AP4B Final - durée 1h30

Documents non autorisés – dictionnaires papier autorisés

Chaque partie doit être rendue sur une copie séparée !!!!!!!

Partie 1 : Java

Méthodes Swap (2 pts)

On considère les 4 méthodes JAVA suivantes relatives à la possibilité d'échanger deux entiers. Pour chacune d'elle, donnez le résultat affiché par l'appel (à droite) et surtout, justifiez votre réponse.

`public void swap1(int i, int j) {`
`  int k = i;`
`  i = j;`
`  j = k;`
`}`
	`int i = 1; int j = 2;`
	`swap1(i, j);`
	`System.out.println("i= "+ i + " j= "+ j);`

`public void swap2(Integer i, Integer j)`
`{`
`  Integer k = i;`
`  i = j;`
`  j = k;`
`}`
	`i = 1; j = 2;`
	`Integer I = i; Integer J = j;`
	`swap2(I, J);`
	`i = I.intValue(); j = J.intValue();`
	`System.out.println("i= "+ i + " j= "+ j);`

`public void swap3(A x, A y) {`
`  A z = x;`
`  x.a = y.a;`
`  y.a = z.a;`
`}`
	`A x = new A(); x.a = 1;`
	`A y = new A(); y.a = 2;`
	`swap3(x, y);`
	`System.out.println("x.a="+x.a+"y.a="+y.a);`

`public void swap4(int[] i, int[] j) {`
`  int k;`
`  k = i[0];`
`  i[0] = j[0];`
`  j[0] = k;`
`}`
	`int [] t1 = new int[1]; t1[0] = 1;`
	`int [] t2 = new int[1]; t2[0] = 2;`
	`swap4(t1, t2);`
	`System.out.println("t1="+t1[0]+"t2="+t2[0]);`

Lambdas (3,5 pts)

1) Ecrivez la classe Operation (correctement) pour que ce programme fonctionne :

```
Operation addition = (a, b) -> a + b;
Operation multiplication = (a, b) -> a * b;

System.out.println(addition.calculer(3, 5));
System.out.println(multiplication.calculer(3, 5));
```

2) Ajouter un operateur pour la division en traitant comme il se doit la division par 0.

3) Que fait et affiche ce code :

```
List<Integer> list = Arrays.asList(1, 2, 3, 4);
list.stream().filter(n -> n % 2 == 0).forEach(System.out::println);
```

4) Et ce code (généré par IA):

```
List<Produit> produits = Arrays.asList(
```

```
new Produit("PC", 1200),
new Produit("Clavier", 50),
new Produit("Téléphone", 600));

produits.stream().filter(p -> p.getPrix() > 100).map(p -> new Produit(p.getNom(),
p.getPrix() * 0.9)).sorted(Comparator.comparing(Produit::getPrix)).forEach(p ->
System.out.println(p.getNom() + " : " + p.getPrix()));

class Produit {
    String nom;
    double prix;
    Produit(String nom, double prix) {
        this.nom = nom;
        this.prix = prix;
    }
    public double getPrix() { return prix; }
    public String getNom() { return nom; }
}
```

Problème. Problème des philosophes dînant (**4,5 pts**)

Dans le problème des "philosophes dînant", il y a cinq philosophes autour d'une table et 5 fourchettes posées sur la table à gauche et à droite de chaque philosophe. Il s'agit d'un problème typique d'allocation de ressource. Chaque philosophe a accès à deux fourchettes situées à sa gauche et sa droite. Une contrainte est qu'un philosophe ne peut manger que s'il a saisi ses deux fourchettes adjacentes, gauche et droite (une dans chaque main). Lorsqu'il a mangé, il repose les 2 fourchettes puis entame une discussion. Lorsqu'il a fini de parler, il décide de manger à nouveau. Et le cycle se répète. On veut réaliser un programme de simulation Java qui simule les philosophes dînant par des threads synchronisés. Chaque philosophe doit donc tenter de s'emparer des 2 fourchettes, s'il réussit, il mange, repose les deux fourchettes, puis discute. Et le cycle manger-discuter reprend, sans fin. Notez que tous les philosophes doivent manger équitablement au bout d'un temps fini.

Afin de garantir le bon fonctionnement, plusieurs solutions sont envisagées pour le choix des objets partagés sur lesquels s'effectue la synchronisation.

1) Une première solution proposée peut consister à se synchroniser sur les fourchettes, chacune d'elles étant partagée entre 2 philosophes. Par exemple, chaque philosophe regarde si sa fourchette gauche est posée et donc libre, il la saisit si c'est le cas, sinon il attend (sur le moniteur de la fourchette), une fois la fourchette gauche dans sa main, il procède de la même façon avec la fourchette droite. En quoi cette solution n'est-elle pas correcte. Que risque-t-il de se passer ? Donner un exemple précis de fonctionnement non désiré.

2) Une deuxième solution proposée consiste à se synchroniser sur un seul et unique objet partagé (prenons la Table) qui gère l'accès aux fourchettes et mémorise l'état de chaque fourchette (posée, prise). Les 2 fourchettes sont saisies en même temps si elles sont libres en même temps sinon il faut attendre. Le philosophe mange puis repose les deux fourchettes en même temps.

