TCP non bloquant et serveur HTTP simple

Exercice 1 - Un serveur Echo en non-bloquant

Ecrire un client serveur Echo acceptant de multiples clients en non-bloquant en utilisant pour cela un sélecteur (`java.nio.channels.Selector`).
Pour simplifier le développement, le serveur ne possèdera qu'une unique thread et dans un premier temps, on considérera

1. Rappeler le principe d'un sélecteur et comment on doit enregistrer un channel en accept, connection, read ou write.
2. Enregistrer le `ServerSocketChannel` pour être prévenu lors d'un `accept`, et faire un affichage indiquant le client qui se connecte.
3. Lors d'un `accept`, dans un premier temps, enregistrer directement le `SocketChannel` et read.
   Lorsque que le `SocketChannel` est ok pour le read, on enregistre celui-ci pour le write en passant la réponse en tant d'objet attaché de la clé de sélection.
4. Faire en sorte de gérer le `accept/connection` de façon non-bloquante.

Exercice 2 - Ecrire un proxy TCP en non-bloquant

Ré-écrire le proxy TCP en utilisant les sélecteurs. Il faudra penser à s'enregistrer en connect lors de l'établissement de la connection avec le serveur.

Exercice 3 - Un simple serveur de fichier HTTP

Écrire un petit programme Java qui permet d'émuler un serveur HTTP très simple en ré-utilisant le serveur TCP avec un pool de threads.
Voici les fonctionnalités qui doivent être implantées par ce petit serveur:

- répondre aux requêtes `GET` des clients sur des noms de fichiers réguliers (pas les répertoires);
- attribuer le type MIME `application/octet-stream` aux ressources retournées. On pourra également faire des essais, plus visuels, avec les types `text/plain` ou `text/html`. Ensuite, on pourra tenter de déterminer le `Content-Type` à partir de l'extension du fichier à renvoyer;
- pour simplifier la gestion de la connexion avec le client, on pourra fermer la connexion après l'envoi de la ressource. En revanche, il est nécessaire de lire l'intégralité de la requête du client avant de fermer la connexion.

Exercice 4 - Les messages d'erreur

Ajouter au serveur précédent la fonctionnalité suivante: lorsqu'une ressource recherchée par un client n'est pas un fichier régulier, et que le serveur n'est pas en mesure de la retourner, renvoyer un message d'erreur `404 Not Found` accompagné d'une page HTML expliquant poliment que la ressource n'est pas accessible.

Exercice 5 - Formulaire POST

Utiliser le formulaire ci-dessous pour regarder ce qu'envoit un browser lors POST sur un formulaire.

```
    <html>
          <body>
            <h1>Enquête de satisfaction</h1>

            <h4>Merci de remplir le questionnaire suivant:</h4>
            <form
              enctype="application/x-www-form-urlencoded"
              action="/enquete"
              method="POST">
              Votre nom : <input type="text" name="lastname"
value="" size="20"><br>
              Votre prénom : <input type="text" name="firstname"
value="" size="20"><br>
              Concernant la matière :
              <select name="thema">
                <option value="Électronique">Électronique
                <option value="Java">Java
                <option value="Java Réseau">Java Réseau
                <option value="Java Avancé">Java Avancé
                <option selected value="Java Réseau Avancé">Java
Réseau Avancé
              </select>, vous vous estimez :<br>

              Mécontent <input type="radio" name="satisfied"
value="10"><br>
              Assez content <input type="radio" name="satisfied"
value="20"><br>
              Content <input type="radio" name="satisfied"
value="30"><br>
              Très content <input checked type="radio"
name="satisfied" value="40"><br>

              <Input type="submit" value="Envoyer">
              <input type="reset" value="Réinitialiser">
            </form>

          </body>
     </html>
```