IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Contrôle – écrit final, MAPD-A

INFO – MAPD-A

13 décembre 2022

Prénom:

Nom:

Consignes

Tout document autorisé. Dispositifs électroniques *non* autorisés 2h.

Vous êtes priés de répondre directement sur le sujet dans les cases prévues à cet effet. Si malgré votre souci de concision, vous aviez besoin d'une feuille supplémentaire, veillez à y faire figurer vos noms, prénoms et la référence du contrôle, à savoir : MAPD-A.

Les barèmes donnés sont indicatifs et pourront être modulés si cela s'avérait nécessaire. À cette fin et également à titre indicatif, essayez d'indiquer le temps passé pour chaque question. L'examen est prévu pour durer 2h (soit environ 5 minutes par point).

Les exercices sont indépendants.

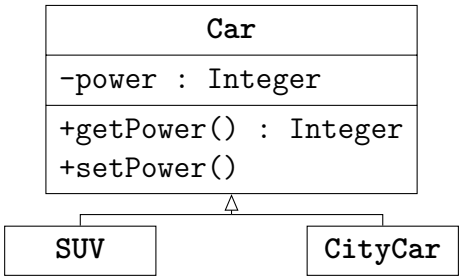
Dernier conseil : lisez bien les énoncés...

Merci d'avance et bon travail !

Exercice 1 (*Pour commencer*)	[2/20]	2
Exercice 2 (*Retour sur le réseau de Petri*)	[13/20]	2
Exercice 3 (*La classe Optional*)	[5/20]	6

Notez que les exercices de modélisation peuvent conduire à de nombreuses propositions différentes. Les exercices sont évalués sur le respect de la notation et les explications associées.

Exercice 1 (*Pour commencer*) [2/20]




```
public void setPower() {
   if (this intanceof SUV) {
      this.power = 25;
   } else if (this intanceof CityCar) {
      this.power = 7;
   } else {
      this.power = 13;
   }
}
```

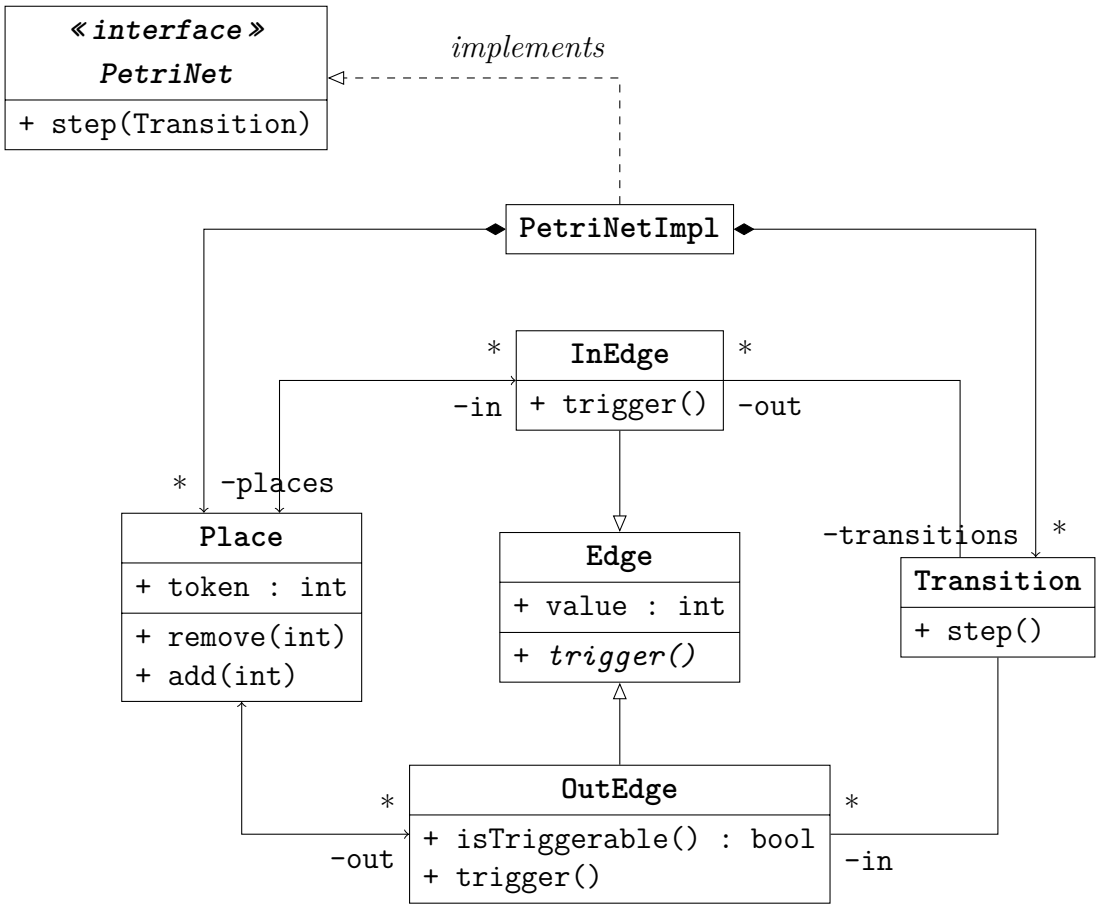
▷ **Question 1.1 :** [2/20]
Un de vos développeurs vous propose le bout de diagramme et l'implémentation de la méthode *setPower* ci-dessus.

