IN101 – TP2 11/12 v1
Sujet

Durée : 2 h

1 OBJECTIFS

- **Savoir lire un énoncé et respecter les consignes (bien respecter toutes les étapes)**
- Comprendre le concept de référence (flèche) et le principe de l'interaction entre objets
- Savoir définir et utiliser des méthodes, ainsi que définir et accéder à des champs de type simple
- Avoir expérimenté la récursivité
- Savoir écrire des commentaires de documentation (commençant par `/**` et finissant par `*/`) et savoir générer une documentation automatiquement (javadoc).

2 LECTURE / COMPILATION (< 15mn)

Sauvegarder (dans `In101/Tp2`) puis Project/Open non BlueJ… le fichier `compilation.jar` .
Lire la classe, regarder comment sont programmées/présentées les différentes méthodes. Lire les commentaires javadoc et les retrouver dans la vision « Documentation » (choisir en haut à droite); *dans quel ordre les accesseurs et les modificateurs ont-ils été écrits ?*
Puis compiler, traduire-comprendre le message d'erreur, corriger, et recommencer jusqu'au succès. Si vous ne comprenez pas un mot en anglais, n'hésitez pas à en demander la traduction.

3 CRÉATION DE MÉTHODES

Nota : le travail demandé doit être terminé, en séance ou, à défaut, hors séance.

3.1 Projet "maison"

Cet exercice va consister à modifier des classes d'un exemple de projet BlueJ : le projet maison

3.1.1 Préparer l'environnement de travail

Sauvegarder (dans `In101/Tp2`) puis ouvrir le fichier `proj_maison.jar` .

3.1.2 Découvrir et essayer le projet *maison*

1. Dans la fenêtre principale de Bluej, les flèches entre classes définissent des liens de dépendance. Les flèches en tireté spécifient un lien "uses". Comme l'indiquent ces flèches, la classe Maison utilise les classes Cercle, Carre et Triangle, et ces dernières utilisent la classe Canvas.
   La classe Canvas définit des utilitaires de bas niveau pour l'affichage graphique d'objets : nous ne l'étudierons pas.
2. Après avoir compilé (si nécessaire), **créer** un objet Carre [clic droit sur cette classe puis choisir new Carre()], le rendre visible, et le déplacer de 200 pixels vers le bas ==> **invoquer** la méthode depVertical [clic droit sur l'objet puis void depVertical( )] ; **créer** de même un objet Maison et **invoquer** la méthode dessine( ) sur cette instance.
   L'image affichée illustre que l'objet de type Maison est une composition d'objets de type Cercle, Carre et Triangle. **Invoquer** maintenant la méthode place. *Que devrait-elle provoquer ?*
   Que se passe-t-il ? Les méthodes appelées dans place effectuent-elles les déplacements souhaités ?

3.1.3 Compléter les méthodes de déplacement lent (par récursivité)

1. **Lire** la classe Carre et chercher la méthode depLentVertical. Un commentaire indique où la compléter. **Écrire** les instructions permettant un appel récursif de cette méthode avec un appel de depVertical d'un seul pixel à chaque fois.
   a) Essayer d'abord avec pDistance toujours positive. Compiler, tester.
   b) Essayer ensuite de tenir compte de la bonne direction (vDelta). Compiler, tester.
   Si vous obtenez une StackOverflowError, c'est que la récursion ne s'arrête jamais.
   [clic droit sur la barre rouge et blanche pour arrêter l'exécution]

2. Reporter vos instructions dans depLentHorizontal en adaptant les noms. Compiler. Tester.
3. Reporter ces modifications dans les classes Cercle et Triangle. Compiler. Tester. *Maison ensoleillée?*

3.1.4 Modifier la classe Cercle

1. Prendre connaissance du code source de la classe Cercle [double-clic].
2. *Exercice 3.1.4a :* **Créer** dans la classe Cercle la **fonction** suivante :
   - nom : getPosition
   - paramètre : aucun
   - valeur de retour : la position de l'objet codée sous la forme de l'entier 1000*x + y, où x et y sont les coordonnées du cercle (on supposera avoir toujours : 0 <= coordonnées < 1000).
3. Compiler, créer un objet de type Cercle, puis **tester** le bon fonctionnement de la méthode getPosition (la méthode doit apparaître grâce à un clic-droit sur l'objet). *Comment savez-vous que c'est le bon résultat ?*
4. *Exercice 3.1.4b :* **Créer** dans la classe Cercle un deuxième **constructeur** :
   - signature : celle d'un constructeur (avec les paramètres ci-dessous) !
   - paramètres : le diamètre, les coordonnées x,y, et la couleur
   - valeur de retour : n'a pas de sens pour un constructeur
   - comportement : initialise les 4 attributs de l'objet avec les 4 paramètres correspondant, et `aEstVisible` systématiquement à `true`.
5. Compiler, puis créer un objet de type Cercle à l'aide de ce nouveau constructeur. *Pourquoi le cercle n'est-il pas visible ?* Voir l'exercice suivant pour résoudre ce problème.

   Rappel : On notera que la classe Cercle possède ainsi deux méthodes de même nom : Cercle. Ceci ne crée pas d'ambiguïté car ces deux méthodes ont des signatures différentes. Cette possibilité qu'offre Java est appelée surcharge.
6. *Exercice 3.1.4c :* **Modifier** ce nouveau constructeur pour que le cercle soit automatiquement visible dès sa création. Regarder la méthode rendVisible() pour comprendre pourquoi le cercle n'était pas visible, puis appeler cette méthode plutôt que positionner `aEstVisible` à `true`.
7. Compiler, créer un objet Cercle, puis **tester** le bon fonctionnement de cette nouvelle version.

