420-D04-SU
Programmation orientée objets I (java)
AEC Programmation et technologies internet
AEC Programmation, réseaux et télécommunications
AEC Intégration de systèmes d'information

Pondération: 40 % de la note finale Durée : 4h00

Nom : _____ Date : _____

Groupe : _____

Instructions :

Le travail DOIT se faire INDIVIDUELLEMENT
Documentation et notes de cours permises.
L'utilisation de l'ordinateur et de tous les outils est autorisée.
Aucune communication autorisée avec une autre personne de quelque manière que ce soit.
Bien indiquer votre nom sur le document.
L'application livrée, même incomplète, doit être fonctionnelle et prête à être exécuté, dans le cas contraire, le travail ne sera pas évalué.

Plagiat :

Toute constatation de plagiat entraînera une note de 0.

Préparation :

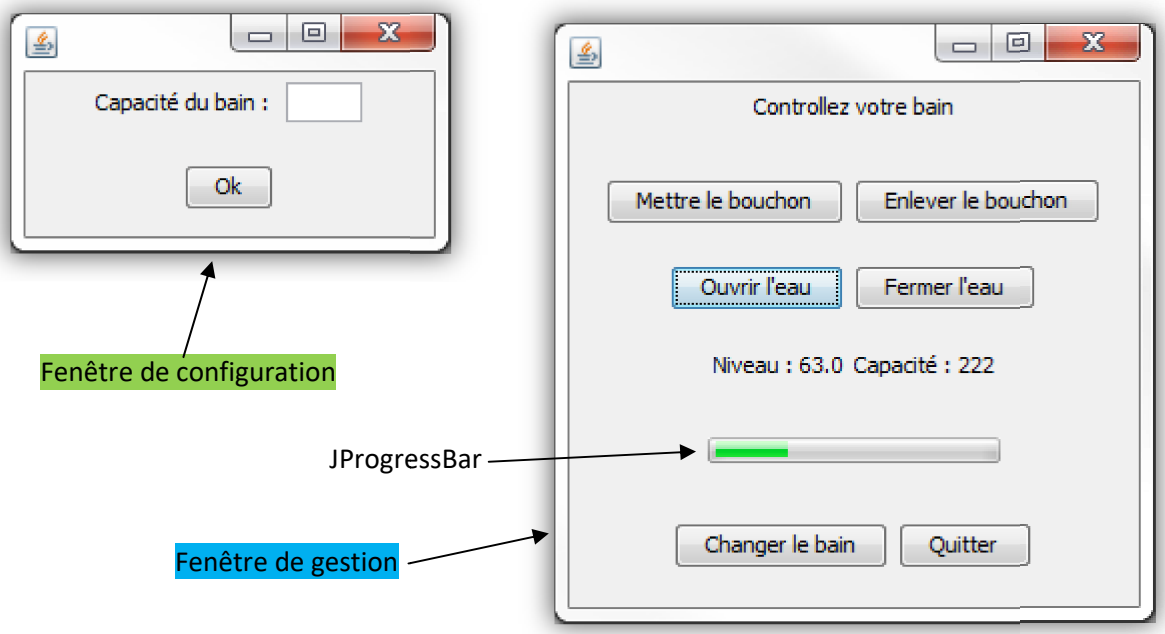
Importez dans eclipse le projet ***prjExamenPratiqueVotreNom*** (remplacez ***VotreNom*** par votre nom…) qui se trouve compressé dans Moodle à la section *Examen pratique final*.
Lisez bien les instructions jusqu'à la fin avant de commencer à développer.

Instructions :

Nous allons simuler une salle de bain qui utilise la domotique.
Nous avons donc un programme qui nous permet de contrôler le bain (ouvrir et fermer l'eau, mettre et enlever le bouchon du bain).
Quand on ouvre l'eau, le bain se remplit si le bouchon est en place, le niveau augmente petit à petit (cela a lieu dans un thread).
Dès que le bain est plein (niveau égal à la capacité) l'eau s'arrête pour éviter un débordement et le système envoie un message par socket pour avertir que le bain est prêt.
Nous aurons donc une autre application (application console) qui attend de recevoir une connexion pour avertir l'utilisateur (imaginons que c'est sur son téléphone cellulaire, par exemple).

