Programmation Orientée Objet
IG2I, LE3 – 2016-2017

CTP – 26 Janvier 2017

Diego Cattaruzza, Philippe Kubiak, Maxime Ogier

Conditions de l'examen

- **durée** : 2h30 ;
- ordinateurs personnels autorisés ;
- documents et pages internet autorisés ;
- accès à Moodle autorisé ;
- les moyens de communication (mails, messageries instantanées, Facebook, Google Drive, Tweeter, téléphones portables, etc.) sont INTERDITS.

Rendu

Vous déposez sur Moodle, dans la zone de dépôt **Dépôt du CTP 2016/2017** une archive compressée de votre répertoire de projet. Cette archive doit être nommée **CTP_Nom_Prenom.zip** avec votre nom et votre prénom. Un nommage différent donnera lieu à deux points de pénalisation. Cette règle est également valable pour toute proposition de nommage de ce CTP : projet, paquetages, classes, méthodes, attributs, ...

Vérifiez bien que les fichiers sources que vous avez réalisés lors du CTP sont présents dans le dossier que vous déposez sur Moodle (oui, vérifiez bien cela : toutes les années il y a des répertoires vides ou contenant les mauvais fichiers ! Il ne sera pas possible de renvoyer une nouvelle version de votre projet après la date limite de dépôt).

Notation

Cette épreuve de CTP est évaluée sur 20 points. En plus de ces 20 points, il y a 4 points bonus sur la dernière partie.

Pour chaque question, la moitié des points est attribuée si le code fonctionne correctement. **L'autre moitié des points est attribuée en fonction de la qualité du code** (respect des principes de la programmation orientée objet, efficacité des algorithmes) et de la présentation du code (indentation correcte, respects des conventions Java dans les noms de classe, attributs, méthodes et variables, commentaires au format Javadoc).

Consignes

Le sujet est composé de 8 parties liées entre elles. Vous devez d'abord traiter dans l'ordre les parties 1 à 6. Puis vous pouvez ensuite traiter les parties 7 et 8 dans l'ordre que vous voulez (7 avant 8 ou 8 avant 7).

Vous êtes bien sûr autorisés à rajouter les attributs ou méthodes que vous jugez nécessaires même si ce n'est pas dit explicitement dans le sujet.

Sujet

Le Docteur Diet propose à ses patients des formules depuis de nombreuses années. Mais ses patients n'en sont pas très satisfait. Ainsi, il souhaite disposer d'un outil informatique pour l'aider à parfaire ses prescriptions de régime.

Le Docteur Diet a pré-sélectionné un ensemble d'aliments qui peuvent entrer dans le menu de régime. Par exemple : ananas, bœuf, chocolat, dinde, épinard, fêta, gruyère, haricots.

Le Docteur Diet a également réalisé une étude afin de mesurer la satisfaction apportée par le fait de manger $100g$ de chacun des aliments du régime. La satisfaction est une fonction linéaire par rapport à la quantité consommée, et si on consomme différents aliments, il suffit de faire la somme des satisfactions apportées par ces aliments. Par ailleurs, le docteur a récolté des données sur les apports énergétiques ainsi que le calcium, la vitamine C et le fer pour chacun des aliments. Un exemple de ces données est présenté dans la Table 1.

	ananas	bœuf	chocolat	dinde	épinard	fêta	gruyère	haricot
satisfaction	12	20	45	11	3	8	16	7
calories ($kcal$)	50	250	600	120	25	260	400	30
calcium (mg)	13	18	56	14	99	490	1000	37
vitamine C (mg)	48	0	0	0	28	0	0	16
fer (μg)	300	2600	8000	1000	2700	700	300	1000

Table 1: Satisfaction et informations nutritionnelles pour $100g$ d'aliment.

Afin de mettre au point son régime parfait, le Docteur Diet souhaite déterminer la quantité de chacun des aliments de manière à maximiser la satisfaction totale apportée par la consommation de ces aliments. Cependant, pour que le menu soit qualifié de régime, pour chaque patient il faut respecter un nombre maximum de calories, un apport en calcium minimal, un apport en vitamines C minimal, et un apport en fer minimal.

