TD – DÉVELOPPEMENT JAVA 2

PARTIE 1 – STRUCTURES CONDITIONNELLES

Exercice 1 – Validation d'inscription (if / else)

Contexte : Une plateforme de formation en ligne vérifie l'éligibilité d'un candidat selon son niveau d'études.

```
int niveauEtudes = 3; // 1=Bac, 2=Bac+1, 3=Bac+2, 4=Bac+3+
```

Travail demandé :

1. Tester si le candidat a un niveau ≥ 2 (Bac+1 minimum requis)
2. Afficher :
   - "Inscription validée – Niveau suffisant" si niveau ≥ 2
   - "Inscription refusée – Niveau insuffisant" sinon
3. Tester avec niveauEtudes = 1, puis niveauEtudes = 4

Attendu : Structure if-else simple avec deux branches

Exercice 2 – Calcul de remise commerciale (if / else if / else)

Contexte : Un site e-commerce applique des réductions selon le montant du panier.

```
double montantPanier = 450.0; // en euros
```

Règles de remise :

- Montant ≥ 500€ → 20% de réduction
- Montant ≥ 300€ → 15% de réduction
- Montant ≥ 100€ → 10% de réduction
- Sinon → Aucune réduction

Travail demandé :

1. Déterminer le taux de remise applicable
2. Calculer le montant après réduction
3. Afficher :
   - "Remise appliquée : XX%"

    - "Montant final : XXX.XX €"
4. Tester avec 50€, 150€, 350€, 600€

Attendu : Structure if-else if-else avec calculs arithmétiques

Exercice 3 – Système de navigation (switch)

Contexte : Un logiciel de gestion dispose d'un menu principal codé par des chiffres.

```
int choixMenu = 2;
```

Options du menu :

- 1 → "Consulter les statistiques"
- 2 → "Exporter les données"
- 3 → "Importer un fichier"
- 4 → "Paramètres"
- Autre → "Erreur : option invalide"

Travail demandé :

1. Implémenter avec switch
2. Gérer le cas par défaut
3. Tester avec les valeurs : 1, 3, 5, 10

Attendu : Structure switch-case avec default

EXERCICE 4 – Éligibilité à un prêt bancaire (conditions composées)

Contexte : Une banque évalue l'éligibilité d'un client à un prêt immobilier selon plusieurs critères.

```
int ancienneteEmploi = 4; // en années
double revenuMensuel = 2800.0; // en euros
boolean incidentsPayement = false; // true = incidents dans les 12 derniers mois
```

Règles d'éligibilité :

- Ancienneté ≥ 2 ans ET revenu ≥ 2500€ ET aucun incident de paiement → Éligible
- Sinon → Non éligible

Travail demandé :

1. Tester les 3 conditions avec des opérateurs logiques (&&)
2. Afficher :
    - "Prêt accordé – Profil validé" si toutes les conditions sont remplies
    - "Prêt refusé – Critères non satisfaits" sinon
3. Afficher également les critères non satisfaits (ex: "Revenu insuffisant")
4. Tester avec 3 profils différents :
    - Profil 1 : anciennete=1, revenu=3000, incidents=false
    - Profil 2 : anciennete=5, revenu=2200, incidents=false
    - Profil 3 : anciennete=3, revenu=3500, incidents=true

Attendu : Conditions composées avec opérateurs logiques &&, ||, !

EXERCICE 5 – Catégorisation de produits (switch avec expressions)

Contexte : Un système de gestion de stock catégorise les produits selon un code à 2 lettres.

```
String codeProduit = "EL";
```

Codes et catégories :

- "AL" → Alimentation
- "EL" → Électronique
- "VT" → Vêtements
- "MB" → Meubles
- "LV" → Livres
- Autre → Catégorie inconnue

Travail demandé :

1. Utiliser un switch sur le String codeProduit
2. Afficher : "Catégorie : [Nom de la catégorie]"
3. Bonus : Associer à chaque catégorie un taux de TVA :
    - Alimentation : 5.5%
    - Électronique : 20%
    - Vêtements : 20%
    - Meubles : 20%
    - Livres : 5.5%
4. Afficher le taux de TVA correspondant
5. Tester avec : "AL", "EL", "XX"

Attendu : switch sur String avec gestion de valeurs par défaut

PARTIE 2 – TABLEAUX

Exercice 6 – Gestion des stocks d'un entrepôt

Contexte : Un entrepôt gère les quantités en stock de 6 produits.

```
int[] quantites = {45, 12, 78, 5, 60, 23};
```

