Programmation orientée objet - Java - Examen du 2 mai 2025.

Pour cet examen, votre fond d'écran doit être vert clair. Vous devez absolument mettre tous les fichiers que vous souhaitez rendre dans le répertoire **EXAM**, qui est déjà présent sur votre compte à l'ouverture de la session. Tout ce qui ne se trouve pas dans ce dossier sera perdu lorsque vous vous déconnecterez. Ne créez pas vous-même de répertoire **EXAM**. Mettez tous vos fichiers dans celui qui existe.

Sous Eclipse, votre workspace doit être dans **EXAM**. Sinon configurez le workspace d'Eclipse (File > Switch WorkSpace) pour qu'il corresponde à un répertoire dans **EXAM**). Attention, les noms des classes, des champs et des méthodes que l'on vous demande doivent impérativement être respectés. De plus, le code doit être indenté correctement.

Commencez par vérifier la configuration d'Eclipse.

- Vérifiez que le JRE est `/usr/local/apps/java23`.
- Vérifiez que le compilateur est java 23.
- Créez ensuite votre Java project qui porte votre nom : `NOM_PRENOM`.

La javadoc 23 et les slides du cours (seuls documents autorisés) sont accessibles ici :

<http://igm.univ-mlv.fr/~juge/javadoc-23/>

<https://igm.univ-mlv.fr/~beal/JavaBUT1.html>

Les tests des exemples donnés dans chaque question doivent être faits (les mettre en commentaire s'ils ne marchent pas). Ces tests sont pris en compte dans la notation.

Toutes les classes seront écrites dans un package `fr.uge.candy` au début et dans un package `fr.uge.candy2` à la fin. On écrira une méthode `main` dans une classe `Main` pour faire les tests dans chacun des packages.

On cherche à modéliser des sucreries récoltées par les enfants pour Halloween. Chaque enfant ramasse ses sucreries dans un panier, et il existe différents types de sucreries avec des caractéristiques différentes.

Exercice 1.

On souhaite définir un type `Candy` qui modélise un bonbon. Chaque bonbon a un goût (un champ `flavor` de type `String`) et peut-être allégé en sucre ou non (avec un champ `light` de type `boolean`). Écrire le type `Candy`. Faites en sorte que l'on puisse afficher les bonbons avec le code suivant :

```
public static void main(String[] args) {
    System.out.println("test 1");
    var candy1 = new Candy("apple", true);
    var candy2 = new Candy("strawberry", false);
    System.out.println(candy1);
    System.out.println(candy2);
}
```

qui donne

```
test 1
apple-flavored candy (sugar-free)
strawberry-flavored candy
```

L'affichage doit indiquer le parfum du bonbon et s'il est allégé ou non, en respectant le format de l'exemple ci-dessus.

Exercice 2.

Écrire une classe `CandyBasket` qui modélise un panier de bonbons (une liste `basket`) de bonbons récoltés par un enfant le soir d'Halloween. Un bonbon pourra figurer plusieurs fois dans la liste de bonbons. La classe doit posséder une méthode `add` qui permet d'ajouter des bonbons au panier et une méthode `averageSugarLevel` qui calcule le taux de sucre moyen du

panier sachant qu'un bonbon allégé a un taux de sucre de 0.1 alors qu'un bonbon "normal" a un taux de sucre de 0.6. La méthode `averageSugarLevel` lèvera une exception si le panier est vide.

Le code suivant devra donc fonctionner :

```
public static void main(String[] args) {
    ....
    System.out.println("test 2");
    var basket1 = new CandyBasket();
    basket1.add(candy1);
    basket1.add(candy1);
    basket1.add(candy2);
    basket1.add(candy1);
    System.out.println("Average sugar level: " + basket1.averageSugarLevel());
}
```

donne

```
test 2
Average sugar level: 0.22499999999999998
```

Exercice 3.

Modifier la classe `CandyBasket` de telle sorte que le code suivant affiche tous les bonbons d'un panier en respectant le formatage (un bonbon par ligne et il ne doit pas y avoir de passage à la ligne à la fin).

```
public static void main(String[] args) {
    ...
    System.out.println("test 3");
    System.out.println(basket1);
}
```

donne

```
test 3
apple-flavored candy (sugar-free)
apple-flavored candy (sugar-free)
strawberry-flavored candy
apple-flavored candy (sugar-free)
```

Exercice 4. On souhaite pouvoir ajouter dans le panier des sucres d'orge (`BarleySugar`). Chaque sucre d'orge a un goût (`flavor` de type `String`) et une couleur (`color` de type `String`). Le taux de sucre d'un sucre d'orge est de 0.8 sauf pour ceux qui sont "orange" ou "black" qui ont un taux de sucre de 0.4.

Écrire le type `BarleySugar` de telle sorte que le code suivant fonctionne. On remarquera qu'il faut ajouter une méthode `sugarLevel` dans `BarleySugar`.

```
public static void main(String[] args) {
    ...
    System.out.println("test 4");
    var barleysugar1 = new BarleySugar("cherry", "red");
    var barleysugar2 = new BarleySugar("licorice", "black");
    System.out.println(barleysugar1 + ", " + barleysugar1.sugarLevel());
    System.out.println(barleysugar2 + ", " + barleysugar2.sugarLevel());
}
```

donne

```
test 4
red barley sugar cherry-flavored, 0.8
black barley sugar licorice-flavored, 0.4
```

Exercice 5.

