allumée et **false** quand la lampe est éteinte. Quant à la variable d'instance **couleur**, elle référence une instance de la classe **Color** (disponible dans la bibliothèque de base de Pharo). Des instances prédéfinies de la classe **Color** peuvent être obtenues à l'aide des expressions suivantes :

- **Color red**
- **Color green**
- **Color blue**
- **Color yellow**
- **Color gray**

Dotez la classe **Lampe** du protocole **etat** et définissez y les méthodes suivantes :

- **estAllumee** retourne le booléen contenu dans la variable d'instance de même nom.
- **estAllumee: unBooleen** affecte à la variable d'instances **estAllumee** le booléen référencé par l'argument **unBooleen**.
- **allumer** allume la lampe, ce qui revient à modifier la valeur de la variable d'instances **estAllumee**. Utilisez pour ce faire la méthode **estAllumee:**.
- **eteindre** éteint la lampe. Opère de manière symétrique à la méthode **allumer.**

Dotez la classe **Lampe** du protocole **apparence** et définissez y les deux méthodes suivantes :

- **couleur: uneCouleur** modifie la couleur de la lampe pour utiliser la couleur correspondant au paramètre **uneCouleur.**
- **couleur** Si la lampe est allumée, cette méthode retourne la couleur courante de la lampe référencée par la variable d'instance de même nom**.** Si la lampe est éteinte, la couleur grise est retournée (**Color gray**).

Nous souhaitons que toute nouvelle lampe soit par défaut jaune et éteinte. Pour ce faire, dotez la classe **Lampe** du protocole **initialisation** et définissez y la méthode suivante :

- **initialize** pour initialiser toute nouvelle lampe avec des valeurs par défaut pour la couleur et l'état. Par défaut une lampe est de couleur jaune (**Color yellow)**. Elle est initialement éteinte.

Dans un **Playground**, testez le code que vous venez de développer. Pour ce faire, créez des instances de la classe **Lampe** et envoyez-leur des messages. Vérifier le bon fonctionnement à l'aide d'un inspecteur (menu « **inspect it** »).

4 Sauvegarde, import/export et récupération de code

3.1 Sauvegardez votre image, puis quittez-la (Menu **Save and Quit**). Relancez-la et vérifiez que votre environnement est inchangé et que votre code a été sauvegardé.

3.2 Sélectionnez le package **Trafic** de votre TP et exportez la à l'aide du menu **fileOut** (clic droit > Extra > File Out). Ouvrez le fichier **Trafic.st** obtenu dans le même dossier que l'image à l'aide d'un éditeur de texte (Notepad++ par exemple). Vérifiez que la totalité de votre code y est sauvegardée.

3.3 Créez une nouvelle image Pharo vierge (qui ne contient pas votre TP) et à l'aide de la souris, déposez y le fichier **Trafic.st** généré dans l'étape précédente. Un menu apparaît avec plusieurs options. Vous pouvez :

- Choisir : « **Install into the image »** pour charger tout le code du fichier dans l'image
- Choisir : «**Change list Browser ».** Vous obtenez une fenêtre de visualisation qui affiche le code contenu dans le fichier **.st** en *lecture seule* (impossible de modifier/sauvegarder). Vous pouvez sélectionner juste les parties de code qui vous intéressent et les charger (Bouton « **fileIn Selected** »). C'est très pratique pour récupérer du code partiellement erroné.

3.5 Chargez votre fichier **Trafic.st** dans une image vierge, puis quittez sans sauvegarder : menu **Quit** puis répondre **No** dans la boîte de dialogue « **Save changes before quitting? »**.

- Relancez la même image et constatez que le package **Trafic** et la classe **Lampe** n'ont pas été sauvegardées
- Cliquez sur le menu **Sources>Code Changes** une fenêtre « **Epicea – Code Changes** » apparaît. Elle contient l'historique de votre code, sauvegardé ou pas.
- Sélectionnez le deuxième élément de la liste gauche. Vous verrez apparaître sur la droite tout le code de la classe **Lampe**.
- Sélectionnez les parties qui vous intéressent, puis chargez-les à l'aide du menu clic droit « **Apply**... ». Le chargement est effectué après confirmation.
- Le menu clic droit « **Revert**... » permet d'annuler un chargement effectué par erreur.

