Programmation orientée objet

DOCUMENTS AUTORISES (le barème est donné à titre indicatif)

1. Exercice (6 pts)

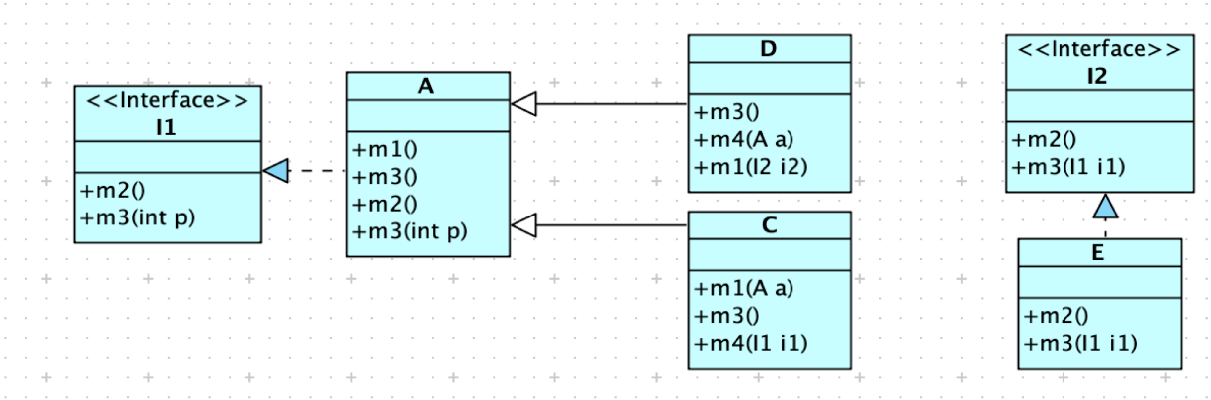

Figure 1.1: Diagramme de classes

Donner pour chacune des instructions du code ci-dessous :

- le nom de la classe dont le code est exécuté lors de l'appel,
- le typage statique et dynamique de la variable sur laquelle la méthode est invoquée,
- et s'il y a lieu, le typage statique et dynamique du paramètre.

Exemple : pour l'instruction *c.m3() ;* (1) : C, (2) : C, (3) : C, (4) et (5) : - [pas de paramètre]

`I1 i1 ;`
`A a = new A() ;`
`C c = new C() ;`
`D d = new D() ;`
`E e = new E() ;`
	`// Déclaration des variables utilisées ci-dessous`				

Instruction	**(1) Si OK, classe qui exécute le code**	**(2) Type statique variable**	**(3) Type dynamique variable**	**(4) Type statique paramètre**	**(5) Type dynamique paramètre**
`c.m2() ;`					
`c.m4(d) ;`					
`c.m1() ;`					
`d.m1(e) ;`					
`d.m1() ;`					
`d.m4(c) ;`					
`d.m1(c) ;`					
`d.m1(e) ;`					
`a = d ;`	Ok ?				
`a.m3() ;`					
`e.m2(a) ;`					
`d.m1(e) ;`					
`a = c;`	Ok ?				

`d.m4(a) ;`					
`a.m3() ;`					
`d = a ;`	Ok ?				
`d.m1() ;`					
`e.m1() ;`					
`e.m2(a) ;`					
`i1 = d ;`	Ok ?				
`c.m4(i1) ;`					
`e.m2(i1) ;`					
`i1.m2();`					

2. Problème (14 pts)

On souhaite pouvoir gérer une collection de données à la fois comme une file et comme une pile.
Pour cela, on veut utiliser un double chaînage, permettant de parcourir la collection de droite à gauche ou de gauche à droite.
L'interface de la collection est la suivante :

```
public interface ICollection<E> {
	/** ajoute un élément en queue de collection
	 * @param e référence de l'élément à ajouter	*/
	public void add(E e);
	/** enleve l'élément en tête de collection si la collection n'est pas vide
	 * @throws RuntimeException exception levée si la collection est vide	*/
	public void removeFirst();
	/** enleve l'élément en queue de collection si la collection n'est pas vide
	 * @throws RuntimeException exception levée si la collection est vide	*/
	public void removeLast();
	/** indique si la collection est vide
	 * @return true si vide, false sinon	*/
	public boolean isEmpty();
	/** retourne le nombre d'éléments stockés dans la collection
	 * @return taille effective de la collection	*/
	public int size();
	/** retourne la référence de l'élément en tête de collection
	 * si la collection n'est pas vide, sans l'enlever de la collection
	 * @return référence de l'élément en tête de collection
	 * @throws RuntimeException exception levée si la collection est vide	*/
	public E getFirst();
	/** retourne la référence de l'élément en queue de collection
	 * si la collection n'est pas vide, sans l'enlever de la collection
	 * @return référence de l'élément en tête de collection
	 * @throws RuntimeException exception levée si la collection est vide	*/
	public E getLast();
}
```

Code 2.1: Interface ICollection

1. Ecrire le code de la classe *MaCollection* qui met en œuvre l'interface *Icollection.* Cette classe utilise une *inner-class* qui implémente un double chaînage, permettant de parcourir la collection du premier élément au dernier et inversement du dernier au premier (et donc n'utilise pas de *Collection* type *ArrayList*).
2. Ajouter à la classe *MaCollection* les méthodes *public String toString()* et *public boolean equals(Object o)* issues de la classe *Object*.
3. Ecrire le code de la fonction de test *main*, qui déclare et teste les méthodes de la classe *MaCollection*. Séparer les tests qui utilisent *MaCollection* comme une *Pile* et comme une *File* en créant (au moins) deux instances différentes de *MaCollection*.