TP noté

— L'évaluation se fera obligatoirement sur Windows, on utilisant le compte d'examen dont les coordonnées vous ont été fournies!

— Les fichiers .java demandés sont à rendre sur e-campus. Vous vous attacherez à ce que ces fichiers compilent parfaitement, et que les tests puissent être appliqués.

— Les deux dernières questions ont un barème très faible par rapport au travail associé. Attachez vous surtout à répondre parfaitement aux questions précédentes.

1 Analyse d'URL

L'objectif de ce TP noté est de définir un outil de lecture et d'analyse d'URL, de la plus grande qualité possible. Vous rédigerez les tests unitaires pour chaque méthode dans une classe `JUnit`.

On accède le plus souvent aux pages html avec une adresse très simple ainsi : `schema://hote` mais la syntaxe complète d'une URL est la suivante :
`schema://login:passphrase@hote:port/chemin?questions#fragment` où `question` consiste en une succession d'affectations `var=valeur` séparés par des &.

1. Créez deux exceptions, une exception `PasDUtilisateurException` et une `PasDeSchemaException`. La seconde héritera de `IllegalArgumentException`.
2. Créez une classe `URL` qui aura 5 attributs de type `String` (schema, login, passphrase, hote, chemin) et un attribut `Integer` (port) pour commencer. Générez les getters de ces différents attributs. Définissez un constructeur sans argument qui ne fait rien. Générez une classe de tests JUnit que vous completerez au fur et à mesure.

Nous allons maintenant définir un certain nombre de méthodes qui vont servir à extraire les différentes parties d'une URL. Les méthodes traiteront des portions de plus en plus grandes de l'URL, afin de terminer par la méthode qui traitera l'ensemble de l'URL. Vous utiliserez plusieurs méthode de la classes `String`, et en particulier les méthodes suivantes que vous retrouverez dans la javadoc `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html`

— `indexOf` permet de récupérer la position du premier caractère en paramètre.

— `charAt` permet de récupérer le caractère à la position en paramètre.

— `substring` qui prend en paramètre un ou deux entiers (le début et éventuellement la fin) et extrait le bout de chaîne entre les deux (fin exclue)

— `split` prend en paramètre une chaîne de caractère (lue comme une expression régulière) et retourne dans un tableau chacun des morceaux de chaînes séparés par cette chaîne.

3. Nous allons tout d'abord traiter les paramètres du serveur. Écrivez le test et la méthode `analyseHote` qui prendra en entrée une chaîne de type `hote:port` ou simplement `hote` et met à jour les attributs `hote` et `port` de l'objet. Quand aucun port n'est précisé, l'attribut port contiendra `null`. Vérifiez bien toutes ces conditions dans les tests unitaires.
4. On veut maintenant traiter toute l'adresse (de type `hote:port/chemin`). Écrivez les tests unitaires et une méthode `analyseAdresse` qui prend en paramètre une chaîne, en extrait le chemin, et appelle la méthode écrite précédemment pour récupérer l'hôte et le port. Le chemin commence au premier caractère / et termine à la fin de la chaîne, y compris si d'autres caractères / sont présent dans la chaîne. Si le chemin est vide ou s'il n'y a pas de caractère /, on placera une chaîne vide (et pas `null`) dans l'attribut correspondant.

5. De la même façon, écrivez les tests et la méthode `analyseUtilisateur` qui prend en argument une chaîne constituée d'un login et potentiellement d'une passphrase et insère les bonnes valeurs dans les champs correspondant. On différenciera bien le cas où la passphrase est vide de celui où il n'y a pas de passphrase fournie (null).
6. Écrivez les tests et la méthode `analyseRessource` qui prend en paramètre une URL sans la partie schema, et qui en extrait la partie utilisateur pour appeler la méthode `analyseUtilisateur` sur la partie utilisateur, la méthode `analyseAdresse` sur la partie adresse. En l'absence d'utilisateur spécifié, la méthode levera l'exception `PasDUtilisateurException` définie précédemment.
7. Écrivez les tests et la méthode `analyseURL` qui prend en paramètre une adresse, en extrait le schéma, et appelle `analyseRessource` pour analyser le reste de la chaîne. Si la méthode ne trouve pas de schéma, elle lève une exception `PasDeSchemaException` qui ne sera pas attrapée. Si la méthode `analyseRessource` lève l'exception `PasDUtilisateurException`, la méthode l'attrape, définit le login et la passphrase à `null`, puis continue l'analyse sur l'adresse. Ajoutez un constructeur prenant un paramètre une chaine de caractère et initialisant l'URL en fonction de cette adresse.
8. Les schémas ne sont pas libres, en principe. Créez une énumération `Schema` contenant au moins les schema http, https, ftp, ftps. Ajoutez une méthode de classe à l'énumération qui permet de récupérer le schema correspondant à une chaîne. Elle lèvera une nouvelle exception `SchemaIllegalException` en cas de schéma non reconnu. Mettez à jour votre classe URL et les tests pour que l'énumération soit utilisée.
9. La passphrase est susceptible de contenir les caractères ' :' et '@'. Assurez vous que vos méthodes gèrent bien ces situations en complétant vos tests, et en corrigeant éventuellement vos méthodes.
10. Enfin, on souhaiterait gérer le chemin un peu plus finement, en traitant notamment le schéma `chemin?questions#frag` sachant que les questions ont la forme `var1=val1`, et que plusieurs questions peuvent être enchaînées séparées par des `&`. Vous stockerez les questions dans une `LinkedHashMap` dont les clés sont les noms des variables. Écrivez les tests et la méthode pour traiter les chemins.