CHAPTER 1

Une Interface Web avec Teapot pour TinyBlog

1.1 Correction semaine précédente

Vous pouvez charger la correction de la semaine précédente en exécutant le code suivant:

```
Metacello new
    smalltalkhubUser: 'PharoMooc' project: 'TinyBlog';
    version: #week2solution;
    configuration: 'TinyBlog';
    load
```

Après le chargement d'un package, il est recommandé d'exécuter les tests unitaires qu'il contient afin de vérifier le bon fonctionnement du code chargé. Pour cela, vous pouvez lancer l'outil TestRunner (World menu > Test Runner), chercher le package TinyBlog-Tests et lancer tous les tests unitaires de la classe `TBBlogTest` en cliquant sur le bouton "Run Selected". Tous les tests doivent être verts. Une alternative est de presser l'icone verte qui se situe à coté de la class `TBBlogTest`.

Ouvrez maintenant un browser de code pour regarder le code des classes `TBBlog` et `TBBlogTest` et compléter votre propre code si nécessaire. Avant de poursuivre, n'oubliez pas de committer une nouvelle version dans votre dépôt sur Smalltalkhub ou SS3 si vous avez modifié votre application.

1.2 Une interface web avec Teapot

Dans la suite, nous allons créer une première interface web pour TinyBlog en utilisant Teapot (http://smalltalkhub.com/#!/~zeroflag/Teapot). Nous verrons une solution bien plus complète avec Seaside par la suite.

La classe TBTeapotWebApp

Créer une classe nommée `TBTeapotWebApp` ainsi:

```
Object subclass: #TBTeapotWebApp
    instanceVariableNames: 'teapot'
    classVariableNames: 'Server'
    package: 'TinyBlog-Teapot'
```

La variable `teapot` contiendra un petit server HTTP Teapot. Ici on utilise une implémentation différente du Design Pattern Singleton en utilisant une variable de classe nommée `Server`. Nous faison cela afin de ne pas avoir deux serveurs gérant les connexions sur le même port.

Ajouter la méthode d'instance `initialize` pour initialiser la variable d'instance `teapot` :

```
TBTeapotWebApp >> initialize
    super initialize.
    teapot := Teapot configure: {
       #port -> 8081.
       #debugMode -> true }.
```

La Page d'accueil

Définissons une méthode `homePage` dans le protocol 'html' qui retourne le code HTML de la page d'accueil de notre application web. Commençons par une version simple :

```
TBTeapotWebApp >> homePage
    ^ '<html><body><h1>TinyBlog Web App</h1></body></html>'
```

Déclarer les URLs (routes)

Ajouter maintenant une méthode `start` pour que l'objet teapot réponde à des URLs particulières. Commençons par répondre à l'URL / lorsqu'elle est accédée en GET :

```
TBTeapotWebApp >> start
    "a get / is now returning an html welcome page"
    teapot
        GET: '/' -> [ self homePage ];
        start
```

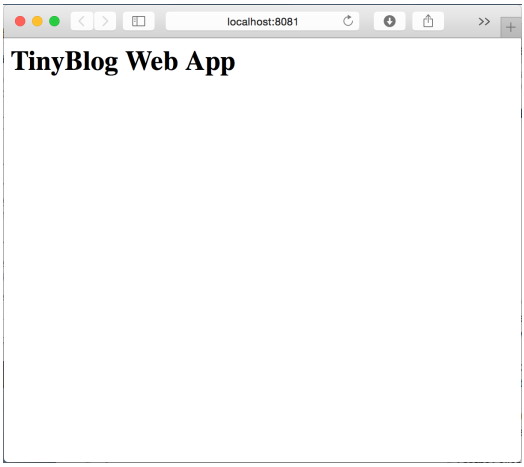

Figure 1.1 Une première page servie par notre application.

Stopper l'application

Ajouter également une méthode pour stopper l'application :

```
TBTeapotWebApp >> stop
    teapot stop
```

Démarrage l'application

Ajouter deux méthodes `start` et `stop` côté classe pour respectivement démarrer et arrêter l'application dans le protocol 'start/stop'. Ces méthodes utilisent la variable de class `Server` pour implanter un Singleton.

```
TBTeapotWebApp class >> start
    Server ifNil: [ Server := self new start ]
```

```
TBTeapotWebApp class >> stop
    Server ifNotNil: [ Server stop. Server := nil ]
```

1.3 Tester votre application

Maintenant nous pouvons lancer notre application en exécutant le code suivant pour démarrer votre application :

```
TBTeapotWebApp start
```

Avec un navigateur web, vous pouvez accéder à l'application via l'URL http://localhost:8081/. Vous devriez voir s'afficher le texte "TinyBlog Web App" comme la figure 1.1.

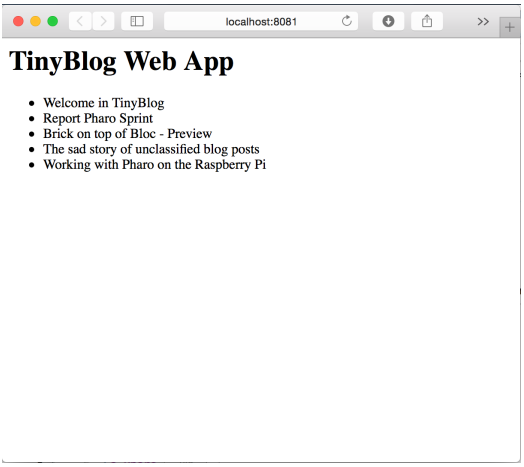

Figure 1.2 Afficher la liste des titres de posts.

