DEVELOPPEMENT D'APPLICATION WEB

JAVA – SERVEUR WEB

OBJECTIFS

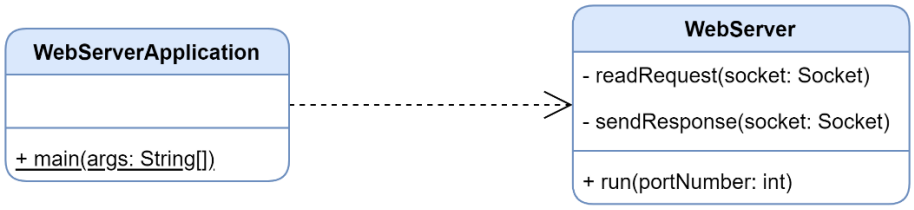
A travers cet exercice, vous allez :

- Etudier et reproduire le comportement d'un serveur web,
- Manipuler les sockets,
- Manipuler les fichiers
- Créer une application multithread.

Tout au long du sujet, vous penserez à traiter traiterez les exceptions.

CE N'EST PAS UNE CHAUSSETTE

➢ Créez un nouveau projet et implémentez les classes suivantes :



Note : en UML, lorsqu'un membre d'une classe est souligné, cela signifie qu'il est **`static`**.

- La méthode **`run`** de **`WebServer`** créera une instance de **`ServerSocket`** qui écoutera sur le port fourni en paramètre à la fonction, puis attendra les connexions des clients.

- Pour chaque connexion, vous appellerez la méthode **`readRequest`** qui affichera dans la console les informations transmises par le client, puis la méthode **`sendResponse`** qui indiquera au client que sa requête a bien été traitée et enfin vous pourrez fermer la connexion. Si vous avez lu votre cours, réaliser ces méthodes devrait être un jeu d'enfant...

- La méthode **`main`** créera une instance de **`WebServer`** qui démarrera sur le port 80.

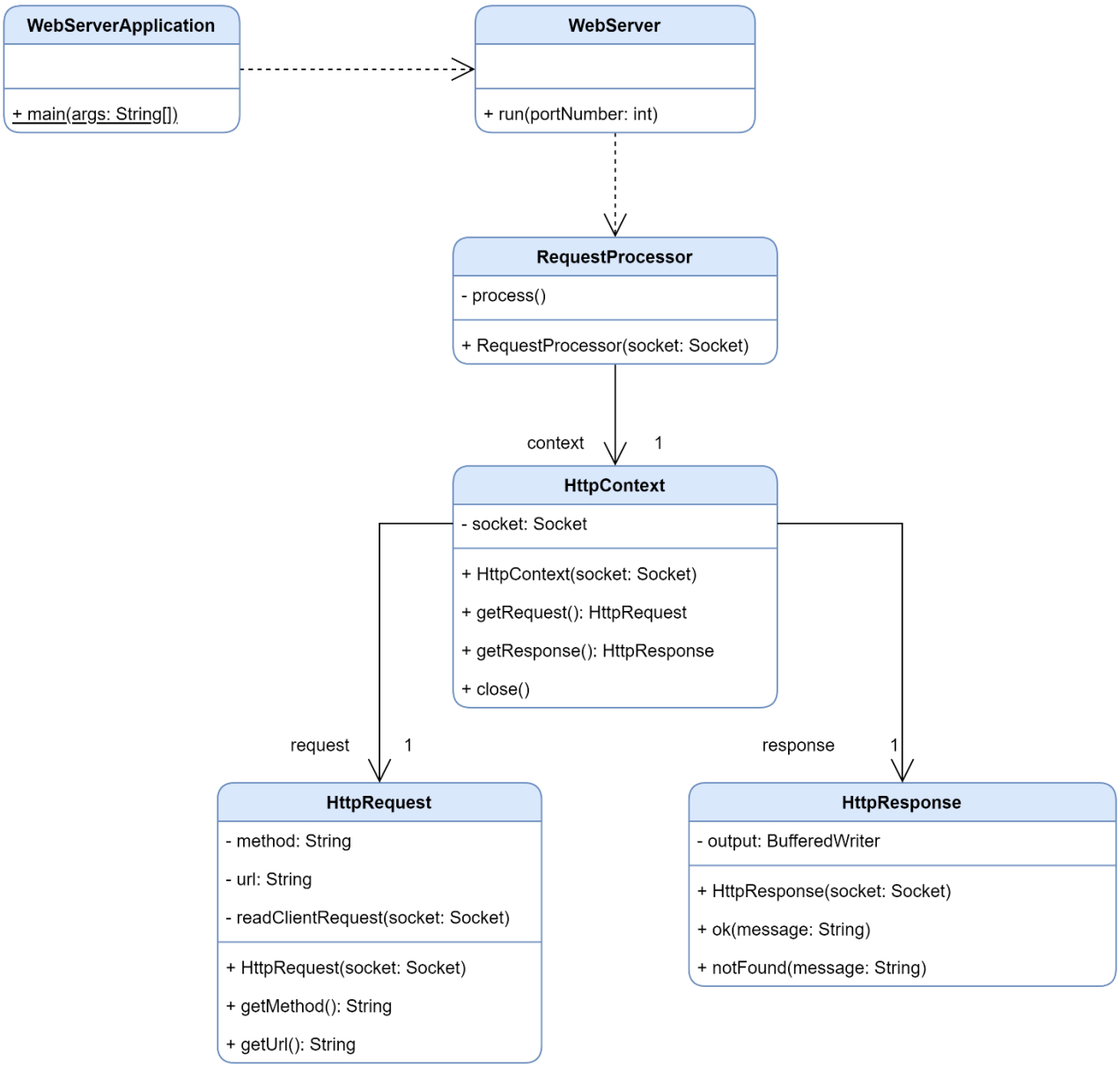
Astuces : Partez du principe, ici, que recevoir une chaine vide en provenance du client signifie qu'il n'y a plus d'information à lire.