Directives techniques et spécifications :

Dans le package **com.isi.prjexamenpratique.views**, vous trouverez 2 classes qu'il vous faudra compléter.
Ces 2 classes permettent l'affichage de 2 fenêtres comme celles-ci :



La première fenêtre lancée est la fenêtre de configuration. Elle permet de spécifier la capacité du bain. Sur l'appui du bouton *OK*, elle instancie le bain et la salle de bain, puis affiche la fenêtre de gestion en lui fournissant la salle de bain.

Dans la fenêtre de gestion, une **JProgressBar** affiche le niveau au fur et à mesure que le bain se vide et se remplit. Le bouton Changer le bain permet d'installer un nouveau bain en repassant par la fenêtre de configuration.

ATTENTION : pour simplifier, nous n'aurons ici que 2 packages. Un pour la vue (les 2 fenêtres précédentes) et un autre pour le modèle (les 6 classes et l'interface suivantes). La vue et le modèle communiquent directement. *Il n'y aura pas de contrôleur*.

Vous devez créer les 6 classes et l'interface suivantes dans le package **com.isi.prjexamenpratique.models** :

Interface : **NiveauListener**

1 méthode :

- niveauChange(**NiveauEvent**) invoquée par le setter de niveau du bain (1 méthode).

Classe : **InvalideCapaciteException**

(qui hérite de **Exception**)

1 méthode :

- toString() : retourne un message indiquant les limites pour une capacité correcte (1 méthode).

Classe : **Bain**

(qui est un thread ou implémente Runnable, selon votre préférence)

5 attributs :

- capacité : entre 125 et 225 litres, sinon une exception **InvalideCapaciteException** est lancée.
- niveau : pour la quantité d'eau présente dans le bain à un moment donné.
- bouchonEnPlace : si faux, le niveau diminue de 1,3 litres par demi-seconde.
- robinetOuvert : si vrai, le niveau augmente de 1 litre par demi-seconde.
- ArrayList d'objets **NiveauListener** : seront notifiés à chaque fois que le niveau change.

12 méthodes :

- constructeur paramétré pouvant lancer une exception **InvalideCapaciteException** (1 méthode).
- getters pour tous les membres sauf pour la liste d'écouteurs (4 méthodes).
- setter (private) pour le niveau (c'est là qu'on notifie les écouteurs) (1 méthode).
- ouvrir et fermer le robinet d'arrivée d'eau (2 méthodes).
- mettre et enlever le bouchon (2 méthodes).
- addNiveauListener : qui attend un objet de type **NiveauListener** (1 méthode).
- run : méthode redéfinie de Thread ou de Runnable (1 méthode).

Classe : **Salle de bain**

1 attribut :

- bain

3 méthodes :

- constructeur paramétré (1 méthode).
- getter et setter (2 méthodes).

Classe : **NiveauEvent**

(paramètre de la méthode niveauChange de l'interface **NiveauListener**)

1 attribut :

- bain

2 méthodes :

- constructeur paramétré (1 méthode).
- getSource : qui retourne le bain (1 méthode).

Classe : **NotificateurBain**

1 méthode :

- un main qui crée un server socket et attend de se faire informer que le bain est prêt. Ce programme affiche que le bain est prêt dans la console (1 méthode).

Classe : **BainPleinListener**

(qui implémente l'interface **NiveauListener**, cette classe est responsable d'envoyer un message par socket au programme NotificateurBain pour lui signifier que le bain est plein)

1 méthode :

- niveauChange(**NiveauEvent**) (1 méthode).