Travail demandé :

1. Afficher la quantité du premier et du dernier produit
2. Modifier la quantité du 4ème produit (index 3) : passer de 5 à 50
3. Afficher tout le tableau après modification (avec une boucle)

Attendu : Accès par index, modification, parcours avec for

Exercice 7 – Analyse des ventes mensuelles

Contexte : Une entreprise analyse ses chiffres d'affaires mensuels sur 6 mois.

```
double[] chiffreAffaires = {12500.0, 9800.0, 15200.0, 11000.0, 18500.0, 13700.0};
```

Travail demandé :

1. Afficher tous les montants avec une boucle for
2. Calculer le chiffre d'affaires total sur les 6 mois
3. Calculer le chiffre d'affaires moyen (total / 6)
4. Compter combien de mois ont dépassé la moyenne
5. Afficher tous les résultats de manière claire

Exemple de sortie :

```
Chiffre d'affaires total : 80700.0 €
Chiffre d'affaires moyen : 13450.0 €
Nombre de mois au-dessus de la moyenne : 3
```

Attendu : Parcours, calculs cumulatifs, compteur conditionnel

Exercice 8 – Relevé de températures hebdomadaires

Contexte : Une station météo enregistre les températures maximales de la semaine.

```
int[] temperaturesMax = {15, 18, 12, 22, 19, 24, 16};
```

Travail demandé :

1. Trouver la température minimale de la semaine
2. Trouver la température maximale de la semaine
3. Afficher uniquement les jours où la température a dépassé 20°C
4. Calculer l'écart entre la température max et min

Exemple de sortie :

```
Température minimale : 12°C
Température maximale : 24°C
Jours > 20°C : 22°C, 24°C
Écart de température : 12°C
```

Attendu : Algorithmes de recherche min/max, filtrage conditionnel

EXERCICE 9 – Gestion de notes d'un groupe d'étudiants

Contexte : Un enseignant dispose des notes de 8 étudiants à un examen.

```
double[] notes = {12.5, 8.0, 15.5, 10.0, 14.0, 6.5, 18.0, 11.5};
```

Travail demandé :

1. Afficher toutes les notes
2. Calculer la moyenne de la classe
3. Compter le nombre d'étudiants ayant obtenu :
   - Une note ≥ 10 (admis)
   - Une note < 10 (non admis)
4. Trouver la meilleure note et la plus mauvaise note
5. Calculer le taux de réussite (% d'étudiants ≥ 10)

Exemple de sortie :

```
Moyenne de la classe : 12.0
Nombre d'admis : 6
Nombre de non-admis : 2
Meilleure note : 18.0
Plus mauvaise note : 6.5
Taux de réussite : 75.0%
```

Attendu : Parcours, compteurs, calculs statistiques

EXERCICE 10 – Filtrage de données (tableau vers tableau)

Contexte : Un système de surveillance surveille des capteurs de qualité de l'air (indice de pollution).

```
int[] indicesPollution = {45, 120, 78, 95, 150, 60, 200, 88, 110};
```

Règles :

- Indice < 50 : Bon
- Indice 50-100 : Moyen
- Indice > 100 : Mauvais

Travail demandé :

1. Compter le nombre d'indices dans chaque catégorie (Bon, Moyen, Mauvais)
2. Créer un nouveau tableau contenant uniquement les indices > 100
3. Afficher ce nouveau tableau
4. Calculer l'indice moyen de pollution

Attendu : Compteurs, création dynamique de tableau, filtrage

PARTIE 3 – LISTES (ArrayList)

Exercice 11 – Gestion d'une bibliothèque de cours

Contexte : Un étudiant gère sa liste de cours du semestre.

Travail demandé :

1. Créer une ArrayList<String> vide nommée cours
2. Ajouter au moins 6 cours (ex: "Mathématiques", "Programmation Java", "Base de données", etc.)
3. Afficher la liste complète
4. Supprimer un cours spécifique (ex: "Base de données")
5. Afficher la taille de la liste après suppression
6. Vérifier si un cours existe dans la liste (ex: "Algorithmique")

Attendu : Création ArrayList, méthodes add(), remove(), size(), contains()

Exercice 12 – Manipulation de scores de jeu

Contexte : Un jeu vidéo enregistre les scores des 5 dernières parties.

```
ArrayList<Integer> scores = new ArrayList<>();
```

Travail demandé :

