Examen Intra (solutionnaire) de LOG2420, Hiver 2011

Professeur: Michel Desmarais

7 mars 2011

- Date et durée de l'examen : 7 mars (2h30).
- L'examen est sur 20 points.
- Le questionnaire comporte 10 pages.
- La documentation et la calculatrice programmable ne sont pas permises.
- Répondre sur le cahier séparé.
- Pour les questions à développement, prenez soin d'exprimer clairement vos arguments, car la correction en tiendra compte.
- Ne pas remettre le questionnaire.

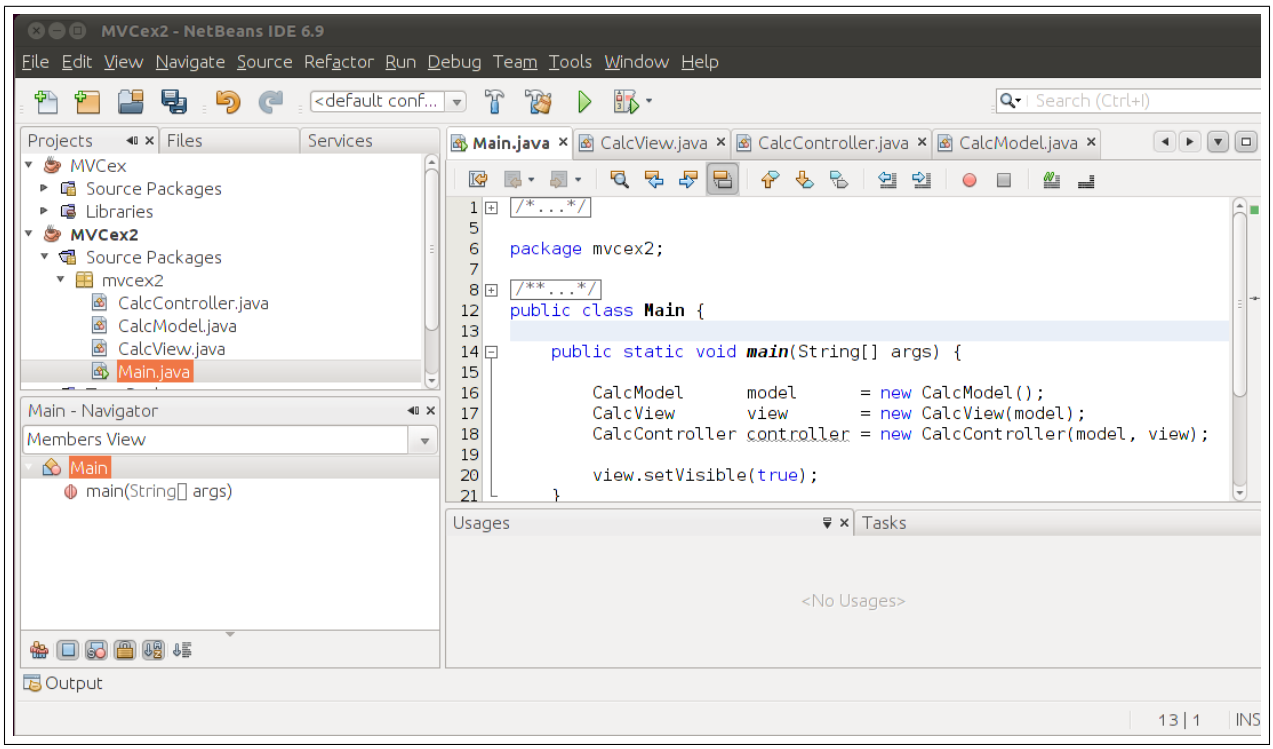
Réservé au correcteur :

Question:	1	2	3	4	5	Total
Points:	5	2	3	3	7	20
Score:						

Question 1 (5 points)

La fenêtre ci-dessous affiche l'application NetBeans utilisée dans les laboratoires. Référez-

vous à celle-ci pour répondre aux questions qui suivent.



(a) [4 points] Pour chacune des quatre heuristiques de Bastien et Scapin ci-dessous, décrivez en un paragraphe de 10 à 15 lignes comment elle est appliquée dans l'interface de Netbeans, en prenant des exemples, et quel impact elle a sur l'utilisabilité. À titre de rappel, l'annexe A fournit la liste des huit heuristiques de Bastien et Scapin (où l'on retrouve les quatre heuristiques ci-dessous).