Dans ce sujet de CTP, on propose d'implémenter une interface graphique qui va permettre d'aider le Docteur Diet à faire à ses patients les meilleures prescriptions possibles. Ceci nécessitera la mise en place d'une base de données, du modèle objet associé, et des requêtes sur la base de données ; ainsi que la création d'une interface graphique dans laquelle nous utiliserons des composants de type **JPanel** et **JSlider**.

Le modèle conceptuel de données ainsi que le modèle logique de données de notre application sont présentés dans les Figures 1 et 2 respectivement.

Le modèle logique de données contient trois tables : **Patient**, **Aliment**, et **Prescription**. Un *patient* représente très simplement un patient du Docteur Diet, défini par son nom et prénom et ses caractéristiques pour le régime. Un *aliment* est défini par son nom et ses caractéristiques sur la satisfaction, les calories, le calcium, les vitamines C et le fer. Ces caractéristiques sont données

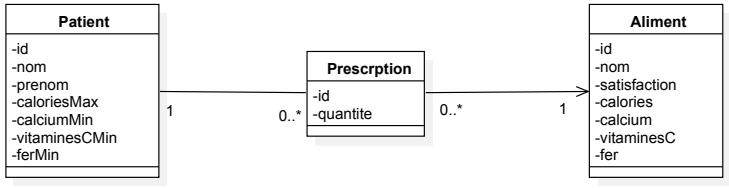

Figure 1: Modèle conceptuel de données pour l'application du Docteur Diet.

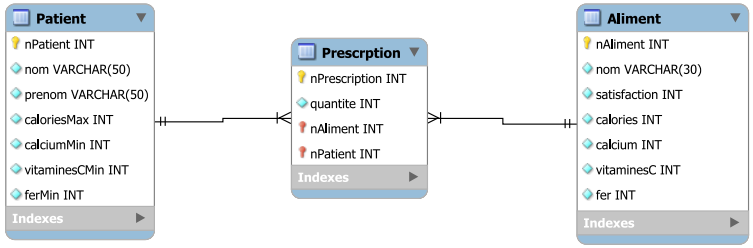

Figure 2: Modèle logique de données pour l'application du Docteur Diet.

pour 100 g d'aliment. Enfin, une *prescription* correspond à l'association d'un aliment à un patient, avec une quantité qui représente la quantité de l'aliment qui est prescrite au patient. Cette quantité est un nombre entier et correspond au nombre de centaines de grammes (ainsi, une quantité de 1 représente 100 g, et une quantité de 10 représente 1 kg).

1 Base de données et connexion avec l'application (1 point)

Question 1. En utilisant le script SQL *derbyRegime.sql* disponible sur Moodle, mettez en place la base de données pour l'application du Docteur Diet. Le script est prévu pour un SGBD Derby, donc le plus simple est de créer votre base de données sur Netbeans, comme vu dans le TP2.

Question 2. Créez un nouveau projet nommé **CTP_Nom_Prenom** avec votre nom et votre prénom. Ajoutez un paquetage **metier** et une classe **RequeteRegime**. Dans cette classe, ajoutez un attribut de type **Connection** et faites en sorte que la connexion à la base de donnée soit établie lors de l'appel au constructeur par défaut. Testez le bon établissement de la connexion à la base de données.

Remarque : n'oubliez pas d'importer le pilote JDBC dans les librairies du projet.

2 Mise en place du modèle objet (2 points)

À présent, nous souhaitons ajouter les classes **Patient**, **Aliment** et **Prescription** pour le modèle objet. Vous pouvez vous référer à la Figure 1 qui présente le modèle conceptuel de données.

Question 3. Ajoutez à votre projet un paquetage **modele**. Ajoutez dans ce paquetage une classe **Patient**, avec ses attributs, un constructeur par données, les accesseurs et mutateurs nécessaires, ainsi qu'une redéfinition de la méthode **toString**. Vérifiez le bon fonctionnement de cette classe.

Question 4. Ajoutez dans le paquetage **modele** une classe **Aliment**, avec ses attributs, un constructeur par données, les accesseurs et mutateurs nécessaires, ainsi qu'une redéfinition de la méthode **toString**. Vérifiez le bon fonctionnement de cette classe.

Question 5. Ajoutez dans le paquetage **modele** une classe **Prescription**, avec ses attributs (en particulier on gardera une référence sur l'aliment et le patient liés à la prescription), un constructeur par données, les accesseurs et mutateurs nécessaires, ainsi qu'une redéfinition de la méthode **toString**. Vérifiez le bon fonctionnement de cette classe.

