Laboratoire informatique – SIMDisplayColor : La gestion des exceptions

Table des matières

Description du programme

Le laboratoire SIMDisplayColor est une application permettant d'effectuer des opérations mathématiques d'addition et de multiplication sur des couleurs de format rgb et d'afficher la couleur calculée dans un fichier de sortie de format png.

Objectif du laboratoire

Ce laboratoire consistera à se familiariser avec la classe de couleur SColor ainsi que de ses opérations mathématiques afin d'approfondir et d'appliquer la notion d'exception en java par l'action de lancer (**throw**) et d'attraper (**try – catch**) des exceptions.

Documentation

Une documentation sur les couleurs de format rgb est disponible au lien suivant :

http://profs.cmaisonneuve.qc.ca/svezina/nyc/note_nyc/NYC_XXI_Chap%206.3.pdf

Programme

Le programme SIMDisplayColor est disponible dans un format jar au lien suivant :

http://profs.cmaisonneuve.qc.ca/svezina/projet/ray_tracer/download/SIMDisplayColor.jar

Partie 1 : Les opérations mathématiques sur les couleurs

1- Vous allez commencer par ouvrir le fichier SIMDisplayColor.java dans le **package** sim/application et lire la méthode **main()** de cette classe. On y retrouve le calcul d'un couleur ainsi que de son affichage. Lancer l'application SIMDisplayColor et vérifiez le résultat de la couleur. Le fichier image qui a été généré est disponible dans le répertoire de votre projet.

- **Quel est le nom de la couleur c1 ?**
- **Quel est le nom de la couleur c2 ?**
- **Quel est le nom de la couleur c3 obtenue par l'addition de la couleur c1 et c2 ?**

2 - Ajoutez une section de code dans la méthode **main()** de la classe SIMDisplayColor.java afin de réaliser l'addition de la couleur (1.0, 1.0, 0.0) avec la couleur (1.0, 0.0, 1.0).

- **Quel est le nom de la couleur obtenue par cette opération ?**
- **Est-ce que cette couleur est le résultat mathématique de l'addition ? Si non, expliquez pourquoi il n'en n'est pas ainsi. Pour trouver une explication, vous devez consulter la méthode normalizeClampChannel() utilisée lors de l'appel de la méthode normalizeColor() de la classe SColor.**

3- Ajoutez une section de code dans la méthode **main()** de la classe SIMDisplayColor.java afin de réaliser la multiplication par un scalaire de la couleur (1.0, 1.0, 1.0) avec le scalaire 0.6.

- **Quel est le nom de la couleur obtenue par cette opération ?**
- **Décrivez une interprétation à la multiplication d'une couleur par un scalaire positif inférieur à 1 sur le résultat de la couleur calculée.**

4- Ajoutez une section de code dans la méthode **main()** de la classe SIMDisplayColor.java afin de réaliser la multiplication de la couleur (0.0, 1.0, 1.0) avec la couleur (1.0, 0.2, 0.5).

- **Quel est le nom de la couleur obtenue par cette opération ?**
- **Décrivez une interprétation à la multiplication d'une couleur par une autre couleur.**

Partie 2 : Gestion d'une couleur négative

1- Ajoutez une section de code dans la méthode **main()** de la classe SIMDisplayColor.java afin de réaliser l'addition de la couleur (0.8, 0.1, 1.0) avec la couleur (-0.7, 0.1, -0.5). Dans cette addition, on y retrouve des couleurs avec des champs « négatif ».

- **Quelle est la valeur numérique du résultat mathématique de cette opération ?**
- **Est-ce qu'une couleur a pu être affichée ? Si oui, préciser le nom de la couleur. Si non, pourquoi ?**

2- Ajoutez une section de code dans la méthode **main()** de la classe SIMDisplayColor.java afin de réaliser l'addition de la couleur (-0.2, 0.5, 0.7) avec la couleur (-0.4, 0.2, -0.5).

- **Quelle est la valeur numérique du résultat mathématique de cette opération ?**
- **Est-ce qu'une couleur a pu être affichée ? Si oui, préciser le nom de la couleur. Si non, pourquoi ?**

3- Les dernières instructions à la méthode **main()** engendre une exception lancée par l'objet **buffer** de la classe **BufferedImage** situé dans la méthode **display(SColor color)**. Conceptuellement, une couleur ne devrait jamais être négative, car elle n'a pas d'interprétation chromatique.