(1) Guidage
(2) Charge de travail
(4) Adaptabilité
(8) Compatibilité

Solution:

- **Guidage** : il y a plusieurs types de guidage et plusieurs exemples dans l'interface. On peut mentionner le guidage par regroupements spatial et par similarité que l'on retrouve dans les boîtes à outil et dans la navigation. Par exemple les menus sont alignés dans un cadre et de police identique, les raccourcis sont indiqués par la lettre souligneé, les icones de boîte à outils sont alignés, de grosseur similaire et certains partagent des formes similaires pour indiquer un regroupement. Les icones en gris sont aussi une forme de guidage indiquant la non disponibilité. D'autres symboles, par exemple les petits triangles (►) et les signes encadrés (⊞) nous suggèrent la présence de fonctionnalité. Les boutons ou fichiers actifs sont mis en valeur par le relief, la densité visuelle ou la couleur.

- **Charge de travail** : ici encore il y a plusieurs exemples de diminution de la charge de travail. Les boîtes à outils constituent un raccourci pour la fonctionnalité de menu qui diminuent le nombre de cliques. Les menus sont eux mêmes utiles à diminuer la charge mnésique de la fonctionnalité. Les sous-fenêtres "Main–Navigator" et "Projects" fournissent des aides mémoire pour les fichiers pertinents et des objets du code. De plus, ces fenêtres offrent aussi un accès rapide à ces objets, diminuant la charge de travail en termes d'actions. Un autre élément important est la densité d'information : les commentaires repliés sur une ligne permettent l'affichage d'une plus grande quantité de code.

- **Adaptabilité** : les sous-fenêtres étant extensibles ceci permet une forme de personnalisation qui est considéré comme de l'adaptabilité. Comme on peut deviner de l'option "default conf..." on peut aussi présumer qu'il s'agit d'une fonction poru faciliter la personnalisation. Les différentes façons d'accéder à des parties du code, comme par la navigation "members view", la recherche, ou simplement par la barre de défilement, sont des exemples de flexibilité qui font partie de l'adaptabilité.

- **Compatibilité** : les changements de vues (affichages) qui se produisent lorsqu'on exécute un programme en mode deboggage ou normal sont des exemples d'affichage que l'on vise à rendre plus compatible avec la tâche. La compabilité avec les utilisateurs peut aussi être évoquée par le fait que les experts utiliseront les raccourcis alors que les autres prendront les menus. Finalement, on peut aussi évoquer que l'information pertinente à la tâche de modification, le déboggage ou de développement du code est normalement présente, ce qui constitue aussi un exemple de compatibilité, mais cet exemple est plus difficile à étayer et à bien justifier.

(b) [1 point] Une des barres d'outils de l'écran principal est reproduite ci-dessous. Expliquez comment les principes de Gestalt et de perception visuelle sont appliqués dans sa conception pour en améliorer l'utilisabilité (10–15 lignes).

Solution: On retrouve dans un premier temps l'alignement d'objets de même grosseur, ce qui suggère immédiatement un regroupement. Les barres de séparation qui ajoutent un espacement et sont des objets d'un autre type, permettent un second niveau de regroupement. La perte de contraste des icones en gris suggère l'éloignement et donc la non disponibilité des fonctions correspondantes. Finalement, plusieurs les icones sont composées de sous-objets (flèches et autres formes régulières), ce qui indique une forme de sous-regroupement et de spécialisation de la fonctionnalité dans ce regroupement par la similarité des formes et des effets de

symmétrie que l'oeil détecte rapidement.

Question 2 (2 points)

Expliquez la notion de *charge cognitive*. À quoi correspond la charge coghnitive et pourquoi est-elle importante en conception d'interface utilisateur? (1/2 page)

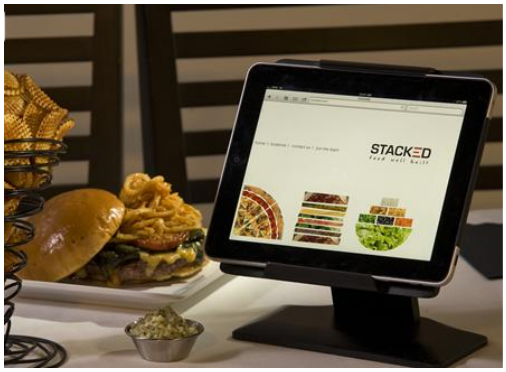
Solution: La charge cognitive est le travail mental nécessaire lors de la réalisation d'une tâche. Cette charge peut être très importante. Par exemple, le fait de devoir se souvenir du nom d'un fichier dans une hiérarchie profonde de répertoires peut s'avérer très difficile. Cette charge étant souvent trop grande, l'utilisateur devra recourir à une recherche, ce qui se traduira par un travail physique fastidieux pour réussir la tâche.