5 Affichage textuel de la lampe

Dans le même package que la classe **Lampe**, définissez la classe **AfficheurTextuelDeLampe**. Comme son nom l'indique, une instance de cette classe affiche de manière textuelle les informations relatives à une lampe. Cet affichage est réalisé sur la console accessible à l'aide de la variable globale **Transcript**. Notez que pour voir l'affichage, la fenêtre de la console (**Transcript**) doit être ouverte via le menu *Tools>Transcript*.

La classe **AfficheurTextuelDeLampe** doit être dotée d'une seule variable d'instances: **affichageAutorise**. Elle correspond à un booléen qui est à **true** quand les informations relatives à une lampe sont affichées.

Dans le protocole **initialisation** de la classe **AfficheurTextuelDeLampe** ajouter la méthode **initialize** qui initialise la variable d'instance **affichageAutorise** avec la valeur par défaut, à savoir **true**.

Dotez la classe **AfficheurTextuelDeLampe** du protocole **affichage** qui contient les méthodes suivantes :

- **demarrerAffichage** affecte **true** à la variable d'instances **affichageAutorise**.
- **arreterAffichage** affecte **false** à la variable d'instances **affichageAutorise**.
- **miseAJourAvec: uneLampe**. Cette méthode effectue l'affichage de l'état (éteinte ou allumée) et la couleur de la lampe passée en paramètre. L'affichage consiste à écrire le texte adéquat sur la console (accessible via la variable globale **Transcript**).
  - Si **uneLampe** est éteinte, l'affichage obtenu est : **Lampe éteinte**
  - Si **uneLampe** est allumée et de couleur jaune (**Color yellow)**, l'affichage obtenu est : **Lampe allumée (Color yellow)**.

Notez que :

- La fenêtre de la console s'ouvre à l'aide du menu **Tools>Transcript**
- l'affichage n'est effectué que si **affichageAutorise** a pour valeur **true**.
- La chaîne de caractères qui représente le nom d'une couleur (*yellow, red, blue, green, ...*) s'obtient en envoyant à une instance de la classe **Color** le message **name**.
- La console (**Transcript**) sait répondre aux messages suivants :
  - **cr** provoque un retour à la ligne (carriage return).
  - **tab** affiche une tablulation
  - **show: uneChaineDeCaracteres** affiche la chaîne de caractères passée en argument.

Modifiez la classe **Lampe** en la dotant d'une variable d'instance **afficheur**. Cette variable d'instance est destinée à référencer un objet affichage de lampe. Dans le protocole **affichage** ajoutez les méthodes suivantes :

- **afficheur** retourne l'afficheur utilisé.
- **afficheur: unAfficheur** mémorise l'afficheur correspondant à l'argument unAfficheur.
- **mettreAJourAfficheur** opère la mise à jour de l'afficheur si la variable d'instances afficheur est initialisée, c'est à dire si la méthode afficheur retourne une valeur différente de **nil**. Notez que les messages **ifNil:** et **ifNotNil:** permettent d'exécuter de manière conditionnelle un bloc de code si le receveur est respectivement **nil** ou non. Voici des exemples d'utilisation des messages **ifNil:** et **ifNotNil:**

- **nil ifNil: [Transcript cr ; show : 'Hello']** affiche Hello sur la console.
- **123 ifNil: [Transcript cr ; show : 'Hello']** n'affiche rien.
- **123 ifNotNil: [Transcript cr ; show : 'GoodBye']** affiche GoodBye sur la console.
- **nil ifNotNil: [Transcript cr ; show : 'GoodBye']** n'affiche rien.
- **nil ifNil: [Transcript cr ; show : 'GoodBye']** affiche GoodBye sur la console.