Compléter le programme Java de simulation ci-dessous (classe Table, classe Philosophe, fonction main()), là où vous trouvez les pointillés « **…** ».

```
class Philosophe implements ...{
	int id;
	private Table table;

	public Philosophe(int id, Table table) {
		this.id = id;
		this.table = table;
		Thread t = new ... (this);
		t. ... ();
	}

	public void run() {
		while (true) {

			// Phase discussion
			System.out.println("Je discute " + id);

			// simulation d'un temps de traitement
			try { Thread.sleep ((int)(Math.random() * 1000)); }
			catch (InterruptedException e) {}

			System.out.println("Je finis de discuter " + id);

			// Demander autorisation acces fourchettes G et D
			table.demanderFourchettesGD(...);

			// Phase manger
			System.out.println("Je mange " + id);

			// simulation d'un temps de traitement
			try { Thread.sleep ((int)(Math.random() * 1000)); }
			catch (InterruptedException e) {}

			System.out.println("Je finis de manger " + id);

			// Liberer les 2 fourchettes
			table.reposerFourchettesGD(...);
		}
	}
}

class Table {
	private boolean[] fourchettePrise;

	public Table(int nbPhilo) {
		fourchettePrise = new boolean[nbPhilo];
		for (int i = 0; i < nbPhilo; i++) {
			fourchettePrise[i] = false;
		}
	}

	// Demander ressource
	public ... ... demanderFourchettesGD(int id) {
		// Determiner id fourchettes G et D
		int idG = id;
		int idD = (id + 1) % fourchettePrise.length;// modulo le nb de fourchettes

		//System.out.println("Id fourchettes " + idG + " " + idD);
	  try {
		while (fourchettePrise[idG] || fourchettePrise[idD]) {
			... . ... ();
```

```
                }
            }
            catch(InterruptedException e) {}

            fourchettePrise[idG] = true;
            fourchettePrise[idD] = true;
        }

        // Liberer ressource
        public ... ... reposerFourchettesGD(int id) {
                // Determiner id fourchettes G et D
                int idG = id;
                int idD = (id + 1) % fourchettePrise.length;
            fourchettePrise[idG] = false;
            fourchettePrise[idD] = false;
                ... . ... ();
        }
}

class PhiloDinant {
        public static void main(String args[]) {
                System.out.println("Philosophes dinant !!!");

                Table table = new Table(5);

                Philosophe p[] = new Philosophe[5];
                for (int i = 0; i < 5; i++) {
                        p[i] = new ... (i, table);
                }
        }
}
```

Partie 2 : UML

Contexte

La série **Stranger Things** arrive à sa conclusion. Entre attentes des fans, contraintes de production et cohérence narrative, l'écriture du dernier scénario est un processus complexe qui nécessite l'intervention de plusieurs spécialistes.

Cet exercice propose de modéliser, à l'aide d'UML, un **générateur automatique de scénario** pour le dernier épisode de Stranger Things.

Le scénario est construit progressivement par plusieurs **scénaristes spécialisés**, chacun responsable d'un type précis d'information. Tous les scénaristes travaillent sur **un scénario partagé**.

Contraintes

- Un seul scénariste peut modifier le scénario à la fois.
- Un scénariste ne peut intervenir que :
    - lorsque le scénario est disponible,
    - et lorsque le scénario attend un élément correspondant à sa spécialité.
- Lorsqu'un scénariste ajoute un élément :
    - le scénario complète son texte,
    - il indique le **prochain type d'information attendu**,
    - puis il attend la prochaine modification.

Nous disposons de **5 scénaristes** :

1. Un(e) scénariste qui génère des **noms de personnages principaux**.
2. Un(e) scénariste qui génère des **noms de créatures ou antagonistes**.
3. Un(e) scénariste qui fournit des **lieux emblématiques**.
4. Un(e) scénariste qui fournit des **objets ou pouvoirs surnaturels**.
5. Un(e) scénariste qui décrit les **destructions ou conséquences** provoquées par ces pouvoirs.

Voici la trame générale d'un scénario. Les éléments entre crochets [] doivent être générés par les scénaristes correspondants.

*In the final season of Stranger Things, **[Hero Name]** and their friends return to face a new threat: **[Monster Name]**.The events take place in **[Location]**, where a mysterious **[Supernatural Object or Power]** begins to emerge. This power is capable of **[Destruction Description]**, putting the entire town at risk. With the help of **[Hero Name]**, the group fights back and manages to seal the rift. In the end, **[Hero Name]** realizes that the Upside Down is deeply connected to **[Location]**. At the end of the fights, **[Hero Name]** disappears. Is he/she dead?*

Question 1 – Problématique de concurrence (2 points)

Le scénario est une ressource partagée modifiée par plusieurs scénaristes.

- Identifiez les **problèmes de concurrence** possibles.
- Indiquez les **mécanismes généraux** permettant de garantir un accès correct au scénario.

Question 2 – Diagramme de classes UML (4 points)

Réalisez un **diagramme de classes UML** du générateur de scénario.

Le diagramme devra faire apparaître :

- la classe représentant le scénario,
- les scénaristes,
- les relations entre les classes,
- les classes abstraites, spécialisations et énumérations pertinentes.

Question 3 – Déroulement d'une génération de scénario (2 points)

À l'aide d'un **diagramme UML adapté**, décrivez le déroulement complet d'une génération de scénario, depuis le premier élément généré jusqu'au scénario final.

Vous préciserez :

- le type de diagramme choisi,
- le rôle des différents acteurs.

Question 4 – États du scénario (2 points)

À l'aide d'un **diagramme UML approprié**, représentez :

- les différents états possibles du scénario,
- les transitions entre ces états,
- les événements déclenchant ces transitions.

Question Bonus :

Proposez des modifications permettant de générer plusieurs scénarios en parallèle