Les principaux facteurs de charge cognitive sont donc la mémorisation et le rappel, le calcul mental, la compréhension de termes, la recherche d'information dans une écran, etc. Bref, tout ce qui nécessite un rappel et un traitement de l'information par le cerveau. Plus la charge mentale est élevée, plus le temps et l'effort pour effectuer une tâche seront grands et plus les chances d'erreurs augmenteront. Une charge mentale trop forte se transforme parfois en une charge d'actions physiques et cognitives. Par exemple, si le rappel d'une information est infructueux, il faut alors rechercher cette information.

Notons finalement que la charge cognitive varie considérablement selon l'utilisateur. La charge pour l'utilisateur expert pourrait être pratiquement nulle alors que pour le néophyte elle pourrait être insurmontable. Elle change aussi avec l'expérience. Plus un utilisateur répète les tâches avec l'interface, plus la charge cognitive diminue.

Question 3 (3 points)

Dans le USA Today, Horovitz rapporte l'utilisation prochaine des iPad dans une chaine de restaurant. La chaine prévoit une centaine de iPads par restaurant. L'article souligne que le iPad a déjà été utilisé dans d'autres restaurants, notamment pour la carte des vins où les ventes auraient grimpées de 20%, et avance qu'un jour, les menus électroniques remplaceront nos menus papier.



Présumons que vous êtes responsable de la conception de l'interface des iPad pour la chaine qui compte l'installer dans une dizaine de restaurants.

(a) [2 points] Définissez deux *objectifs d'utilisabilité* pour l'interface du menu électronique et justifiez-les (par exemple en référant à des éléments du contexte d'utilisation).

Solution:

1. Tâche : commander un poulet du Général Tao avec une entrée de *Prosciuto e melone*. La tâche doit être effectuée par un nouvel utilisateur sans aucune familiarité avec le iPad et le menu en deça de 30 secondes, et en deça de 15 secondes pour un utilisateur qui aura pu explorer le menu durant 2 minutes au préalable.
2. Tâche : annuler la commande du poulet et la remplacer par un *Burger maison*. La tâche doit être effectuée en deça de 30 secondes par un nouvel utilisateur qui aura au préalable commandé le poulet.
3.

(b) [1 point] Joseph considère qu'il serait utile de fournir une aide en ligne accessible en tout temps. Marie considère au contraire que ce serait une mauvaise idée. Donnez les arguments que Joseph et Marie pourraient utiliser (1/2 page).

Solution: Très souvent, les décisions de conception de l'interface dépendent du contexte d'utilisation. Il s'agit ici de faire appel au contexte d'utilisation pour justifier des décisions de conception :

Marie : les utilisateurs du restaurant sont en majorité à la presse et n'auront pas la patience de consulter l'aide si un problème survient, ni même pour une première fois. Ils viennent pour manger. Si un problème survient ou qu'ils n'arrivent pas à commander, on doit leur signifier clairement que les serveurs sont là pour les aider ou pour prendre leur commande s'ils le désirent.

Joseph : certains utilisateurs seront curieux, surtout si c'est leur première utilisation de l'application ou de l'iPad. Ils seront donc heureux d'avoir une documentation à consulter. Même une fois commandé, certains utilisateurs voudront voir ce qu'ils peuvent faire avec l'appareil en attendant d'être servi. Ces utilisateurs seront alors mieux en mesure d'utiliser et d'aider leur amis à table avec eux les prochaines fois (sachant que le restaurant a une clientèle régulière importante).

Question 4 (3 points)

Nous avons discuté, durant le cours, des avantages du iPod et comment sa conception se démarquait de produits comme le baladeur Sanyo.

(a) [2 points] Décrivez en 1/2 page les décisions de conceptions les plus importantes du iPod et qui expliquent en grande partie son succès.

Solution: Les décisions de conception les plus importantes sont :

- séparation des fonctions : les fonctions complexe sont faites sur iTunes pour ne

laisser que les fonctions de lecture sur le iPod;

- seulement 5 boutons : 4 organisés en fonction de la navigation et un de sélection au milieu;
- grande facilité pour télécharger la musique directement sur le iPod à partir du magasin en ligne;
- écran haute résolution suffisamment grand pour afficher quelques lignes et mécanisme de navigation simple et relativement flexible (différents modes d'accès aux chansons);
- mécanisme de défilement particulièrement efficace basé sur le mouvement du pouce autour d'un cercle.