On souhaite pouvoir ajouter aussi bien des `BarleySugar` que des `Candy` à un panier de sucreries. De plus, l'association des parents a fourni une liste des sucreries interdites que l'on ne peut pas ajouter au panier :

1. les sucres d'orge de couleur "orange",
2. toutes les sucreries au goût "licorice".

On aura donc besoin d'une interface `Treat` pour toutes les sucreries que l'on peut ajouter dans un panier.

Dans ce qui suit, quand on parle de bonbon, cela désigne un objet `Candy`, quand on parle de sucre d'orge, cela désigne un objet `BarleySugar`, et quand on parle de sucrerie, c'est un objet de type `Treat` qui peut être bonbon ou sucre d'orge.

Écrire l'interface `Treat` (sucrerie) de telle sorte que le code suivant fonctionne et que l'on ne puisse pas ajouter les sucreries interdites au panier. Une tentative d'ajout d'une telle sucrerie provoquera la levée d'une exception. Attention on ne doit pas empêcher la création des sucreries interdites mais interdire seulement leur ajout dans un panier.

```
public static void main(String[] args) {
    ....
    System.out.println("test 5");
    var barleysugar3 = new BarleySugar("strawberry", "red");
    basket1.add(barleysugar1);
    basket1.add(barleysugar3);
    System.out.println("Average sugar level: " + basket1.averageSugarLevel());
    // basket1.add(barleysugar2); // provoque une exception
}
```

donne

```
test 5
Average sugar level: 0.4166666666666667
```

Exercice 6.

Écrire une méthode `Map<Treat, Long> countSameCandies` qui renvoie une map donnant pour chaque sucrerie le nombre d'occurrences (de type `Long`) de cette sucrerie présente dans le panier.

```
public static void main(String[] args) {
    ...
    System.out.println("test 6");
    System.out.println(basket1.countSameCandies());
}
```

donne

```
test 6
{red barley sugar cherry-flavored=1,
  strawberry-flavored candy=1,
  apple-flavored candy (sugar-free)=3,
  red barley sugar strawberry-flavored=1}
```

L'affichage de la map ci-dessus est en fait sur une ligne.

Exercice 7.

Écrire une méthode `List<Treat> barleySugars()` qui renvoie la liste non modifiable des sucres d'orge (uniquement les `BarleySugar` et pas les `Candy`) qui sont présents dans le panier.

```
public static void main(String[] args) {
    System.out.println("test 7");
    System.out.println(basket1.barleySugars());
}
```

donne

test 7

```
[red barley sugar cherry-flavored, red barley sugar strawberry-flavored]
```

Exercice 8.

Faites une copier-coller du package `fr.uge.candy` pour créer un nouveau package `fr.uge.candy2`. On conserve dans `fr.uge.candy2` les records `Candy` et `BarleySugar`, et l'interface `Treat`. On va changer les classes `Main` et `CandyBasket`.

Les sucreries sont maintenant stockées dans la classe `CandyBasket` sous forme d'une map dont les clés sont les sucreries et les valeurs sont leur nombre d'exemplaires dans le panier. La classe contiendra donc un seul champ `basket` de type `HashMap<Treat, Long>`.

Exercice 9.

Ecrire la classe `CandyBasket` avec une méthode `add` pour ajouter une sucrerie dans la map. Les sucreries interdites ne seront pas ajoutées comme dans le package précédent.

Exercice 10.

Ajouter une méthode `toString` pour afficher le panier comme ci-dessous.

```
public static void main(String[] args) {
    System.out.println("test 8");
    var candy1 = new Candy("apple", true);
    var candy2 = new Candy("strawberry", false);
    var basket1 = new CandyBasket();
    var barleysugar1 = new BarleySugar("cherry", "red");
    var barleysugar2 = new BarleySugar("licorice", "black");
    var barleysugar3 = new BarleySugar("strawberry", "red");
    basket1.add(candy1);
    basket1.add(candy1);
    basket1.add(candy2);
    basket1.add(candy1);
    basket1.add(barleysugar1);
    basket1.add(barleysugar3);
    System.out.println(basket1);
}
```

donne

test 8

```
apple-flavored candy (sugar-free): 3
strawberry-flavored candy: 1
red barley sugar strawberry-flavored: 1
red barley sugar cherry-flavored: 1
```

Exercice 11.

Ecrire la méthode `averageSugarLevel` pour cette nouvelle classe `CandyBasket`.

```
public static void main(String[] args) {
    ....
    System.out.println("test 9");
    System.out.println("Average sugar level: " + basket1.averageSugarLevel());
}
```

donne

test 9

```
Average sugar level: 0.4166666666666667
```

Exercice 12.

Ecrire une méthode `get(Treat treat)` qui renvoie le nombre de sucrerie `treat` dans le panier.

```
public static void main(String[] args) {
    ....
    System.out.println("test 10");
    System.out.println(basket1.get(candy1));
}
```

donne

```
test 10
3
```

Exercice 13.

Ecrire la méthode `countSameCandies()` qui renvoie une map non modifiable donnant pour chaque sucrerie son nombre d'exemplaires dans le panier.

```
public static void main(String[] args) {
    ....
    System.out.println("test 11");
    System.out.println(basket1.countSameCandies());
}
```

donne

```
test 11
{apple-flavored candy (sugar-free)=3,
  strawberry-flavored candy=1,
  red barley sugar cherry-flavored=1,
  red barley sugar strawberry-flavored=1}
```

L'affichage ci-dessus doit être sur une seule ligne.