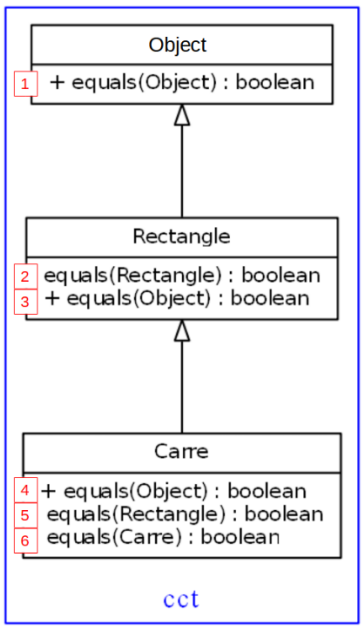
Conception Objet & Programmation

L3 MIAGE – 2022/2023

Durée : 2h00

Les deux questions sont indépendantes les unes des autres et peuvent être traitées dans n'importe quel ordre. Le barème est donné à titre indicatif. Notes et polycopiés de cours autorisés. Les téléphones doivent être éteints. **Les copies illisibles ne seront pas corrigées !**

Exercice A – Polymorphisme et Liaison dynamique [5 points]



Le diagramme ci-contre propose une conception objet avec 3 classes : **Object**, **Rectangle**, **Carre**.

On se donne deux objets et quatre références :

Rectangle rect = new Rectangle();

Carre carre = new Carre();

Object obj = carre ;

Rectangle rc = carre ;

Pour chacune des instructions suivantes :

- Identifier la méthode **equals** qui permet la compilation (choix entre 1 et 6) ou dire si l'instruction ne compile pas.
- Identifier la méthode **equals** qui sera effectivement exécutée (choix entre 1 et 6) ou dire l'instruction ne s'exécute pas.
    - rect.equals(obj)
    - rect.equals(rc)
    - obj.equals(rc)
    - rc.equals(carre)
    - carre.equals(carre)

Exercice B – Conception [15 points]

Nous avons vu en cours les collections et nous avons décrit le fonctionnement d'une liste chaînée.

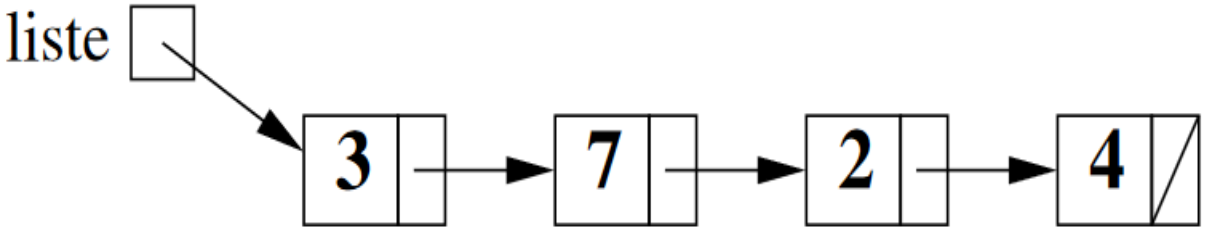


Figure 1 - exemple de liste chaînée avec 4 éléments

1. Proposer le code Java d'une classe *package* **Cellule** qui représente un seul élément de la liste chaînée. On pourra supposer qu'on veut y mettre des entiers. La flèche représente une référence sur la cellule suivante. Cette classe aura :
    - Un constructeur qui permet d'initialiser la valeur entière d'une **Cellule** **[0.5 point]**

- Un constructeur qui permet d'initialiser la valeur entière d'une **Cellule** et de le connecter à une cellule suivante **[0.5 point]**
- Un attribut package **element** pour connaître la valeur entière contenue dans la cellule **[1 point]**
- Un attribut package **next** pour accéder à la cellule suivante **[1 point]**

2. Proposer le code Java d'une classe **Liste** qui représente une liste chaînée (de **Cellule**s). La classe aura :
   - Un constructeur qui construit une liste en l'initialisant avec un nombre indéterminé d'entiers, mais au moins 1. **Ex : new Liste(3, 7, 2, 4)** pour construire la liste de la Figure 1. Attention : **new Liste()** sans paramètre doit être interdit ! **[1 point]**
   - Une méthode **int size()** pour connaître le nombre d'éléments de la liste **[1 point]**
   - Une méthode **void add(int)** pour ajouter un entier au début de la liste **[1 point]**
   - Une méthode **void removeFirst()** pour enlever le premier élément de la liste **[1 point]**
   - Une méthode **int contains(int)** pour savoir si une liste contient un élément entier, on renverra sa position dans la liste ou -1 si l'élément n'est pas contenu **[1 point]**
   - Une méthode **Liste reverse()** qui construit une nouvelle liste dont les éléments sont dans l'ordre inverse de la liste courante. Attention à ne pas modifier la liste initiale.
     Ex : 4 -> 2 -> 7 -> 3. **[1 point]**
   - Une méthode **void concat(Liste autre)** pour concaténer la liste courante à une **autre** sans modifier l'autre.

     [2 points]

3. On voudrait que notre liste soit de type **Iterable<Integer>**. On rappelle pour cela les interfaces **Iterable** et **Iterator** vues en cours (cf. Annexe).
   - Proposer le code Java d'une classe **ListeIterator** qui réalise (implements) l'interface **Iterator<Integer>**
     - Une méthode **boolean hasNext()** qui dit si il y a un élément suivant **[1 point]**
     - Une méthode **Integer next()** qui donne l'élément suivant en supposant qu'il existe ou envoie une exception si il n'existe pas **[1 point]**
   - Ajouter une méthode **Iterator<Integer> iterator()** à la classe **Liste** **[1 point]**
   - Ajouter une méthode **public String toString()** à la classe **Liste** **[1 point]**

Annexe – Iterable et Iterator

Interface Iterable

```
interface Iterable<T> {
  Iterator<T> iterator();
}
```

Interface Iterator

```
interface Iterator<T> {
  boolean hasNext();
  T next();
}
```