UEO55 - Programmation objet

Contrôle terminal
Durée 2H *-Documents autorisés-*

Cours (4 points)

1. Expliquez brièvement le fonctionnement de l'interaction entre un utilisateur et une interface graphique en java. Illustrer ce principe avec par exemple un clic à la souris sur un bouton.
2. Expliquer brièvement le mécanisme de sérialisation.

Exercice 1 : Mise en défaut de la généricité (3 points)

Le code suivant produit l'erreur de compilation suivante : *can not cast from Vector<Integer> to Vector<String>*

```
Vector<Integer> vec = new Vector<Integer>();
vec.add(4);
Vector<String> vec2 = (Vector<String>) vec;
```

1. Expliquer ce message.

Le code suivant ne produit ni erreur à la compilation, ni erreur à l'exécution

```
Vector<Integer> vec = new Vector<Integer>();
vec.add(4);
Object aux = vec;
Vector<String> vec2 = (Vector<String>) aux;
vec2.add("un texte");
for (Object obj:vec) { System.out.println(obj); }
```

2. Quel résultat produit ce code ?
3. D'après vous pourquoi n'y a-t-il pas d'erreur à l'exécution ?

Exercice 2 : Interfaces Graphiques au comportement aléatoire (7 points)

On souhaite développer un jeu de hasard dont le principe est le suivant. Une interface graphique affiche un certain nombre de boutons (***nbBoutons***) dont le comportement lors d'un clic est choisi aléatoirement parmi plusieurs comportements possibles.
Nous définirons trois comportements : **actionNulle, actionGagné, actionPerdu**.
Pour gagner au jeu, il faut cliquer sur deux boutons actionGagné sans avoir cliqué sur une bouton actionPerdu. Un clic sur un bouton actionNulle n'est pas pris en compte.
On considère codée la classe **compteur** possédant trois méthodes : *incremente()* qui incrémente le compteur de 1, *reset()* qui remet le compteur à 0 et *get()* qui renvoie la valeur du compteur.

1. On souhaite utiliser une liste pour stocker les boutons. Comment est elle déclarée et instanciée ?
2. Les trois comportements sont des écouteurs des évènements de type **Action**. Coder la classe actionGagné qui possède un attribut de type **compteur** (passé en paramètre du constructeur) et qui incrémente le compteur à chaque clic puis vérifie si le joueur à gagné ou non. S'il a gagné un message est affiché dans la console et le compteur réinitialisé. Remarque : le compteur est partagé par les différentes instances des classes actionNulle, actionGagné, actionPerdu.
3. On considère les trois écouteurs codés sur le même principe que précédemment. On s'intéresse maintenant à l'instanciation des boutons et des écouteurs. Coder une méthode prenant en paramètre un entier ***nbBoutons*** et qui permet d'instancier nbBoutons boutons (le texte affiché sur le bouton peut être le numéro du bouton ou un texte vide) et de leur associer un écouteur sur les évènements de type Action. Le choix de l'écouteur est fait aléatoirement et on considère codée une méthode *getAleatoire()* qui renvoie un chiffre entre 1 et 3.

4. On veut maintenant ajouter les boutons au conteneur de la fenêtre. La fenêtre est un attribut de type *JFrame* appelé *fenetre*. Donner le code correspondant.

Exercice 3 : Les poupées russes 2 (6 points)

On souhaite modéliser à nouveau le fonctionnement des poupées russes mais cette fois ci à l'aide d'une structure de données.
Pour cela on définit la classe ListeDePoupees qui hérite de la version générique de la classe LinkedList. Le paramètre générique de la classe ListeDePoupees doit être défini de manière à n'autoriser que des poupées Russes à être ajoutées à la liste.
L'objectif est de modifier la méthode add() de la classe LinkedList de manière à réaliser les tests nécessaires avant l'ajout.

1. Donner l'entête de la classe ListeDePoupees
2. Donner le code et l'entête de la méthode add() de la classe ListeDePoupees qui permet d'ajouter en fin de liste la poupée passée en paramètre si les conditions suivantes sont respectées : la dernière poupée de la liste est fermée et a une taille inférieure à la poupée passée en paramètre, la poupée passée en paramètre est ouverte et vide. (on considère la classe PoupeeRusse déjà codée et possédant les méthodes nécessaires à son utilisation).
3. Si on souhaite garantir qu'une poupée n'est pas présente dans plusieurs listes de poupées (relation d'agrégation), comment peut-on procéder ?