- **Est-ce que ce code peut fonctionner correctement ?**
- **Est-il criticable et pourquoi ?**
- **Proposez une écriture alternative ?**

Exercice 2 (*Retour sur le réseau de Petri*) [13/20]

Depuis peu, vous travaillez dans une entreprise du numérique et une cliente souhaite vous confier le développement d'un simulateur basé sur les réseaux de Petri. Une chance, vous les connaissez. D'ailleurs, voici un modèle, page suivante, qui nous servira de support tout au long de l'exercice.

Ce modèle vous a été fourni par un collègue et a été utilisé dans un projet antérieur. Il manque volontairement de nombreuses méthodes (en particulier pour la construction/déconstruction du réseau) auxquelles nous ne nous intéresserons pas.



▷ **Question 2.1 :** **[1/20]**

Votre collègue vous demande de relire et de lui faire des retours pour améliorer son diagramme. Proposez 2 améliorations structurelles (relations, navigabilité, attributs, obligation de programmation) et justifiez-les.

Votre cliente, qui n'est pas habituée à la programmation objet, ne comprend pas bien comment ceci fonctionne.

▷ **Question 2.2 :** **[2/20]**

Pour lui montrer la dynamique ou comment s'exécute un pas (step), dessinez un diagramme (cohérent avec le diagramme de classe précédent) qui explique comment `step()` du *PetriNet* fonctionne.

Votre cliente vous fait remarquer que cette solution ne lui convient pas. Les tokens qu'elle manipule sont bien plus sophistiqués. Un jeton peut « porter » un état, une couleur, une forme selon le contexte.

▷ **Question 2.3 :** **[2/20]**
Trouvez deux arguments pour la rassurer justifiant l'utilisation d'une approche objet.

Dans la suite de l'exercice, on va donc généraliser la notion de token. Les tokens ne seront plus simplement représentés par une valeur dans une place (int), mais par un véritable `Objet`, capable d'avoir des propriétés.

Vous suggérez donc de *réifier* les Tokens.

▷ **Question 2.4 :** **[1/20]**
Faites évoluer le diagramme de classes pour résoudre le problème en supprimant l'attribut `token` de la classe `Place`. Les tokens seront des instances : 3 tokens seront donc représentés par 3 instances. De l'ancien diagramme vous ne montrerez que la classe `Place`.

▷ **Question 2.5 :** **[1/20]**
Pour assurer la compatibilité avec la version précédente, écrivez en Java, la méthode qui retourne le nombre de tokens contenus dans la place.