3 Requêtes sur la base de données (3 points)

Question 6. Dans la classe **RequeteRegime** ajoutez une méthode **public List<Aliment> ensAliments() throws SQLException** qui renvoie la liste des aliments. Testez le bon fonctionnement de cette méthode.

Question 7. Dans la classe **RequeteRegime** ajoutez une méthode **public List<Patient> ensPatients(String nomPrenom) throws SQLException** qui renvoie la liste des patients contentant la chaîne de caractères **nomPrenom** dans leur nom *ou* dans leur prénom (pas de sensibilité à la casse de la chaîne de caractères donnée en paramètre). Testez le bon fonctionnement de cette méthode.

Question 8. On souhaite à présent ajouter à un patient l'ensemble des prescriptions qui lui sont associées. Dans la classe **RequeteRegime** ajoutez une méthode **public void addAllPrescriptionsToPatient (Patient p) throws SQLException** qui ajoute au patient **p** l'ensemble des ses prescriptions. Bien évidemment, vous pouvez ajoutez les attributs et méthodes que vous jugez nécessaires dans la classe **Patient**. Testez le bon fonctionnement de cette méthode.

4 Affichage de la page d'accueil (1 point)

On souhaite à présent ajouter à notre application une première fenêtre qui sera l'accueil de notre application. Sur cette fenêtre, on placera :

- une **JList** qui contient les noms et prénoms des patients ;
- un **JLabel** avec le texte "*Recherche :* " ;
- un **JTextField** dans lequel le texte rentré permettra de filtrer les clients affichés dans la liste : leur nom ou prénom devra contenir le texte rentré dans le **JTextField** ;
- un **JButton** qui permettra par la suite d'accéder à la fiche de régime du patient sélectionné dans la liste.

Un exemple de cette fenêtre est donné dans la Figure 3.

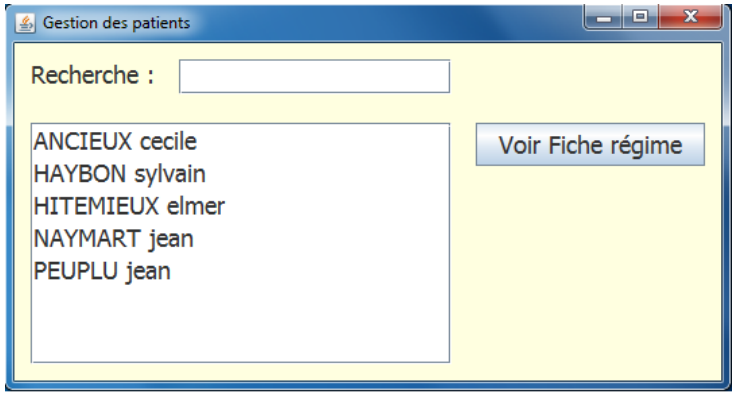

Figure 3: Exemple d'affichage de la page d'accueil.

Question 9. Ajoutez à votre projet un paquetage **vuecontrole**. Ajoutez dans ce paquetage une fenêtre **Accueil** qui sera similaire à celle présentée dans la Figure 3. À l'ouverture de cette fenêtre, l'ensemble des patients sera affiché dans la liste. Ajoutez la logique de contrôle de telle sorte que lorsque le texte dans la zone de texte est modifié, la liste des patients est mise à jour (seulement les patients contenant le texte dans le nom ou le prénom sont affichés). Vérifiez le bon fonctionnement de cette fenêtre.

5 Affichage de la fiche de régime d'un patient (8 points)

On souhaite à présent mettre en place une fenêtre qui permettra de définir le régime d'un patient. Cette fenêtre sera ouverte lors de l'appui sur le bouton "*Voir Fiche régime*" dans la fenêtre d'accueil pour le patient sélectionné dans la liste. Sur cette fenêtre, on placera :

- au moins 8 **JPanel** : un pour chaque prescription du patient et sur lequel sera affiché le nom de l'aliment ;
- sur chaque **JPanel**, un **JSlider** qui permettra de définir la quantité de l'aliment dans la prescription (en centaine de grammes) ;
- une zone de texte non éditable qui présente les indicateurs du régime (calories, calcium, vitamines C, fer et satisfaction) ;
- 2 boutons : un pour mettre à jour le régime dans la base de données, et un pour déterminer un régime optimal.

