

## 1 Téléchargez Pharo Smalltalk

Télécharger la dernière version de Pharo Smalltalk (v6) à partir de <http://pharo.org/> (balladez vous un peu sur le site), désarchivez selon le système que vous utilisez. La commande exécutable est **Pharo** dans le dossier obtenu. Exécutez la en ligne de commande dans un terminal ou avec l'interface graphique.

## 2 La syntaxe et les bases avec le tutorial

L'application s'ouvre avec une fenêtre ouverte : "Welcome to Pharo xxx". Dans cette fenêtre repérer : "PharoTutorial go." ou "ProfStef go" (avec *Pharo5*, c'est dans l'onglet "Learn Pharo Smalltalk". Placez le curseur derrière le point, ouvrez le menu contextuel "command-Clic" ou "control-clic" ou "clic droit" (selon souris), choisissez "doIt". Vous obtenez le même résultat avec le raccourci clavier "Cmd-d" ou "Control d" selon votre système. Idem pour "printIt" avec "Cmd-p" et "InspectIt" avec "Cmd-i", utiles partout et tout le temps.

Le tutorial va vous faire aller de fenêtre en fenêtre et vous présenter toute la syntaxe de base et quelques autres choses. Vous avez accès à un livre en ligne : <http://www.pharobyexample.org/>. Vous avez un Mooc en ligne : <http://mooc.pharo.org>.

## 3 L'Environnement, premières indications

Comme indiqué en cours, l'environnement n'est pas un détail mais une part intégrante du concept, permettant dans la vraie vie de programmer "in the large" vite et bien.

- Menu *World* : clic sur fond d'écran. Les items essentiels dans un premier temps sont *System Browser*, *Playground* et *Tools-Transcript* si vous voulez afficher des messages ; par exemple (`Transcript show: 'Hello World'; cr.`).
- Chaque sous-genêtre de chaque outil possède un menu contextuel, "Cmd-clic" ou "Ctrl-clic"
- Sauvegarde de vos travaux : *World-save*. Nous ferons plus subtil ultérieurement.
- Ouvrez un **System Browser**, dans le menu contextuel de sa fenêtre en haut à gauche, faites "Find Class" et cherchez la classe *OrderedCollection*. Une fois sélectionnée, regardez la liste de ses méthodes et le classement en catégories. Les catégories sont un concept de l'environnement ; elles n'ont pas d'incidence sur l'exécution des programmes.
- Ouvrez un *Playground*, c'est comme un tableau de travail, entrez des expressions, choisissez "doIt", "print It" ou "inspectIt" pour exécuter, exécuter et afficher le résultat ou exécuter et inspecter le résultat. Ceci vaut pour toute expression. Toute instruction est une expression.

---

```

1  t := Array new: 2.
2  t at: 1 put: #quelquechose.
3  t at: 1

5  c := OrderedCollection new: 4
6  1 to: 20 do: [:i | c add: i]
7  "even dit si un nombre est pair"
8  c count: [:each | each even]
9  "aller vous ballader sur la classe Collection pour regarder les
10 itérateurs disponibles"
```

---

## 4 Exercice 1 - classes, instances, méthodes d'instance

Ouvrez un *System Browser*, dans le menu contextuel de sa fenêtre haut-gauche, faites "Add Package", donnez lui un nom, par exemple HLIN603.

1. A définir la classe `Pile` implantée avec un `Array` (ce sera ainsi dans le corrigé) ou une `OrderedCollection` qui va bien aussi.
- Sélectionnez votre package et pour créer la classe, cliquez dans la seconde fenêtre du browser, renseignez le template, puis “accept”. Ensuite dans le *playground* essayez `Pile new` “inspectIt”.

---

```

1 Object subclass: #Pile
2   instanceVariableNames: 'contenu index capacite'
3   classVariableNames: 'tailleDefaut'
4   category: 'HLIN603'

```

---

- Définissez la méthode `initialize:`, équivalent d’un constructeur à 1 paramètre, qui initialise les 3 attributs (dites variables d’instance). Essayez ensuite : `Pile new initialize: 5`.

---

```

1 initialize: taille
2   "la pile est vide quand index = 0"
3   index := 0.
4   "la pile est pleine quand index = capacite"
5   capacite := taille.
6   "le contenu est stocké dans un tableau"
7   contenu := Array new: capacite.
8   "pour les tests"
9   self assert: (self isEmpty).

```

---

- Ecrivez les méthodes : `isEmpty`, `isFull`, `push: unObjet`, `pop`, `top`. Testez les dans le playground.
- Pour la rendre compatible avec le *printIt*, définissez la méthode suivante sur la classe. C’est l’équivalent du `toString()` de Java. L’opérateur de concaténation est “,” (par exemple ‘ab’ , ‘cd’).

