TP OOP Scala

Objectif

Familiarisation avec les aspects OOP du langage Scala.

Exercice 1

1.1. Préparation de l'environnement

Installation de Scala sur votre machine ou une machine de l'université.

Vous téléchargez une version **2.x (pas de version 3 ou supérieure)** de Scala. Vous pouvez par exemple télécharger la version **2.13.14** à l'url suivante:
https://www.scala-lang.org/download/2.13.14.html

Si vous souhaitez travailler avec IntelliJ (ce qui est conseillé dans le cadre de ce cours), vous pouvez suivre le lien suivant: "Download and use Scala!"

1.2. Création d'une classe

Une fois l'environnement fonctionnel, créez une classe `Bike` dans le package `fr.uge.td.scala`. Cette classe doit contenir les attributs suivants:
- `color` de type `String`
- `wheelSize` de type `Int`
- `speed` également de type `Int`, mais cette fois mutable avec comme valeur par défaut 0

1.3. Création de méthodes

Créez les deux méthodes suivantes:
- `speedUp` qui prend en paramètre un entier correspondant à l'augmentation de la vitesse. Par exemple `speedUp(10)` augmente la vitesse de 10km/h. Le type de retour doit être `Unit`.

- `brake` qui prend également un paramètre entier. Celui-ci correspond à la diminution de la vitesse. Le comportement est similaire à la méthode précédente, c'est-à-dire que `brake(5)` baisse la vitesse de 5km/h. Le type de retour est également `Unit`.

Pour ces deux méthodes, si la valeur passée en paramètre est négative, votre code devra lever une exception `IllegalArgumentException`.

1.4. Test du code

Vous allez maintenant tester que votre code adopte bien le comportement attendu. Créez un `object` Scala exécutable qui teste la création d'instances de `Bike` et teste les différents aspects suivants:
- fournir ou pas la vitesse à la construction
- tester si l'affichage et la modification de certains attributs est possible
- invoquer accélération/freinage avec des paramètres positifs et négatifs

1.5. Bilan de la première partie du TP

Si vous avez suivi les consignes du TP jusqu'ici, votre code devrait fonctionner. En revanche, il ne respecte pas les grandes lignes de la programmation fonctionnelle. Les principales raisons sont la présence d'attributs mutables et l'utilisation d'exceptions.

1.6. Pour un code plus fonctionnel

Remplacez les attributs mutables en attributs non mutables.
Modifiez les 2 méthodes d'accélération et de freinage, de manière à ne pas modifier l'état interne d'une instance.
Dans un premier temps, modifiez simplement la signature des méthodes de manière à ce qu'elles retournent un nouvel objet de type `Bike`.

1.7. Gestion des exceptions

La modification des signatures est un bon début, mais elle n'a pas résolu le problème des exceptions.
Dans un premier temps, modifiez le code des méthodes afin d'avoir un type de retour pertinent dans ce cas d'usage, c'est-à-dire basé sur `Option` ou `Either` de Scala.
Adaptez le code de l'`object` Scala responsable de l'exécution de votre programme.

Exercice 2

2.1 Analyse d'une structure du type liste chaînée

Le code suivant structure la création d'une liste chaînée.

```
sealed trait List[+A]
case object Nil extends List[Nothing]
case class Cons[+A](head: A, tail: List[A]) extends List[A]
```

En utilisant les éléments du cours et ceux que vous pouvez trouver dans le document du langage, essayez d'expliquer les choix d'implémentation de cette structure de données.Vous devez être capables d'expliquer les éléments suivants:

- L'utilisation du mot-clé `sealed`
- Les raisons de l'utilisation du mot clé `case`, dans les déclarations `case object` et `case class`.
- Les choix de variance des types paramétrés (pourquoi certaines fois `[+A]` et d'autres uniquement `[A]`)
- L'utilisation du type `Nothing`

2.2. Implémentation de la structure de liste chaînée

À partir de l'extrait de code que vous venez d'analyser, créez un `object` permettant d'instancier une liste comportant les valeurs 1, 2 et 3.

Rédigez une fonction qui retournera la queue de d'une liste. La signature doit être la suivante:

```
def tail[A](list: List[A]): List[A] = ???
```

Rédigez une fonction qui retournera une nouvelle liste avec un élément ajouté à sa tête. La signature doit être la suivante:

```
def prepend[A](list: List[A], a: A): List[A] = ???
```

Exercice 3

3.1. Création d'une structure du type arbre binaire

En vous inspirant de la structure de l'exercice précédent, créez une structure de données correspondant à un arbre binaire.

Fournir une implémentation d'une fonction qui calculera la taille d'un arbre. La signature est:

```
def size[A](t : Tree[A]): Int = ???
```