Un exemple de cette fenêtre est donné dans la Figure 4.

Question 10. Dans le paquetage **vuecontrole**, ajoutez une fenêtre nommée **Regime**, et ajoutez-y les composants graphiques de telle sorte que cette fenêtre ressemble à celle de la Figure 4 (pour le moment, ne cherchez pas à afficher de texte sur les objets **JPanel** ou **JSldier**). Pour la construction des **JPanel**, vous pouvez procéder de la manière suivante :

- ajouter un **JPanel** sur la fenêtre ;
- ajouter un **JSlider** *dans* ce **JPanel** (vous pouvez vérifier dans la fenêtre *Navigateur* de Netbeans (*Ctrl* + 7) que le **JSlider** est bien un sous-composant du **JPanel**) ;
- faire un copier-coller du **JPanel** autant de fois que nécessaire (au moins 8), et vérifiez bien (dans la fenêtre *Navigateur* de Netbeans) que pour chaque **JPanel** il y a un sous-composant de type **JSlider**.

Question 11. Dans la classe **Regime**, ajoutez un attribut de type **Patient** qui représente le patient pour lequel on définit le régime. Modifier le constructeur de telle sorte que le patient soit passé en paramètre du constructeur et que l'attribut soit initialisé. Faites également en sorte que toutes les prescriptions soient ajoutées au patient. Ajoutez également une méthode **initFenetre** qui sera appelée dans le constructeur. Cette méthode définit l'emplacement de la fenêtre, son titre (avec le nom du patient), sa couleur de fond, et rend la fenêtre visible. Dans la classe **Accueil**, faites en sorte que lors du clic sur le bouton "*Voir Fiche Régime*", on ouvre une fenêtre **Regime** pour le patient sélectionné dans la liste de la fenêtre **Accueil**. Testez le bon fonctionnement.

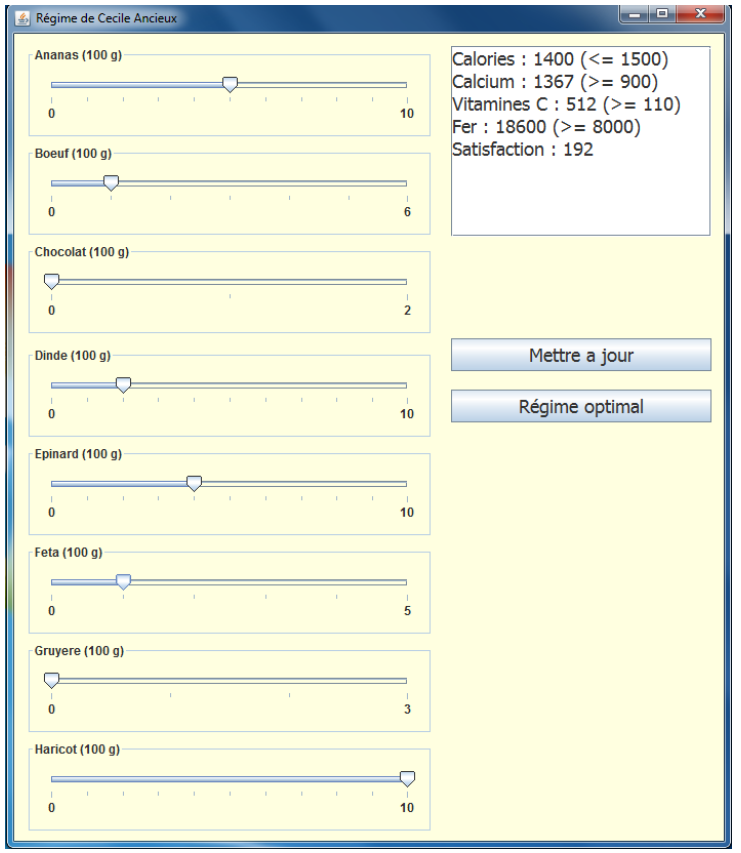

Figure 4: Exemple d'affichage de la fiche régime d'un patient.