1.4 Afficher la liste des posts

On souhaite maintenant modifier le code de la méthode `homePage` pour que la page d'accueil affiche la liste de tous les posts visibles. Pour rappel, tous les posts peuvent être obtenus via `TBBlog current allVisibleBlogPosts`. Ajoutons une méthode d'accès aux posts dans le protocol 'accessing' et modifions la méthode `homePage` ainsi que deux petites méthodes auxilliaires.

```
TBTeapotWebApp >> allPosts
   ^ TBBlog current allVisibleBlogPosts
```

Comme il faut générer une longue chaine de caractères contenant le code HTML de cette page, nous utilisons un flôt (stream) dans la méthode `homePage`. Nous avons également découpé le code en plusieurs méthodes dont `renderPageHeaderOn:` et `renderPageFooterOn:` qui permettent de générer l'en-tête et le pieds de la page html.

```
TBTeapotWebApp >> homePage
    ^ String streamContents: [ :s |
            self renderPageHeaderOn: s.
            s << '<h1>TinyBlog Web App</h1>'.
            s << '<ul>'.
            self allPosts do: [ :aPost |
                s << ('<li>', aPost title, '</li>') ].
            s << '</ul>'.
            self renderPageFooterOn: s.
        ]
```

Notez que le message `<<` est un synonyme du message `nextPutAll:` qui ajoute une collection d'éléments dans un flôt (stream).

```
TBTeapotWebApp >> renderPageHeaderOn: aStream
    aStream << '<html><body>'
```

```
TBTeapotWebApp >> renderPageFooterOn: aStream
    aStream << '</body></html>'
```

Tester l'application dans un navigateur web, vous devez maintenant voir la liste des titres des posts comme dans la figure 1.2. Si ce n'est pas le cas, assurez vous que votre blog a bien des posts. Vous pouvez utiliser le message `createDemoPosts` pour ajouter quelques postes génériques.

```
TBBlog createDemoPosts
```

1.5 Détails d'un Post

Ajouter une Nouvelle Page

Améliorons notre application. On souhaite que l'URL http://localhost:8081/post/1 permette de voir le post numéro 1.

Commençons par penser au pire, et définissons une méthode pour les erreurs. Nous définissons la méthode `errorPage` comme suit :

```
TBTeapotWebApp >> errorPage
    ^ String streamContents: [ :s |
        self renderPageHeaderOn: s.
        s << '<p>Oups, an error occurred</p>'.
        self renderPageFooterOn: s.
        ]
```

Teapot permet de définir des routes avec des patterns comme '<id>' dont la valeur est ensuite accessible dans l'objet requête reçu en paramètre du bloc.

Nous modifions donc la méthode `start` pour ajouter une nouvelle route à notre application permettant d'afficher le contenu d'un post.

```
TBTeapotWebApp >> start
    teapot
       GET: '/' -> [ self homePage ];
       GET: '/post/<id>' -> [ :request | self pageForPostNumber:
    (request at: #id) asNumber ];
       start
```

Il faut maintenant définir la méthode `pageForPostNumber:` qui affiche toutes les informations d'un post:

```
TBTeapotWebApp >> pageForPostNumber: aPostNumber
    |currentPost|
    currentPost := self allPosts at: aPostNumber ifAbsent: [ ^ self
    errorPage ].
    ^ String streamContents: [ :s |
        self renderPageHeaderOn: s.
```

```
        s << ('<h1>', currentPost title, '</h1>').
        s << ('<h3>', currentPost date mmddyyyy, '</h3>').
        s << ('<p> Category: ', currentPost category, '</p>').
        s << ('<p>', currentPost text, '</p>').
        self renderPageFooterOn: s.
        ]
```

Vous devez maintenant redémarrer le serveur avant de tester votre application en accédant à l'URL: http://localhost:8081/post/1.

Dans le code ci-dessus, on peut voir que le nombre passé dans l'URL est utilisé comme la position du post à afficher dans la collection des posts. Cette solution est simple mais fragile puisque si l'ordre des posts dans la collection change, une même URL ne désignera plus le même post.

Ajouter des liens vers les posts

Modifions la méthode `homePage` pour que les titres des posts soient des liens vers leur page respective.

```
TBTeapotWebApp >> homePage
    ^ String streamContents: [ :s |
         self renderPageHeaderOn: s.
         s << '<h1>TinyBlog Web App</h1>'.
         s << '<ul>'.
         self allPosts withIndexDo: [ :aPost :index |
              s << '<li>';
                << ('<a href="/post/', index asString, '">');
                << aPost title ;
                << '</a></li>' ].
            s << '</ul>'.
            self renderPageFooterOn: s
        ]
```

Maintenant, la page d'accueil de l'application affiche bien une liste de lien vers les posts.

1.6 Extensions possibles

Cette application est un exemple pédagogique à travers lequel vous avez manipulé des collections, des flôts (Streams), etc.

Plusieurs évolutions peuvent être apportées telles que:

- sur la page de détails d'un post, ajouter un lien pour revenir à la page d'accueil,
- ajouter une page affichant la liste cliquable des catégories de posts,
- ajouter une page affichant tous les posts d'une catégorie donnée,

- ajouter des styles CSS pour avoir un rendu plus agréable.