TP1 Conception Objet

Notions mobilisées: Héritage, Singleton Pattern

Exercice 1 – Bibliothèque

Téléchargez le dossier BibliothequeErreur, lisez attentivement le code, déterminez pourquoi ca ne compile pas et corrigez-le.

Exercice 2 – Airport

1. Téléchargez le projet Airport (y compris le dossier img) et lisez attentivement le code sans l'exécuter (le fichier LoadAndShow et le dossier img ne vous intéressent pas) mais assurez-vous que dans le main les images rate.jpg et bravo.jpg (qui se trouvent dans le dossier img) soient referencées avec le bon chemin.

Explication du programme : des avions sont créés par le programme principal, un avion est un thread qui souhaite décoller et atterrir et demande donc l'accès à une piste. Une tour de contrôle doit gérer l'unique piste. Quand un avion souhaite décoller ou atterrir, il attend l'authorisation de la tour de contrôle. Deux avions ne doivent pas réserver la piste en même temps, sous peine d'un accident.

2. La classe Airport est partiellement définie avec des méthodes statiques et 'synchronized', ce qui limite l'exécution de ces méthodes à un seul thread à la fois. Complétez la définition de la classe Airport en déclarant les variables `planes_on_runway` et `accident` avec les bons modificateurs, en suite lancez le programme.
3. Au lancement du programme on a un crash, bien que les méthodes de Airport et de ControlTower étaient tous 'synchronized'. Modifiez la classe ControlTower afin de la rendre un singleton; vous allez devoir modifier le constructeur de la classe Plane conformément. Puis lancez le programme, est ce que vous avez réussi à éviter un crash ?

Exercice 3 – Decoratorion d'images

1. Téléchargez le projet Decorator. La classe MyImage permet d'afficher des images à partir d'un chemin d'accès au fichier (path) via la méthode display. La classe Dog construit le chemin d'accès pour la photo Dog.jpg qui se trouve dans le répertoire img, et comporte une méthode MyImage buildImage() permettant de construire l'image concernée. Dans le Main activez la partie en commentaire pour placer un chapeau sur le chien.
2. On veut ajouter des lunettes en utilisant l'image Sunglasses.png, écrivez le code permettant d'afficher les lunettes sur le chien sachant qu'elles doivent être positionnées en cordonnées (250, 46).
3. On veut ajouter plusieurs accessoires à l'image du chien (des lunettes, un chapeau... selon les images disponibles sur le dossier img). Tout décorateur doit avoir un champ pour le chemin d'accès et deux attributs numériques (int) pour définir la poition de l'accessoir sur la photo, un decorateur doit également comporter une méthode MyImage buildImage() permettant de superposer à l'image du chien la décoration concernée en appelant la méthode paintOver(path, x, y) de tout objet de type MyImage. Définissez donc des classes Sunglasses et Hat sous-classes de Accessory afin que le code dans le Main puisse être remplacé par les deux lignes suivantes :

```
MyImage picture = new Hat(new Sunglasses(new Dog().buildImage()).buildImage()).buildImage();
picture.display();
```

4. On veut ajouter les mêmes accessoires à la photo Penguin.jpg, il convient alors de créer une interface Picture avec une méthode abstraite buildImage, définissez donc une classe Penguin qui implément Picture. Modifiez donc le code afin que les classes Dog et Accessory

implémentent cette même interface ; dans les classes qui définissent les décorateurs, il conviendra alors de remplacer l'attribut MyImage par un attribut de type Picture. Si le code a été correctement modifié, vous pouvez remplacer le code du Main comme suit :

```
Picture picture = new Hat(new Sunglasses(new Dog()));
picture.buildImage().display();
```