Attention : Soyez prudent lorsque vous comparez des chaines de caractères en Java...

➢ Exécutez votre programme.
➢ Démarrez un navigateur depuis lequel vous accéderez à l'adresse `http://localhost`.
➢ Si tout fonctionne bien, votre navigateur doit afficher une page blanche sans symbole de chargement dans l'onglet de la page.
➢ Etudiez les informations reçues en provenance du client.

REORGANISATION

Vous êtes en mesure de lire les informations en provenance d'un client HTTP et vous parvenez à lui transmettre des données. A présent, allons un peu plus loin avec la modélisation suivante :



Allons, allons, il n'y a pas de quoi paniquer. Il y a juste 4 classes de plus pour lesquelles la plupart du code a déjà été partiellement écrit précédemment. Procédons, étape par étape.

HTTPREQUEST

La classe **HttpRequest** sera chargée de lire les informations transmises par le client et d'en extraire les éléments utiles.

- Implémentez la classe **HttpRequest**

  La méthode **readClientRequest** lira les informations en provenance du client et extraira la méthode HTTP (**GET**, **POST**, **PUT**, ...) utilisée et l'URL demandée.

HTTPRESPONSE

La classe **HttpResponse** sera utilisée pour transmettre des informations au client.

- Implémentez la classe **HttpResponse**

  La méthode **ok** indiquera au client que sa requête a bien été traitée

  ```
  HTTP/1.0 200 Message
  ```

  La méthode **notFound** indiquera au client que la ressource demandée n'a pas été trouvée :

  ```
  HTTP/1.0 404 Message
  ```

HTTPCONTEXT

La classe **HttpContext** gèrera la connexion avec le client et fournira un accès aux éléments de la requêtes ainsi qu'aux fonctionnalités de réponse.

- Implémentez la classe **HttpContext**

  La méthode **close** se chargera de fermer la connexion.

REQUESTPROCESSOR

La classe **RequestProcessor** s'occupera de traiter une requête d'un client.

- Implémentez la classe **RequestProcessor**

  La méthode process renverra un code **200** au client si l'url demandée est / et un code **404** dans les autres cas.

WEBSERVER

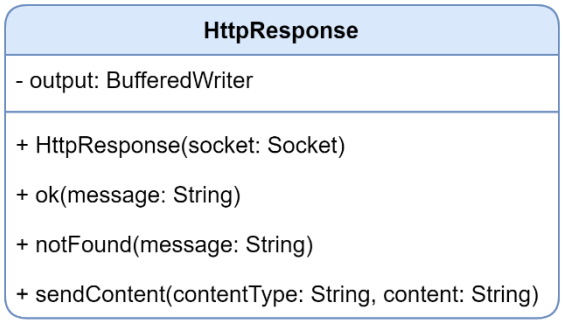
- Les méthodes **readRequest** et **sendResponse** ne sont plus d'actualité et doivent être supprimée.
- Créez une instance de la classe **RequestProcessor** qui traitera la requête du client lorsqu'une connexion est acceptée.
- Testez le bon fonctionnement du programme. L'accès à une page autre que la racine (ex : http://localhost/toto) devrait donner une erreur 404.

