Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Jijel
Faculté des Sciences Exactes et Informatique
Département d'informatique

Examen : Environnements et Programmation Dédiés

Nom : ……..... **Master 1 - Intelligence Artificielle**

Prénom : …… **Année universitaire : 2025 - 2026**

Note : ……/**20**. **Durée : 2 h 00**

Enoncé : Félicitations ! Vous venez d'intégrer une entreprise spécialisée dans le développement de solutions informatiques innovantes. Votre mission consiste à analyser des problématiques variées liées et à proposer des solutions adaptées, robustes et maintenables. Les situations suivantes illustrent des cas concrets que vous pourriez rencontrer dans votre activité professionnelle.

Situation 1 - QCM - Concepts généraux de programmation (6 pts)

Chaque question peut avoir **une ou plusieurs** bonnes réponses. Une réponse **incomplète ou partiellement fausse** vaut 0 point.

1. Soit le programme JavaScript suivant :

```
function f(list, operation) {
   return list
      .map(operation)
      .reduce((acc, valeur) => acc + valeur, 1);
}
const nombres = [1, 2, 3, 4];
const resultat = f(nombres, x => x * 2);
console.log(resultat);
```

☐ Ce programme adopte le **paradigme impératif**, car il définit une séquence d'opérations de filtrage et de transformation sur une collection de données.

☐ Les types des variables de ce programme sont vérifiés au moment de l'exécution, car il est écrit dans un langage **statiquement typé**.

☐ La fonction « **(acc, valeur) => acc + valeur** » passée en argument à **reduce**() est un exemple de fonction de première classe, car elle peut être créée dynamiquement et transmise comme valeur.

☐ f est une **fonction impure**, car elle n'engendre pas un effet de bord et elle respecte le principe de la transparence référentielle.

☐ L'exécution de ce programme affiche la valeur **20** dans la console.

☐ Aucune réponse n'est juste.

2. Parmi les affirmations suivantes concernant les grilles de calcul, cochez celles qui sont correctes :

☐ Les intergiciels (middleware) assurent la coordination et l'interopérabilité dans les grilles.

☐ Toutes les ressources d'une grille appartiennent à une seule entreprise.

☐ Les grilles de calcul utilisent des ressources disponibles à la demande.

☐ Les grilles de calcul permettent l'accès à des ressources distribuées géographiquement.

☐ Aucune réponse n'est juste.

3. Soit le programme java suivant :

```
class ShoppingCart {
   public void addItem(StringBuilder cart, String item) {
      cart.append(", ").append(item);
   }

   public void addItem(int itemId) {
      System.out.println("Item added with ID: " + itemId);
   }

   public static void main(String[] args) {
      ShoppingCart cartManager = new ShoppingCart();
      StringBuilder myCart = new StringBuilder("Apple");
      cartManager.addItem(myCart, "Banana");
      cartManager.addItem(101);
      System.out.println("Cart contents: " + myCart);
   }
}
```

☐ La méthode « **addItem** » est une méthode **surchargée**, car elle a le même nom mais des paramètres différents.

☐ La méthode « **addItem** » est une méthode **redéfinie**, car elle a le même nom mais des paramètres différents.

☐ Le type de polymorphisme utilisé dans ce code est le « **polymorphisme paramétrique** ».

☐ Le type de polymorphisme utilisé dans ce code est le « **polymorphisme Adhoc** ».

☐ La valeur de l'objet « **myCart** » après modification reste toujours « **Apple** », car le passage de paramètres par référence n'est pas supporté en java.

☐ La méthode **addItem**(StringBuilder **cart**, String item) montre qu'une modification des propriétés de l'objet « **cart** » affecte l'objet original « **myCart** ».

☐ Aucune réponse n'est juste.

4. Soit la requête HTTP suivante :

```
GET /api/v2/products/15/reviews?limit=10

Host: reviewService.com

Accept: application/json
```

☐ L'URI de cette requête identifie les avis associés au produit dont l'identifiant est 15.

☐ La méthode HTTP GET est utilisée ici pour modifier la liste des avis du produit.

☐ Le paramètre « **limit** » permet de contrôler le nombre de ressources retournées dans la réponse.

☐ L'utilisation de la version « **V2** » dans la requête illustre le principe « **sans état** » (**Stateless**) dans les services REST.

☐ La réponse de cette requête doit être au format **JSON**.

☐ Aucune réponse n'est juste.

5. Lesquels des critères suivants permettent de choisir un langage de programmation de manière **objective**.

☐ « **C** » est un langage compilé offrant un contrôle fin de la mémoire et de bonnes performances, ce qui le rend adapté au développement de systèmes bas niveau.

☐ « **Rust** » est apprécié pour sa syntaxe moderne et élégante, ce qui en fait un langage plus agréable à utiliser que d'autres pour les développeurs.

☐ « **Java** » propose une machine virtuelle garantissant la portabilité du code sur différentes plateformes disposant d'une JVM.

☐ **Python** est un langage interprété disposant d'un vaste écosystème de bibliothèques, facilitant le développement rapide d'applications dans des domaines variés.

☐ Aucune réponse n'est juste.