Pour régler cette situation, vous allez améliorer le constructeur de la classe SColor en introduisant une contrainte sur les valeurs numériques admissibles au champ r, g et b de la classe SColor.

Dans le constructeur

public SColor (double red, double green, double blue, double alpha) ,

vous allez vérifier que les champs soient positifs avant d'effectuer l'affectation de ces valeurs à votre objet SColor. Vous allez lancer une exception de type SConstructorException avec un message décrivant les raisons de l'exception si l'un des paramètres r, g, ou b est invalide. Exécutez votre application.

- **Est-ce qu'une exception de type SConstructorException est lancée par votre application ? Si oui, expliquez pourquoi.**

4- Afin de permettre l'exécution complète du programme, il faut préciser au programme ce qu'il doit faire lorsque survient une exception. Regroupez dans un bloc « **try – catch** » les instructions pouvant potentiellement générer des exceptions de type SConstructorException. Dans le bloc « **catch** », affichez un message console lorsqu'une exception survient. Un message adéquat doit décrire la nature de l'exception qui est survenue. Exécutez votre application et assurez-vous que toutes les exceptions de type SConstructorException ont été attrapées.

- **Présentement, combien d'exception(s) sont lancées par l'exécution de votre application ?**

5- Ajoutez une section de code dans la méthode **main()** de la classe SIMDisplayColor.java afin de réaliser la multiplication par un scalaire de la couleur (0.5, 0.5, 1.0) avec le scalaire -0.3.

- **Est-ce qu'une multiplication par un scalaire négatif est une opération mathématique adéquate ?**
- **Est-ce qu'il faudrait ajouter des instructions permettant de lancer des exceptions de type SConstructorException lors de l'appel de la méthode**

  public SColor multiply(double m)

  de la classe SColor lorsqu'il y a un scalaire négatif en multiplication avec une couleur ? Justifiez votre réponse.

Partie 3 : Plusieurs types d'exception pouvant être lancées

1- Pour construire une couleur de type SColor, on peut également utiliser le constructeur

public SColor(String string) throws SReadingException

utilisant comme paramètre une expression en mot représentant une couleur. Ajoutez une section de code dans la méthode **main()** de la classe SIMDisplayColor.java afin de réaliser l'addition de deux couleurs de votre choix en utilisant le « nom » des couleurs (ex : mauve, violet, rose).

- **Est-ce que vous pouvez strictement utiliser ce nouveau constructeur pour définir des nouvelles couleurs ? Si non, justifiez pourquoi.**

2- Après avoir réalisé cette tâche précédente, vous remarquerez que vous ne pouvez pas lancer l'exécution, car une erreur est toujours présente dans votre code. Le constructeur que vous utilisez peut lancer des exceptions de type SReadingException. Cette exception héritant de la classe *Exception* (comme **IOException**) doit obligatoirement être gérée dans un bloc « **try – catch** » contrairement à l'exception SConstructorException héritant de la classe *RuntimeException* (comme **NullPointerException**) où la gestion est facultative.

Regroupez votre bloc d'instruction dans un bloc « **try –catch** » en n'oubliant pas d'afficher un message console adéquat dans le bloc « **catch** ». Lancez l'exécution du programme.

- **Est-ce que votre exécution s'est réalisée sans lancer d'exception ? S'il y a eu une exception de lancée, décrivez la raison ?**

3- Ajoutez une section de code dans la méthode **main()** de la classe SIMDisplayColor.java afin de réaliser l'addition de la couleur (0.3, 0.2, 0.7) avec la couleur (0.7, -0.3, -0.5), mais en utilisant les constructeurs suivantes

```
c1 = new SColor(" [0.3 , 0.2 , 0.7 ] ");
```

```
c2 = new SColor(" [0.7 , -0.3 , -0.5 ] ");
```

et affichez la couleur résultante.

- **Quel type d'exception l'application a-t-elle lancée ?**
- **Pourquoi ce problème est toujours présent malgré le fait qu'une solution avait été préalablement trouvée au sujet des couleurs négatives ?**

4- Modifiez le constructeur

```
public SColor(String string) throws SReadingException
```

de la classe **SColor** afin que les exceptions que pourrait engendrer les instructions précédentes puissent être gérées adéquatement.