▷ **Question 2.6 :** **[1/20]**
Commentez l'accès au nombre de tokens. La méthode `isTriggerable()` en a besoin. Comparez la solution initiale de votre collègue (accès direct à l'attribut) et celle que vous proposez.

Votre cliente précise le type de réseau de Petri qu'elle souhaite pouvoir manipuler :

Informellement, un réseau de « haut niveau » inclut les propriétés suivantes :

1. Capacité pour un token de porter une valeur.
2. Capacité de traiter plusieurs tokens comme un token simple.
3. Capacité pour les transitions de produire des tokens différents de ceux consommés.
4. Capacité pour les arcs d'avoir un prédicat pour décider s'ils sont tirables.

Dans le cadre de cet exercice nous allons traiter les 3 premières spécifications, et suggérer une solution à la quatrième à la fin.

La propriété (2) vous rappelle une solution classique : le patron *composite* qui permet de manipuler un groupe/composite de la même façon que ses composantes.

▷ **Question 2.7 :** **[2/20]**
Reprenez la solution proposée à la question 2.4 pour ajouter (1) de quoi traiter une valeur (la plus abstraite possible) et assurer la propriété (2) à l'aide du patron *composite*.

▷ **Question 2.8 :** **[2/20]**
Pour la troisième propriété [Capacité pour les transitions de produire des tokens différents de ceux consommés], suggérez une solution avec un diagramme de classe et une implantation en Java de la méthode `trigger()`.

Justifier votre proposition.

▷ **Question 2.9 :** **[1/20]**
Suggérez par un commentaire comment vous pourriez traiter la propriété (4) [Capacité pour les arcs d'avoir un prédicat pour décider s'ils sont tirables.]

Exercice 3 (*La classe Optional*) [5/20]

Une méthode de signature `public Type f()` retourne une instance de `Type`. Toutefois, un usage particulier peut faire en sorte que la fonction ne retourne rien ; ou plutôt, elle retourne `null`.

Cette utilisation des méthodes (par exemple dans une recherche d'objet dans une liste qui retourne **null** si on ne trouve pas l'objet) est classique, mais est une mauvaise pratique.

En effet, elle oblige celui qui appelle à être très prudent dans l'usage du résultat. Ce qui conduit à du code avec des expressions du style :

```
Type result =  o.f();
if (result !=null)
  result.doSomething();
```

▷ **Question 3.1 :** **[1/20]**
Que risque-t-on si on appelle directement `result.doSomething();` ?

▷ **Question 3.2 :** **[1/20]**
Proposez une solution classique pour éviter de retourner `null`. Donner alors la signature de la méthode `f()`

Pourtant, il est parfois pratique de retourner un objet dans tous les cas (et donc jamais **null**).

Une solution proposée par Java depuis la version 8 est la notion de `Optional`. La documentation de cette classe précise ceci :

> A container object which may or may not contain a non-null value. If a value is present, `isPresent()` returns **true**. If no value is present, the object is considered empty and `isPresent()` returns **false**.

Un `Optional` offre un service pour accéder à la valeur (*get*), un pour tester s'il y a un objet (*isPresent*), un pour comparer (*equals*), une méthode *of(Type)* qui retourne un `Optional` non-vide d'une instance de `Type` et une méthode *empty* qui retourne un `Optional` vide.

▷ **Question 3.3 :** **[1/20]**
Proposez une classe *générique*, sous la forme d'un diagramme de classe.

▷ **Question 3.4 :** [1/20]
Écrivez le code de la méthode `get`. Vous préciserez une précondition et une postcondition (par une expression ou un commentaire).

▷ **Question 3.5 :** [1/20]
La classe `Optional` offre la méthode de signature suivante. Expliquez (1) l'intention de la fonction `map` et (2) le type de son argument.

```
<U> Optional<U> map(Function<? super T, ? extends U> mapper)
If a value is present, returns an Optional describing the result
of applying the given mapping function to the value, otherwise
returns an empty Optional.
```