Question 12. Comme on ne connait pas exactement à l'avance le nombre de **JPanel** et à quel prescription ils sont liés, nous allons faire cela de manière dynamique. Pour cela :

- ajoutez dans la classe **Regime** un attribut nommé **panelCollection** qui contiendra tous les **JPanel** ;
- ajoutez une méthode **memoriserPanels** qui ajoute tous les **JPanel** présents dans la fenêtre à **panelCollection**, et les trie selon leur ordonnée croissante (on peut également en profiter pour leur donner une couleur de fond) ;

- faites appel à cette méthode **memoriserPanels** dans la méthode **initFenetre**.

Remarquez que vous pouvez avoir accès à tous les composants présents dans la fenêtre en faisant appel à la méthode **this.getContentPane().getComponents()**. Testez que tout fonctionne correctement.

Nous souhaitons à présent initialiser les composants **JPanel** pour chaque prescription du patient en mettant en titre le nom de l'aliment lié à la prescription.

Pour modifier le titre d'un **JPanel**, on peut utiliser la méthode **setBorder(Border b)**. Pour créer un objet de type **Border** avec un titre, on peut utiliser la méthode statique **BorderFactory.createTitledBorder(String titreBordure)** où **titreBordure** est le titre de la bordure.

Question 13. À présent :

- ajoutez une méthode **initPanelsEtSliders** (même si pour le moment on ne fait que l'initialisation des **JPanel**) ;
- implémentez cette méthode de telle sorte que, pour chaque prescription du patient, le nom de l'aliment soit affiché dans le titre du **JPanel** (suivi de la chaîne de caractères "*(100g)*") ;
- faites en sorte que cette méthode rende non visibles les **JPanel** qui n'ont pas été initialisés (dans le cas où il y a plus de **JPanel** que de prescriptions) ;
- faites appel à cette méthode dans la méthode **initFenetre**.

Testez que tout fonctionne correctement (les *labels* sur les **JPanel** sont corrects).

Nous souhaitons à présent initialiser les composants **JSlider** pour chaque prescription du patient. Nous allons dans un premier temps nous intéresser à ce qui concerne le style d'un **JSlider**, puis nous nous focaliserons sur les valeurs (minimum, maximum et valeur courante) à affecter à un **JSlider** lié à une prescription.

Voici quelques méthodes utiles pour mettre à jour le style d'un **JSlider** :

- **setMinorTickSpacing** permet de définir l'espace entre les marqueurs mineurs (on peut définir un espace de 1 unité) ;
- **setPaintLabels** et **setPaintTicks** permettent de définir si les labels et les marqueurs sont affichés ;
- **setSnapToTicks** permet de forcer le curseur à être sur un des marqueurs (un marqueur mineur) ;
- **setBackground** permet de définir la couleur de fond.

Notez que le curseur devra toujours être sur un nombre entier (et que la distance entre les marqueurs mineurs est de 1).

Question 14. Ajoutez dans la classe **Regime** une méthode **setStyleSlider (JSlider s)**. Cette méthode doit mettre à jour le style du **JSlider s** (voir les méthodes proposées ci-dessus).

Nous souhaitons à présent nous intéresser aux valeurs (minimum, maximum et valeur courante) à affecter à un **JSlider** ainsi qu'à l'espace entre les marqueurs majeurs (on mettra la valeur maximale du **JSlider**), en lien avec une prescription.

Voici quelques méthodes utiles que vous pouvez utiliser :

- **setMinimum**, **setMaximum** et **setValue** permettent de définir respectivement les valeurs minimale, maximale et la valeur courante (le curseur) du **JSlider** ;
- **setMajorTickSpacing** permet de définir l'espace entre les marqueurs majeurs.

Par ailleurs, notez bien que pour chaque aliment, la quantité est définie en centaine de grammes (1 équivaut à $100g$, et 10 équivaut à $1000g$), et que cette quantité doit toujours être un nombre entier. Par ailleurs, la quantité d'un aliment ne doit pas dépasser $1kg$ (donc 10). On peut également calculer une borne sur la quantité maximale pour un aliment en faisant le ratio du nombre de calories maximales pour le patient divisé par le nombre de calories pour $100g$ de l'aliment.

Question 15. Ajoutez dans la classe **Regime** une méthode **setValeursSlider (JSlider s, Prescription p)**. Cette méthode doit mettre à jour les valeurs du **JSlider s** ainsi que l'espace entre les marqueurs majeurs en fonction de la prescription **p** (voir les méthodes proposées ci-dessus). Bien évidemment, vous pouvez ajoutez les méthodes que vous jugez nécessaires dans la classe **Prescription**.