UN PEU DE CONTENU

A LA MANO

- Ajoutez une fonction **`sendContent`** à la classe **`HttpResponse`**.

  Cette fonction prendra en paramètres le type de contenu transmis (ex : « **`text/plain`** »), ainsi que le contenu à transmettre sous la forme d'une chaine de caractères.



  Jeter un œil au cours devrait vous permettre de gérer cela sans trop de difficultés.

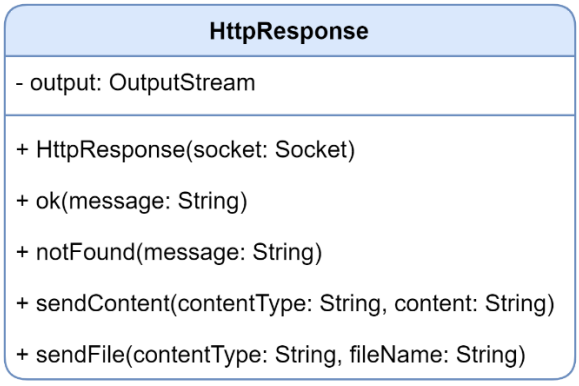
- Modifiez le code de la fonction process de **`RequestProcessor`** pour envoyer le texte « Hello World ! » (type : **`text/plain`**) après avoir envoyé le code **`200`**.
- Testez le bon fonctionnement.

- Modifiez à nouveau le code la fonction **`process`** pour envoyer le même texte mais en précisant le type **`text/html`**.
- Testez le fonctionnement. Une différence ?
- Intégrez des balise HTML (ex : **`strong`**) dans le contenu envoyé et testez à nouveau.

AVEC DES FICHIERS

- Ajoutez une fonction **`sendFile`** à la classe **`HttpResponse`**.

  Cette fonction prendra en paramètres le type de contenu transmis (ex : « **`text/html`** »), ainsi que le nom du fichier dont le contenu sera transmis.



Astuces : Quel que soit le type de fichier, traiter le comme un fichier binaire. Cela vous permettra, avec la même fonction, de manipuler des fichiers HTML comme des fichiers JPG.

Pour cela, modifiez l'attribut **`output`** de **`HttpResponse`** pour qu'il soit de type **`OutputStream`** (le type de retour de **`Socket.getOutputStream`**) et adaptez le code des méthodes de **`HttpResponse`**.

La fonction **`getBytes`** de **`String`** pourra vous être utile.

- Créez un dossier **`public`** à la racine de votre projet dans lequel vous placerez un fichier **`index.html`** contenant une page HTML simple (ex : votre création du dernier TD).
- Modifiez le code de la fonction **`process`** de la classe **`RequestProcessor`** afin que cette dernière transmette un code **`200`** ainsi que le fichier **`index.html`** précédemment créé lorsque l'url demandée est **`/`**.
- Testez le bon fonctionnement.

- Modifiez le code la fonction **`process`** pour que cette dernière :
  - Transmette un code **`200`** et le fichier **`index.html`** si l'url **`/`** est demandée,
  - Transmette un code **`200`** et le fichier demandé s'il existe dans le dossier **`public`**,
  - Retourne un code **`404`** si le fichier demandé n'existe pas.
  - Attention à bien fournir le bon type MIME pour l'entête **`Content-Type`** en fonction du type de fichier envoyé (dans un premier temps, prévoyez html, css et png).
- Testez le bon fonctionnement en ajoutant un fichier CSS lié à votre page HTML.

- Ajoutez une image à votre fichier HTML
- Testez le bon fonctionnement.

LA FIN DU « CHACUN SON TOUR »

- Modifiez le code de la fonction **`process`** de la classe **`RequestProcessor`** afin d'afficher la méthode de la requête, puis l'url demandée dans la console.
- Testez en demandant l'affichage de votre page Web.
  Vous devriez voir la liste des requêtes effectuées par le navigateur : le fichier HTML, le fichier CSS et l'image. Les requêtes sont traitées les unes après les autres, ce qui peut vite être problématique lorsque plusieurs utilisateurs tentent de se connecter en même temps ou que les temps de traitement des requêtes est long.

- Modifiez le code de **`RequestProcessor`** de manière à ce qu'il implémente l'interface **`Runnable`** vue en cours.
- Modifiez le code de **`WebServer`** afin de créer un nouveau thread pour chaque connexion.
- Testez le bon fonctionnement et observez.