Situation 2 – Programmation fonctionnelle (8 pts)

Dans le cadre de vos fonctions au sein d'une équipe dédiée à l'analyse des alertes issues d'un système de détection d'intrusion, vous examinez des journaux retraçant des tentatives d'accès à des services du système. Le jeu de données ci-dessous est fourni pour cette analyse :

```
const securityLogs = [
  { id: "LOG-01", service: "SSH", severity: 7, origin: "192.168.1.50", attempts: 12, status: "blocked" },
  { id: "LOG-02", service: "HTTP", severity: 3, origin: "10.0.0.5", attempts: 1, status: "allowed" },
  { id: "LOG-03", service: "SSH", severity: 9, origin: "45.12.8.99", attempts: 450, status: "blocked" },
  { id: "LOG-04", service: "FTP", severity: 5, origin: "192.168.1.50", attempts: 5, status: "flagged" },
  { id: "LOG-05", service: "HTTP", severity: 8, origin: "88.200.15.4", attempts: 120, status: "blocked" },
  { id: "LOG-06", service: "SSH", severity: 2, origin: "10.0.0.5", attempts: 2, status: "allowed" }
];
```

1. Produire une nouvelle liste contenant uniquement les alertes dont **la sévérité est supérieure ou égale à 5** et dont **le statut n'est pas "allowed".**
2. Calculer **le nombre total de tentatives d'accès associées aux alertes bloquées** (status === "blocked") **« totalBlockedAttempts ».**
3. Produire la liste des adresses IP correspondant aux alertes dont le nombre de tentatives est strictement supérieur à 100 « **highRiskIPs** ». Exemple de forme attendue : **["45.12.8.99", "88.200.15.4"].**
4. Écrire une fonction **buildIncidentsByService (list)** qui permet de calculer pour chaque service (SSH, HTTP, FTP, etc.), le total des tentatives d'accès qui lui sont associées. Exemple de forme attendue : **{ SSH: 464, HTTP: 121, FTP: 5}.**
5. Écrire une fonction **generateThreatReport (**liste, riskFunction**)** qui :
    a. prend en paramètre une fonction « computeRiskScores », qui permet de calculer un **riskScore** selon la formule suivante : $\textbf{riskScore} = \boldsymbol{severity} \times \boldsymbol{attempts}$ **,**
    b. applique cette fonction à chaque alerte,
    c. et retourne une nouvelle liste d'objets contenant l'id de chaque alerte et son riskScore.

```
[ // Exemple de sortie attendue
  { id: "LOG-01", riskScore: 84 },
  { id: "LOG-02", riskScore: 3 },
  { id: "LOG-03", riskScore: 4050 }, ……
]
```

Remarque : Toute approche impérative (boucles, …) est interdite. Vous devez utiliser exclusivement la programmation fonctionnelle.

Astuces :

- L'instruction **«** jsonObj.**hasOwnProperty**("key") **»** vérifie si l'objet contient la clé "key".
- « **jsonObj**["**key**"] = 2 » crée la clé "key" si elle n'existe pas, sinon met à jour sa valeur.
- L'instruction « **return { [k1]: v1, [k2]: v2 }; »** permet de retourner directement un objet contenant plusieurs paires clé–valeur.

Situation 3 - Programmation Orientée Objet (partie 1) (3 pts)

Votre client est une entreprise spécialisée dans la gestion d'espaces de travail « coworking ». Elle souhaite mettre en place un système de réservation et de gestion de ses différents espaces. Le système doit modéliser deux types d'espaces : **les bureaux individuels** et **les salles de réunion**. Chaque espace est caractérisé par un identifiant unique (**spaceId**) et une superficie (**area**) en m^2.

Les bureaux individuels disposent d'un équipement informatique fixe défini par le nombre d'écrans (**nbScreens**) et une consommation électrique horaire (**powerUsage**) exprimée en watts. **Les salles de réunion** sont définies par une capacité maximale de participants (**maxCap**) et un équipement de visioconférence décrit par une qualité vidéo (**videoRes**) exprimée en résolution. Elles peuvent également être équipées d'un tableau interactif en option.

Question : Concevez une hiérarchie de classes en Java visant à représenter les différents types d'espaces et leurs attributs, en respectant les concepts de la programmation orientée objet. **Les constructeurs et les getter/setter ne devront pas être déclarés afin d'alléger la réponse.**

Situation 4 - Programmation Orientée Objet (partie 2) (3pts)

Une application calcule des frais de livraison selon le type de client et le mode de livraison. Le code ci-dessous est fonctionnel et produit des résultats corrects.

```
class LivraisonService {
   public double calculerFraisStandard(double poids) {
      return poids * 2.0;
   }
   public double calculerFraisExpress(double poids) {
      return poids * 4.0;
   }
   public double calculerFraisClientPremium(double poids) {
      return poids * 1.5;
   }
}

public class TestLivraison {
   public static void main(String[] args) {
      LivraisonService service = new LivraisonService();
      System.out.println(service.calculerFraisStandard(10));
      System.out.println(service.calculerFraisExpress(10));
      System.out.println(service.calculerFraisClientPremium(10));
   }
}
```

1. Refactorisez la classe **LivraisonService** en utilisant uniquement des méthodes surchargées afin de limiter la duplication de code.
2. Étendre le service pour permettre un calcul basé sur **le poids du colis** et **la distance de livraison** (prix = poids × 2.0 + distance × 0.5).
3. Adapter la méthode main pour appeler chacune des 4 variantes de calcul.