Pour récupérer un composant qui est sur un **Jpanel**, on peut faire appel à la méthode **getComponent(int i)** qui renvoie le $i^{ème}$ composant du **JPanel**. S'il n'y a qu'un seul composant, **getComponent(0)** renvoie ce composant.

Question 16. Modifiez la méthode **initPanelsEtSliders** afin de mettre à jour le **JSlider** correspondant à chacune des prescriptions. Vous veillerez à mettre à

jour le style ainsi que les valeurs de chaque **JSlider**. Testez que tout fonctionne correctement.

Nous souhaitons à présent mettre à jour la zone de texte qui présente l'ensemble des indicateurs du régime (calories, calcium, vitamines C, fer et satisfaction).

Question 17. Ajoutez dans la classe **Regime** une méthode **updateIndicateurs** qui met à jour la zone de texte avec les indicateurs sur le régime pour le patient (vous pouvez vous référer à la Figure 4 pour voir un exemple). Faites appel à cette méthode à la fin de la méthode **initFenetre**. Vérifiez que tout fonctionne correctement.

6 Logique de contrôle sur la fiche de régime (3 points)

On souhaite à présent faire en sorte que chaque déplacement d'un **JSlider** entraîne la mise à jour de la prescription associée, et donc des informations sur les indicateurs dans la zone de texte.

Pour ce faire, il est nécessaire de connaître, pour chaque **JSlider** la prescription qui lui est associée. En effet, lors du déplacement du curseur d'un **JSlider**, il faut pouvoir mettre à jour la quantité dans la prescription associée. Notez bien qu'à chaque **JSlider** est associée une prescription *unique*.

Question 18. Ajoutez dans la classe **Regime** un attribut nommé **prescriptions** qui contiendra toutes les prescriptions affichées sur la fenêtre, et qui permettra de faire l'association entre un **JSlider** et la prescription correspondante. Modifiez en conséquence la méthode **initPanelsEtSliders** afin d'initialiser cet attribut avec toutes les prescriptions.

Question 19. Ajoutez dans la classe **Regime** une méthode **updatePrescription** qui prend en paramètre un objet de type **JSlider** et qui modifie la quantité de la prescription associée à ce **JSlider**. Ensuite, cette méthode doit mettre à jour l'affichage des indicateurs dans la zone de texte. Notez que la méthode **getValue()** de la classe **JSlider** permet de récupérer la valeur actuelle sur laquelle le curseur est positionné.

Question 20. Pour chacun des **JSlider**, ajoutez un écouteur sur l'événement **StateChanged** (déclenché dès que la valeur du curseur est modifiée). Pour chacun de ces écouteurs, faites en sorte que la prescription et les indicateurs dans la zone de texte soient mis à jour.

7 Mise à jour du régime du client (2 points)

Question 21. Faites en sorte que lorsque l'utilisateur appuie sur le bouton "*Mettre à jour*", toutes les prescriptions du patient soient mises à jour en fonction des valeurs courantes des curseurs (i.e. les quantités des aliments). On ne vérifie pas ici si ces prescriptions vérifient bien l'apport calorique maximal ou les apports minimaux en calcium, vitamines C et fer.

8 Optimisation du régime (4 points BONUS)

En fait, déterminer le régime du patient peut être vu comme un problème d'optimisation sous contraintes. Il s'agit de déterminer les quantités de chacun des aliments afin de maximiser la satisfaction totale apportée, tout en tenant compte de l'apport calorique maximal et des apports minimaux en calcium, vitamines C et fer.

Dans une première approche pour résoudre ce problème, vous pouvez vous baser sur un tri des aliments (selon un ratio que vous déterminez).

Question 22. Dans la classe **Patient**, ajoutez une méthode **regimeOptimal** qui met à jour les quantités des aliments dans les prescriptions de manière à avoir le meilleur régime possible. Même si la solution que vous obtenez n'est pas de très grande qualité, toute tentative d'implémentation de la méthode **regimeOptimal** sera considérée et valorisée. Dans la classe **Regime**, faites en sorte que lorsque l'utilisateur appuie sur le bouton "*Régime optimal*", ce régime optimal soit calculé, puis que les curseurs des **JSlider** et la zone de texte avec les indicateurs soient mis à jour en conséquence.