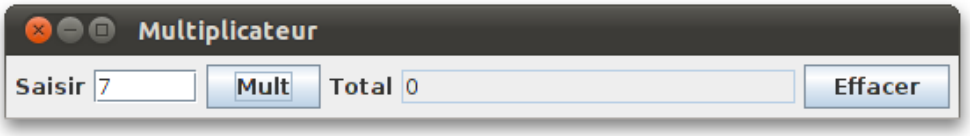
(b) [1 point] Décrivez les deux plus importantes erreurs de conception que l'on peut attribuer au baladeur Sanyo.

Solution:

- Fonctionnalité de recherche par mot clés : nécessite un grand nombre de boutons et deux modes. Le résultat est que ceci ajoute une énorme complexité qui dépasse largement la capacité d'utilisation sporadique de ces fonctions de manière efficace et qui alourdit l'utilisation pour des tâches courantes.
- Fonctionnalité album avant et album arrière non disponible (ou du moins non évidente)
- Écran de deux lignes ne permet pas de consulter adéquatement le contenu MP3 et de naviguer efficacement.
- Erreur de contraste de couleur (toutefois moins importante en terme d'impact que les autres erreurs).

Question 5 (7 points)

Le code de l'annexe B implémente une architecture MVC pour un calculateur dont la fenêtre est affichée dans la figure ci-dessous.



(a) [3 points] Analyser le code de l'annexe B pour décrire le comportement de l'application selon les instructions suivantes :

1. Après le démarrage, si l'utilisateur saisit le chiffre 4 dans le champ de gauche (`Saisir`) et appuie sur le bouton `Mult`, quel sera le chiffre affiché dans le champ de droite (`Total`)? (n.b. la figure ci-dessus affiche un état arbitraire)

   Solution: Le chiffre 4.

2. Si, par après 1. ci-dessus, l'utilisateur appuit à nouveau sur `Mult`, quel sera le chiffre affiché dans le champ `Total`?

   Solution: Le chiffre 16.

3. Énumérez, dans l'ordre temporel d'exécution, les méthodes du code de l'annexe qui sont appelées lorsque le bouton `Mult` est appuyé? (ne mentionnez que les méthodes définies dans le code de l'annexe, pas les méthodes de librairies externes)

   Solution:

   (a) `controller.actionPerformed(ActionEvent e)` (classe `CalcController.MultiplyListener`)

   (b) `m_view.getUserInput()` (classe `CalcView`)

   (c) `m_model.multiplyBy(userInput)` (classe `CalcModel`)

   (d) `m_model.getValue()` (classe `CalcModel`)

   (e) `m_view.setTotal()` (classe `CalcView`)

(b) [2 points] Pour cette version du code, aucun calcul n'est effectué si l'utilisateur saisit un nombre et appuit sur le retour de chariot (*return*). Modifiez le code pour que le calcul se fasse dès que le retour de chariot est saisi, comme si le bouton `Mult` avait été appuyé. N.B. Une seule ligne de code additionnelle est nécessaire. Précisez les hypothèses que vous faites si vous n'êtes pas certain de l'implémentation de la librairie Swing.

Solution: Le `return` (ou `enter`) correspond à l'action *sémantique* `ActionPerformed` du composant `JTextField`. Il suffit donc d'ajouter à la ligne 80 l'instruction :
`m_userInputTf.addActionListener(mal);`

(c) [2 points] En présumant que les calculs pourraient être très longs et qu'on décide d'ajouter un bouton `Annuler`, expliquez les changements que vous devez apporter au code pour que l'action d'annulation produise le résultat attendu.

Solution: La réponse doit expliquer les principes de la concurrence pour ce type de fonctionnalité. Il s'agit essentiellement d'effectuer le code de

`ActionPerformed` de la version actuelle à partir d'un fil `SwingWorker`. Le code de `ActionPerformed` doit donc créer à chaque appel un objet héritant de `SwingWorker`. Le code de *worker* doit implémenter `doInBackground` en répliquant essentiellement les instructions originales de `ActionPerformed`. Puis, nous devons appeler
`SwingWorker.execute()` afin de démarrer le fil *worker* à partir de `ActionPerformed`.

Annexe A — Heuristiques de Bastien et Scapin

1. **Guidage** : ensemble des moyens mis en oeuvre pour conseiller, orienter, informer et conduire l'utilisateur lors de ses interactions avec l'ordinateur.

2. **Charge de travail** : ensemble des éléments de l'interface qui a un rôle dans la réduction de la charge perceptive ou mnésique des utilisateurs (charge cognitive), de même que dans l'augmentation de l'efficacité du dialogue.