1. Ajouter les scores suivants : 1200, 950, 1450, 800, 1600
2. Vérifier si le score 950 existe dans la liste
3. Afficher l'index du score 1450
4. Remplacer le score 800 par 1000 (utiliser set())
5. Afficher la liste mise à jour
6. Afficher le premier et le dernier score

Attendu : Méthodes contains(), indexOf(), set(), get()

EXERCICE 13 – Gestion dynamique d'une file d'attente

Contexte : Un centre d'appels gère une file d'attente de clients.

```
ArrayList<String> fileAttente = new ArrayList<>();
```

Travail demandé :

1. Ajouter 5 clients : "Client_A", "Client_B", "Client_C", "Client_D", "Client_E"
2. Afficher la file d'attente
3. Retirer le premier client de la file (index 0) → simuler un traitement
4. Afficher la file après traitement
5. Ajouter un nouveau client "Client_F" à la fin

6. Afficher la position actuelle de "Client_D" dans la file
7. Afficher le nombre total de clients en attente

Attendu : Gestion dynamique avec remove(index), add(), indexOf(), size()

EXERCICE 14 – Fusion de deux listes

Contexte : Deux équipes de développeurs ont chacune une liste de tâches. Il faut fusionner les listes.

```
ArrayList<String> tachesEquipe1 = new ArrayList<>(Arrays.asList("Tâche1", "Tâche2",
"Tâche3"));
ArrayList<String> tachesEquipe2 = new ArrayList<>(Arrays.asList("Tâche4", "Tâche5"));
```

Travail demandé :

1. Créer une nouvelle liste tachesGlobales vide
2. Ajouter toutes les tâches de tachesEquipe1 dans tachesGlobales (utiliser addAll())
3. Ajouter toutes les tâches de tachesEquipe2
4. Afficher tachesGlobales (doit contenir 5 tâches)
5. Vérifier si "Tâche3" est dans la liste globale

Attendu : Méthode addAll(), fusion de collections

PARTIE 4 – TRIS ET CONVERSIONS

Exercice 15 – Tri de noms de clients

Contexte : Un système CRM doit afficher une liste de clients triée.

```
List<String> clients = new ArrayList<>(Arrays.asList("Dupont", "Martin", "Bernard",
"Dubois", "Lefebvre"));
```

Travail demandé :

1. Trier la liste par ordre alphabétique croissant (A→Z)
2. Afficher la liste triée
3. Trier la liste par ordre alphabétique décroissant (Z→A)
4. Afficher la liste triée

Attendu : Méthode Collections.sort() et Collections.reverse()

Exercice 16 – Conversion Liste ↔ Tableau

Contexte : Conversion bidirectionnelle entre ArrayList et tableau natif.

```
List<String> pays = new ArrayList<>(Arrays.asList("France", "Allemagne", "Italie", "Espagne"));
```

Travail demandé :

1. Convertir la liste pays en tableau String[]
2. Afficher le tableau (avec boucle for)
3. Modifier un élément du tableau : remplacer "Italie" par "Belgique"
4. Reconvertir le tableau en liste
5. Afficher la nouvelle liste

Attendu : Méthode toArray() et Arrays.asList()

EXERCICE 17 – Tri de notes avec recherche

Contexte : Un professeur souhaite trier les notes d'un examen et identifier la médiane.

```
ArrayList<Double> notes = new ArrayList<>(Arrays.asList(14.5, 8.0, 16.0, 10.5, 12.0, 9.5, 18.0, 11.5));
```

Travail demandé :

1. Afficher la liste avant tri
2. Trier la liste par ordre croissant
3. Afficher la liste après tri
4. Calculer la médiane :
    - Si nombre pair d'éléments : médiane = (valeur_milieu1 + valeur_milieu2) / 2
    - Si nombre impair : médiane = valeur_centrale
5. Afficher la médiane

Indice : Pour une liste de 8 éléments (indices 0-7), la médiane = (note + note) / 2

Attendu : Tri + calcul statistique

PARTIE 5 – EXERCICES DE SYNTHÈSE

Exercice 18 – Système de gestion de notes (Synthèse complète)

Contexte : Créer un mini-système de gestion de notes pour un groupe d'étudiants.