---

```

1 printOn: aStream
2   aStream nextPutAll: 'une Pile, de taille: '.
3   capacite printOn: aStream.
4   aStream nextPutAll: ' contenant: '.
5   index printOn: aStream.
6   aStream nextPutAll: ' objets : ('.
7   contenu do: [ :each | each printOn: aStream. aStream space ].
8   aStream nextPut: \$.

```

---

- Signalez les exeptions, en première approche, vous écrirez : `self error: 'pile vide'..`
2. Apprenez à utiliser le debugger. insérer l’expression `self halt.` au début de la méthode `push:`. A l’exécution, le programme stoppe, choisissez “debug” dans le menu. Vous voyez la pile d’exécution. Vous pouvez exécuter le programme en pas à pas (les items de menu importants sont “into” et “over” pour entrer, ou pas, dans le détail de l’évaluation de l’expression courante. Le debugger est aussi un éditeur permettant le remplacement “à chaud”. Le debugger d’Eclipse a été construit sur le modèle de celui-ci.
3. Ecrire une méthode `grow` qui double la capacité d’une pile.

## 5 Exercice 2 - Composition - méthodes à plusieurs paramètres.

Réaliser une classe distributeur de Bonbons (type foire foraine) à n colonnes utilisant un tableau de n Piles de Bonbons.

- Création d’un distributeur de 2 colonnes : `D:= Distributeur new colonnes: 2 taille: 5`.
- Utilisation : `D remplir: 1 avec: Carambar.`, remplit la colonne 1 avec des instances de la classe `Carambar`. Créer une classe `Bonbon` et deux sous-classes `Carambar` et `Malabar` (par exemple). `D donner: 1.` : rend a `Carambar` s’il en reste dans la colonne 1, sinon `#YenAPlus`.

- Nous verrons la semaine prochaine comment imposer qu’une colonne contienne des carambars et pas autre chose.

## 6 Exercice 3 - Jeux de Test

Pharo intègre une solution rationnelle pour organiser des jeux de tests systématique dans l’espace (couverture du code) et le temps (rejouer les tests après une modification du code). Il faut pour cela créer, dans le même package que l’application une sous-classe de `TestCase`, comme indiqué en : <http://pharo.gforge.inria.fr/PBE1/PBE1ch8.html>. Appliquez ce tutoriel aux cas de la pile et du distributeur.

## 7 Exercice 4 - méthodes de classes (Take a walk on the wild side)

Les méthodes de classe s’exécutent en envoyant des messages aux classes (ainsi considérées comme des objets. Par exemple `Date today`. Pour observer ou définir des méthodes de classes, il faut cliquer sur le bouton “class” du browser.

- Avant de passer côté *class*, ajouter une variable de classe `tailleDefault` à la classe `Pile` (cela se fait côté instance - demandez vous pourquoi?).
- Coéfinir une méthode de classe `initialize` qui fixe à 5 la taille par défaut des piles, stockée dans la variable de classe `tailleDefault`. De par son nom cette méthode sera invoquée automatiquement.
- Redéfinir sur `Pile` la méthode de classe `new` pour qu’elle appelle la méthode **d’instance** `initialize` définie en section 5.
- Définir une méthode exemple contenant un programme utilisant une pile;

## 8 Exercice 5 - Héritage

Typage dynamique ne signifie pas absence de types. Il est possible de programmer des contrôles qui auront lieu à l’exécution.

- Ecrire une sous-classe `PileTypée` de la classe `Pile` qui permet de d’imposer le type des éléments empilés. Utilisation :

---

```

1 p := PileTypee de: Bonbon.
2 p push: Carambar new.
3 p push: #Carambar --> exception, l'habit ne fait pas le moine

```

---

- Si vous avez le temps, étendre la problématique au distributeur.

## 9 Exercice 6

Reprendre l’exercice des comptes bancaires fait en Ocaml. Même énoncé On considère les classes `Account`, `InterestAccount` et `SecureAccount` vues en cours. écrire la classe `bank` avec les méthodes suivantes : `add` : ajoute un compte bancaire ; `balance` : calcule la somme des soldes des comptes ; `fees` : prélève 5% de frais à tous les comptes.

Je vous donne la méthode qui permet d’adapter cette classe au *printIt* du système ;

---

```

1 printOn: aStream
2   aStream nextPutAll: 'une banque, avec comptes : '.
3   accounts printOn: aStream

```

---