3. **Contrôle explicite** : prise en compte par le système des actions explicites des utilisateurs et le contrôle qu'ont les utilisateurs sur le traitement de leurs actions.

4. **Adaptibilité** : capacité à réagir selon le contexte et selon les besoins et les préférences des utilisateurs.

5. **Gestion des Erreurs** : moyens permettant d'une part d'éviter ou de réduire les erreurs, d'autre part de les corriger lorsqu'elles surviennent.

6. **Homogénéité/Cohérence** : les choix de conception d'interface doivent être conservés pour des contextes identiques, et doivent être différents pour des contextes différents.

7. **Signifiance des codes et dénominations** : il doit y avoir adéquation entre l'objet ou l'information affichée ou entrée, et son référent. Par exemple, rendre les règles d'abréviation explicites.

8. **Compatibilité** : il faut qu'il y ait accord entre, d'une part, les tâches et, d'autre part, la fonctionnalité, l'information affichée, l'organisation des sorties, des entrées et du dialogue d'une application donnée. Il faut aussi un accord entre les fonctionnalités et l'information affichée et les caractéristiques des utilisateurs.

Annexe B — Code Java Swing pour créer un calculateur

```
public class Main {

    public static void main(String[] args) {
        CalcModel      model      = new CalcModel();
        CalcView       view       = new CalcView(model);
        CalcController controller = new CalcController(model, view);

        view.setVisible(true);
    }
}
```

```
public class CalcModel {

    private static final String INITIAL_VALUE = "0";
    private BigInteger m_total;
```

```
16      CalcModel() {
            reset();
18      }

20      public void reset() {
            m_total = new BigInteger(INITIAL_VALUE);
22      }
        public void multiplyBy(String operand) {
24          m_total = m_total.multiply(new BigInteger(operand));
        }
26      public void setValue(String value) {
            m_total = new BigInteger(value);
28      }
        public String getValue() {
30          return m_total.toString();
        }
32  }
```

```
    class CalcView extends JFrame {
34      private static final String INITIAL_VALUE = "1";

36      private JTextField m_userInputTf = new JTextField(5);
        private JTextField m_totalTf     = new JTextField(20);
38      private JButton    m_multiplyBtn = new JButton("Mult");
        private JButton    m_clearBtn    = new JButton("Effacer");
40
        private CalcModel m_model;
42
        CalcView(CalcModel model) {
44
            m_model = model;
46          m_model.setValue(INITIAL_VALUE);

48          m_totalTf.setText(m_model.getValue());
            m_totalTf.setEditable(false);
50
            JPanel content = new JPanel();
52          content.setLayout(new FlowLayout());
            content.add(new JLabel("Saisir"));
54          content.add(m_userInputTf);
            content.add(m_multiplyBtn);
56          content.add(new JLabel("Total"));
            content.add(m_totalTf);
58          content.add(m_clearBtn);

60          this.setContentPane(content);
            this.pack();
62
            this.setTitle("Multiplicateur");
64          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
66
        void reset() {
```

```
68          m_totalTf.setText(INITIAL_VALUE);
        }
70      String getUserInput() {
            return m_userInputTf.getText();
72      }
        void setTotal(String newTotal) {
74          m_totalTf.setText(newTotal);
        }
76      void showError(String errMessage) {
            JOptionPane.showMessageDialog(this, errMessage);
78      }
        void addMultiplyListener(ActionListener mal) {
80          m_multiplyBtn.addActionListener(mal);
        }
82      void addClearListener(ActionListener cal) {
            m_clearBtn.addActionListener(cal);
84      }
    }
```

```
86  public class CalcController {

88      private CalcModel m_model;
        private CalcView  m_view;
90
        CalcController(CalcModel model, CalcView view) {
92          m_model = model;
            m_view  = view;
94
            view.addMultiplyListener(new MultiplyListener());
96          view.addClearListener(new ClearListener());
        }
98
        class MultiplyListener implements ActionListener {
100         public void actionPerformed(ActionEvent e) {
                String userInput = "";
102             try {
                    userInput = m_view.getUserInput();
104                 m_model.multiplyBy(userInput);
                    m_view.setTotal(m_model.getValue());
106
                } catch (NumberFormatException nfex) {
108                 m_view.showError("Bad input: '" + userInput + "'");
                }
110         }
        }
112     class ClearListener implements ActionListener {
            public void actionPerformed(ActionEvent e) {
114             m_model.reset();
                m_view.reset();
116         }
        }
118 }
```