Le modèle sera constitué de **7 fichiers** contenant chacun un seul type (une classe ou une interface).
La vue sera constituée des **2 fichiers** proposés mis à jour pour répondre à l'énoncé.
Il n'y a pas de classe interne ici.

Description des comportements des classes :

Classe Bain

- le constructeur vérifie que la capacité donnée est correcte, sinon il envoie une exception **InvalideCapaciteException**.
- quand le niveau du bain varie, il avertit ses écouteurs.
- dans cette application, un bain aura 2 écouteurs :
  - une instance de **FenetreBain** : cette instance est avertie que le niveau du bain a changé quand sa méthode NiveauChange est appelée. À ce moment là, elle met à jour sa barre de progression et le label indiquant le niveau du bain.
  - une instance de **BainPleinListener** : quand cette instance est avertie que le niveau du bain a changé, elle vérifie si le bain est plein. Si c'est le cas, elle se connecte au serveur de la classe **NotificateurBain**. Sinon, elle ne fait rien.
- la méthode run du bain fait varier le niveau du bain selon l'état de ses attributs bouchonEnPlace et robinetOuvert.

Classe FenetreConfiguration

- ajouter un écouteur sur le bouton OK.
- dans la méthode d'interface de l'écouteur (actionPerformed) :
  - on crée les instances de **SalleDeBain** et de **Bain**.
  - on crée l'instance de **FenêtreBain** ***en lui donnant la salle de bain***.
  - on cache la fenêtre de configuration.

Classe FenetreBain

- cette classe connaît le **bain** sur lequel elle agit.
- mettre des écouteurs sur les 4 boutons qui agissent sur l'état du Bain (attributs bouchonEnPlace et robinetOuvert).
- la ou les méthodes associée(s) aux boutons (actionPerformed) appelle(nt) la bonne méthode sur le bain selon le bouton appuyé.

Conseils

Procédez par étape et complétez chaque étape avant de passer à la suivante.

1. créer la classe **Bain** qui hérite de Thread ou implémente Runnable (sans exception, ni évènement).
2. créer la classe **SalleDeBain**.
3. ajouter la gestion d'événement sur les composants graphiques en implémentant ActionListener sur les 2 classes **FenetreConfiguration** et **FenetreBain**.
4. ajouter la salle de bain à l'interface graphique. Sur les événements, invoquer les méthodes pertinentes du bain. S'assurer que le tout fonctionne jusqu'ici et faire une copie de sécurité du projet.
5. ajouter l'exception: créer la classe d'exception et modifier la classe **Bain** pour déclencher cette exception. Modifier en conséquence le code qui instancie le bain pour tenir compte de l'exception.
6. ajouter l'événement: créer l'interface **NiveauListener** et la classe **NiveauEvent**. Implémenter cette interface dans la vue, restructurer la classe **Bain** pour déclencher l'événement. Ajuster la **JProgressBar** sur le déclenchement de cet événement, ainsi que le **JLabel** indiquant le niveau du bain.
7. ajouter la partie réseau: créer l'application console **NotificateurBain** qui écoute avec un **serverSocket**. Créer la classe **BainPleinListener** qui écoute l'événement du bain et se connecte à l'application console **NotificateurBain** dès que le bain est plein et lui envoie un message.

Remise:

Soumettez votre examen dans BitBucket, comme un exercice habituel.

Évaluation:

L'évaluation aura lieu selon les critères suivants :

critères	pts	commentaires
1- Livraison:		
Respect des consignes techniques	1	
Application compile et s'exécute	1	
2- Classes:		
Structure du projet et packages	1	
Vues	1	
Classes d'entités	2	
Thread	2	
Gestion des événements	2	
Ajout de l'exception personnalisée	1	
Ajout de l'événement personnalisé	2	
Notification par socket	2	
Possibilité de changer le bain	1	
Programme serveur	1	
3- Autres:		
Propreté du code	2	
Validations	1	

Total _____ / 20