3.1.5 Modifier la classe Maison

1. Prendre connaissance du code source de la classe Maison.
2. *Exercice 3.1.5a :* **Modifier** la classe Maison de façon à ajouter comme 5ème composant du dessin un deuxième soleil dont **seule** la couleur a été modifiée. Compiler et tester. *Pourquoi le premier soleil n'est-il plus visible ? `setNoirEtBlanc()` et `setCouleur()` fonctionnent-elles toujours ?* Sinon, corriger et tester.
3. *Exercice 3.1.5b :* **Modifier** la classe Maison de façon à ce que la méthode qui rend l'image visible soit appelée automatiquement à la création de l'objet. Compiler et tester.
4. *Exercice 3.1.5c :* **Créer** dans la classe Maison la méthode suivante

   sans modifier la classe `Cercle`, puis compiler et **tester**.
   - nom : getPosition**s**DeuxSoleils
   - valeur de retour : la String décrite ci-dessous
   - comportement : retourne la chaîne de caractères de la forme `"x=12, y=7"` à partir des coordonnées aXPosition et aYPosition de chacun des soleils de l'image ; *comment récupérer ces 2 valeurs ?*

     Rappels :

     - `a= b % c;` met dans `a` le reste de la division entière de `b` par `c`
     - `%` est l'opérateur souvent appelé « modulo »

…/…

- `a= b / c;` met dans `a` le résultat (ou quotient) de la division entière de `b` par `c`
- le `+` appliqué à une String permet de lui concaténer la représentation textuelle de n'importe quel objet ou nombre.
- pour séparer les positions des deux soleils, on peut utiliser `" | "` par exemple.

5. *Exercice 3.1.5d :* **Modifier** la classe Maison de façon à ne pas avoir à répéter 2 fois à peu près la même chose dans getPosition**s**DeuxSoleils(). Pour cela, créer une nouvelle fonction getPositionSoleil() (publique ou privée ?), retournant une String et acceptant un cercle en paramètre, et ne retournant la position **que** de ce cercle. Ensuite, getPosition**s**DeuxSoleils() n'a qu'à appeler getPositionSoleil() 2 fois. Compiler et tester.
6. *Exercice 3.1.5e :* **Modifier** la méthode getPositionSoleil() pour qu'elle accepte en plus un paramètre String contenant le nom (que vous avez choisi) du cercle dont elle retourne la position ; la String retournée commencera donc par `"leNom : "`. Modifier la méthode getPosition**s**DeuxSoleils() en conséquence. Compiler et tester.

3.1.6 Générer automatiquement la documentation

1. *Exercice 3.1.6a :* **Ajouter** des commentaires de documentation aux méthodes que vous avez développées pour qu'elles soient renseignées au moins au même niveau que les autres. Regarder la « Documentation », puis **prendre connaissance** de l'arborescence des fichiers générés et, sous navigateur, **consulter** le fichier `index.html`.
2. *Exercice 3.1.6b :* Mêmes étapes pour la classe Maison (une des méthodes n'est pas commentée).
3. Prendre connaissance de l'arborescence des fichiers générés et, sous navigateur, **consulter le fichier `index.html`**. *Quel est le problème ?*
4. Pour y remédier, [menu Outils, choix Générer la documentation du projet]. **Attendre suffisamment** longtemps, puis vérifier le résultat en navigant parmi toutes les classes du projet.

3.1.7 Améliorer la documentation

1. Se reporter au « styleguide » de javadoc : http://java.sun.com/j2se/javadoc/writingdoccomments/index.html#styleguide .
2. Lire les parties consacrées aux « tags » @author, @version, @param, @return, et @see . Lire le reste en travail personnel.
3. *Exercice 3.1.7a :* **Ajouter/compléter** (si nécessaire) les commentaires javadoc en incorporant des `@author` et un `@version` dans chaque classe, ainsi que le `@return` et les `@param` dans chaque méthode. **Regénérer** la documentation et vérifier les informations qui y figurent désormais.

3.1.8 Terminer l'exercice

Regénérer la javadoc puis sauvegarder le projet `Maison` [menu Project, choix Save] et envoyer les fichiers à votre voisin. *Un peu fastidieux, non ?* Alors essayer plutôt [menu Project, choix Create Jar File…] puis envoyer uniquement le fichier .jar ainsi créé. Le destinataire n'aura plus qu'à utiliser [menu Project, choix Open non BlueJ…] et à désigner le fichier .jar.

Fermer le projet maison [menu Project, choix Close] , voire BlueJ.

4 Fin du TP

- Avez-vous bien fait TOUS les exercices, dans TOUTES leurs étapes ?
- Avez-vous compris TOUT ce que vous avez vu ? Sinon, demandez à un intervenant.
- Avez-vous expérimenté le Code-Pad ? (Essayez : `int i=12;↵ i*2↵` etc…)
- N'avez-vous pas d'idées d'autres modifications de méthodes que celles proposées ? Expérimentez-les.
- Lisez les compléments de cours présents sur la page web de l'unité.
- Si vous avez fini avant les 2 heures prévues, demandez aux intervenants si vous pouvez partir avant la fin du TP ou bien s'ils vous donnent un autre exercice à faire.
- **Sinon, terminez tout en travail personnel avant le prochain tp.**

Ce sujet a été élaboré par Denis Bureau, d'après un sujet d'Albin Morelle.