Fonctionnalités à implémenter :

1. Ajouter une note (méthode `ajouterNote`)
    - Lire une note au clavier (Scanner)
    - Ajouter à la liste
2. Supprimer une note (méthode `supprimerNote`)
    - Lire l'index de la note à supprimer
    - Vérifier que l'index est valide
    - Supprimer la note
3. Afficher toutes les notes (méthode `afficherNotes`)
    - Parcourir et afficher toutes les notes
4. Calculer la moyenne (méthode `calculerMoyenne`)
    - Calculer la moyenne des notes
    - Afficher le résultat
5. Afficher les notes supérieures à la moyenne (méthode `notesSuperieures`)
    - Filtrer et afficher uniquement les notes > moyenne
6. Menu principal interactif :

```
=== GESTION DES NOTES ===
1. Ajouter une note
2. Supprimer une note
3. Afficher toutes les notes
4. Calculer la moyenne
5. Afficher notes > moyenne
6. Quitter
Votre choix :
```

Contraintes :

- Utiliser `ArrayList<Double>`
- Créer des méthodes séparées pour chaque fonctionnalité
- Utiliser `Scanner` pour les entrées utilisateur
- Boucle principale avec `switch` pour le menu
- Gestion des erreurs (ex: suppression d'une note inexistante)

Attendu : Programme complet avec menu interactif, gestion d'erreurs, modularité

EXERCICE 19 – Système de gestion de stock avancé

Contexte : Créer un système de gestion de stock pour un petit magasin.

Données :

```
ArrayList<String> nomsProduits = new ArrayList<>();
ArrayList<Integer> quantites = new ArrayList<>();
ArrayList<Double> prix = new ArrayList<>();
```

Fonctionnalités à implémenter :

1. Ajouter un produit
    - Lire nom, quantité, prix
    - Ajouter dans les 3 listes (indices synchronisés)
2. Afficher le stock complet
    - Afficher sous forme de tableau :

```
ID   Nom          Quantité   Prix unitaire   Valeur stock
-----------------------------------------------------------
0    Ordinateur    5          800,00          € 4000,00       €
1    Souris        20         15,00           € 300,00        €
```

3. Rechercher un produit par nom
    - Lire le nom
    - Afficher ses informations (index, quantité, prix)
4. Mettre à jour la quantité
    - Lire l'index du produit
    - Lire la nouvelle quantité
    - Mettre à jour
5. Calculer la valeur totale du stock
    - Valeur = Σ (quantité × prix) pour tous les produits
6. Identifier les produits en rupture
    - Afficher les produits avec quantité = 0
7. Menu principal (similaire à l'exercice 18)

Contraintes :

- Gérer 3 listes synchronisées
- Gestion d'erreurs (index invalide, produit introuvable)
- Formatage propre de l'affichage

Attendu : Gestion multi-listes, recherche, calculs, formatage

EXERCICE 20 – Analyse de données de vente (Synthèse algorithmique)

Contexte : Analyser les performances commerciales d'un vendeur sur 14 jours.

Données :

```
ArrayList<Double> ventesQuotidiennes = new ArrayList<>(Arrays.asList(
    150.0, 200.0, 120.0, 300.0, 180.0, 220.0, 250.0,
    190.0, 280.0, 210.0, 160.0, 290.0, 230.0, 270.0
));
```

Travail demandé :

1. Statistiques de base :
    - Vente totale
    - Vente moyenne
    - Vente maximale et jour correspondant
    - Vente minimale et jour correspondant
2. Analyse de tendance :
    - Nombre de jours où ventes > moyenne
    - Pourcentage de jours "performants" (>moyenne)
3. Objectif de vente :
    - Objectif = 250€/jour
    - Compter combien de jours ont atteint l'objectif
    - Taux d'atteinte de l'objectif (en %)
4. Top 3 des meilleures journées :
    - Trier une copie de la liste (sans modifier l'originale)
    - Afficher les 3 meilleures ventes
5. Répartition par tranches :
    - Compter les jours dans chaque tranche :
        - 0-150€ : [X] jours
        - 151-200€ : [Y] jours
        - 201-250€ : [Z] jours
        - 250€ : [W] jours

```
=== ANALYSE DES VENTES ===
```

Vente totale : 3020.0€
Vente moyenne : 215.71€
Meilleure vente : 300.0€ (jour 4)
Pire vente : 120.0€ (jour 3)

Jours > moyenne : 8 (57.14%)
Objectif atteint : 5 jours (35.71%)

Top 3 des ventes : 300.0€, 290.0€, 280.0€

Répartition :
0-150€ : 2 jours
151-200€ : 4 jours
201-250€ : 3 jours
>250€ : 5 jours