- **demarrerAffichage** ne fait rien s'il n'y a pas d'afficheur (variable d'instance **afficheur** à **nil**). Sinon, envoi le message **demarrerAffichage** à l'afficheur et le met à jour.
- **arreterAffichage** ne fait rien s'il n'y a pas d'afficheur (variable d'instance **afficheur** à **nil**). Sinon, envoie le message **arreterAffichage** à l'afficheur.

Pour finir cet exercice, identifiez les méthodes de la classe **Lampe** à modifier pour mettre à jour l'afficheur (envoi du message **mettreAJourAfficheur**) de manière à afficher les changements de l'état et de la couleur de la lampe.

Enfin, assurez-vous que le **Transcript** est ouvert puis testez votre code en évaluant des expressions dans un **Playground**. Dans le tableau qui suit vous est fournie une série d'expressions évaluées dans le **Playground** et l'affichage correspondant dans le **Transcript**.

Code évalué dans le Playground	*Affichage sur le Transcript*
lampe := Lampe new. afficheur := AfficheurTextuelDeLampe new. lampe afficheur: afficheur.	Lampe éteinte
lampe allumer.	Lampe allumée (Color yellow)
lampe eteindre.	Lampe éteinte
lampe couleur: Color red.	Lampe éteinte
lampe allumer.	Lampe allumée (Color red)
lampe couleur: Color blue.	Lampe allumée (Color blue)
lampe arreterAffichage.	
lampe couleur: Color green.	
lampe eteindre.	
lampe demarrerAffichage.	Lampe éteinte
lampe allumer.	Lampe allumée (Color green)

6 Polymorphisme

Exportez votre code du package **Trafic** dans un fichier **.st** comme expliqué dans l'exercice 4. Puis, décompressez le fichier **Tp1.zip** fourni dans le dossier contenant toutes vos autres images. Ensuite, raffrachissez le *Pharo Launcher* (bouton **referesh**). Vous verrez l'image fournie pour le TP1.

Lancer l'image TP1 et charger y le fichier **.st** avec le code de votre package **Trafic**. Vérifiez qu'il est bien dans l'image **TP1**, puis sauvegardez l'image.

Dans le code fourni, vous trouverez la classe **AfficheurLEDRonde** dans le package **AfficheurLED**. Vous utilisez une instance de cette classe à la place de l'afficheur textuel. Ce remplacement doit se faire en changeant seulement la classe. **Le reste du code ne doit pas être modifié**. Pourtant il doit continuer à fonctionner. C'est une conséquence directe du polymorphisme. En effet, les instances des 2 classes **AfficheurLEDRonde** et

AfficheurTextuelDeLampe savent répondre aux messages envoyés par une lampe à son afficheur.

Astuce : L'expression suivante permet de supprimer tous les afficheurs graphiques ouvert et éviter d'avoir à cliquer plusieurs fois :

```
AfficheurLEDCarree toutSupprimer.
```

7 Feu tricolore

Toujours dans le package **Trafic**, définissez la classe **FeuTricolore**. Les instances de cette classe doivent avoir trois lampes : la première rouge, la seconde orange et la troisième verte. Le feu tricolore mémorise la lampe allumée à un instant donné. Lorsqu'il reçoit le message **allumerLampeSuivante**, il éteint la lampe courante, détermine puis allume la lampe suivante.

A l'initialisation, le feu tricolore associe à chaque lampe un afficheur graphique. Chaque afficheur peut être placé sur l'écran à l'aide du message **position:** qui prend comme paramètre un point dans l'écran. Exemple :

AfficheurLEDRonde position: 100@200 place l'afficheur au point d'abscisse 100 et de coordonnée 200

Enfin, le feu tricolore doit disposer de deux méthodes **demarrerAffichage** et **arreterAffichage** qui font respectivement apparaître et disparaître les afficheurs des lampes de l'écran.