TD 4 : bserveur et bttp

Remarque préalable :

Il y a deux objectifs à ce td : ***bserveur*** et ***bttp*** ; si vous n'avez pas fini l'exercice ServiceCours du précédent td, ne faites que l'exercice 1 ***bserveur*** et finissez en priorité ServiceCours du précédent td (avec ***bserveur***).

Objectifs :

Création d'une bibliothèque logicielle spécifiant une architecture logicielle Serveur-Service
Passage d'une classe comme objet en paramètre
Définition-création d'un protocole de communication client-serveur simple

Introduction

Dans ce td, vous allez écrire une bibliothèque logicielle ***bserveur.jar*** dans laquelle le couple Serveur-Service est indépendant du service réellement rendu et pourra donc être testée sur les deux exercices (ServiceInversion-ServiceCours) du td 3.
La généralisation permettra également bien sur de se servir de cette bibliothèque dans le projet à venir.

Dans la mesure où l'on vise un service le plus général possible, il sera utile de mettre en place également un client ***bttp*** protocolaire à base d'échanges en boucle.

Exercice 1 : bserveur.jar et test sur ServiceInversion

Le premier point est de changer l'instanciation du service au sein du serveur

new ServiceInversion (socket)

par l'invocation de la méthode newInstance() de la classe Class

uneClasse.newinstance()

où uneClasse correspond à ServiceInversion.class. newinstance() n'acceptant pas de paramètres, il sera nécessaire de prévoir un setter pour la socket.
Cette classe manipulée comme objet devra être passée au constructeur de Serveur dans le main (comme présenté au dernier transparent du cours) et bien sur conservée en tant qu'attribut de Serveur :

new Thread(new Serveur (ServiceInversion.class, 1234)).start() ;

Attention à correctement typer cet attribut class de façon à n'accepter que des classes sous-classes de ***serveur.Service***.
La bibliothèque ***bserveur.jar*** doit contenir, dans un package ***serveur*** les 2 classes suivantes :
Serveur – avec les modifications indiquées ci dessus
Service - une classe abstraite qui doit factoriser les éléments communs à tout service, c'est-à-dire essentiellement l'attribut socket avec ses accesseurs et l'implémentation de Runnable (le run() n'étant bien sur pas implémenté)

Le package ***serveur*** au final doit pouvoir être exporté comme bibliothèque logicielle. On commence à avoir plusieurs bibliothèques utiles au fur et à mesure des tds, il serait bon de choisir un répertoire (*lib* dans votre *workspace* par exemple) pour les y placer systématiquement.

Ce travail fait, vous adapterez le corrigé (ou votre version) du td socket ServiceInversion et testerez le tout. Vous pouvez mettre AppliServeur et ServiceInversion dans un même package ***appli***.

Exercice 2 : le protocole de communication client/serveur BTTP1.0

Afin de faciliter l'utilisation de bserveur par des services variés, on établit un protocole simple à base de questions-réponses entre le service et le client, le Brette Texte Transfert Protocole (BTTP, petit frère le moins futé d'HTTP), qui démarre les échanges ping-pong par un envoi service→client

1. le service envoie un texte au client ; ce texte implique une réponse du client
2. le client affiche ce texte dans la console et saisie la réponse qui est envoyée au service

Cet échange se poursuit jusqu'à ce que, soit le service ferme la connexion (client jugé indésirable, temps d'attente de requête dépassé ou autre critère à déterminer), soit le client, via sa réponse, indique qu'il souhaite arrêter les échanges (une saisie «» par exemple ou autre critère à déterminer).

Le respect de cet échange permet d'envisager d'écrire un **client BTTP** totalement indépendant du service, des questions posées,etc à base d'une répétition de cycles
« je reçois une string – je l'affiche »
« je saisie une string, je l'envoie »

Ce client se connectera en tapant une url « BTTP:host:port » - par exemple « BTTP:localhost:1234 »

Exemple pour votre service d'inversion :

1. le premier échange est l'envoi par le service de « Tapez une chaîne de caractères » au client
2. le client affiche cette invite à l'écran, saisie au clavier un texte et l'envoie au service
3. le service envoie le texte inversé et ferme la connexion
4. le client affiche le texte inversé et ferme la connexion

Dans tous les cas, le premier échange se fait dans le sens service → client
Dans le cas présent, la fin des échanges s'est faite à l'initiative du service

Exercice 3 : BTTP2.0

Les réceptions via un bufferedReader.readLine() ne permettent pas d'envoyer une String contenant des \n (readline() s'arrête au premier). On adoptera donc l'évolution suivante pour définir un client bttp2.0 : le message sera codé à l'envoi et décodé à la réception, en remplaçant tout \n par ## (la classe String contient une méthode faisant cela directement).
Le codage-décodage devra donner lieu à une bibliothèque bttp2 contenant une seule classe Codage avec deux méthodes statiques coder et décoder. Cette bibliothèque devra être présente coté client comme coté serveur.