CORRIGÉS ET CODE COMPLET

public class Groupe { public String nom; private double cachet; public int nbMembres;

```
public Groupe(String nom, int nbMembres) {
    this.nom = nom;
    this.nbMembres = nbMembres;
    this.cachet = 0;
}

public double getCachet() {
    return cachet;
}

public void setCachet(double c) {
    if (c > 0) {
        this.cachet = c;
    } else {
        System.out.println("Erreur: le cachet doit être positif");
    }
}

public double partParMusicien() {
    if (nbMembres == 0) {
        System.out.println("Erreur: groupe sans membres");
        return 0;
    }
    return cachet / nbMembres;
}

public String toString() {
    return nom + " (" + nbMembres + " membres) - Cachet: " + cachet +
"€";
}
```

}

// TESTS EXERCICE 1

Groupe g = new Groupe(“Les Rockers”, 4);

g.setCachet(2000);

System.out.println(g);

System.out.println(“Part par musicien:” + g.partParMusicien() + “€”);

// Test validation g.setCachet(-500);

// Doit afficher erreur

// Test accès direct (doit échouer ou être impossible selon JShell)

// g.cachet = 1000; // Erreur: cachet has private access

Réponses aux questions

Q1 : g.cachet est inaccessible car private. JShell affiche : cachet has private access in Groupe

Q2 : Le setter permet de valider les données (cachet > 0). Si public, n'importe quelle classe pourrait mettre un cachet négatif ou aberrant.

Q3 : Division par zéro ! Solution : vérifier nbMembres > 0 dans partParMusicien() et le constructeur.

// Recopiez d'abord la classe Groupe ci-dessus

public class GroupePro extends Groupe {

public String manager;

private static final double COMMISSION = 0.20;

```
public GroupePro(String nom, int nbMembres, String manager) {
    super(nom, nbMembres);
    this.manager = manager;
}

public double commissionManager() {
    return getCachet() * COMMISSION;
}

@Override
public double partParMusicien() {
    double reste = getCachet() * (1 - COMMISSION);
    return reste / nbMembres;
}

@Override
public String toString() {
    return super.toString() + " [Manager: " + manager + "]";
}
```

}

// TESTS EXERCICE 2

Groupe amateurs = new Groupe("Les Debutants", 3);

amateurs.setCachet(1500);

GroupePro pros = new GroupePro("The Stars", 5, "John Smith");

pros.setCachet(10000);

System.out.println(“=== AMATEURS ===”); System.out.println(amateurs);
System.out.println(“Part/musicen:” + amateurs.partParMusicien() + “€”);

System.out.println(“=== PROS ===”); System.out.println(pros);
System.out.println(“Commission:” + pros.commissionManager() + “€”);
System.out.println(“Part/musicen:” + pros.partParMusicien() + “€”);

Réponses aux questions

Q1 : super(…) appelle le constructeur parent pour initialiser nom et nbMembres. Doit être la première instruction.

Q2 : Si cachet était privé sans getter, GroupePro ne pourrait pas y accéder. L’héritage ne donne pas accès aux membres privés du parent.

Q3 : Non. Une variable Groupe ne connaît que les méthodes déclarées dans Groupe. Il faudrait caster : ((GroupePro)g).commissionManager().

// Recopiez Groupe et GroupePro d’abord

import java.util.ArrayList; import java.util.List;

public class Organisateur { private List programmation = new ArrayList<>();

```
public void ajouter(Groupe g) {
    programmation.add(g);
}

public double cachetTotal() {
    double total = 0;
    for (Groupe g : programmation) {
        total += g.getCachet();
    }
    return total;
}

public void fichePaie() {
    System.out.println("=== FICHES DE PAIE ===");
    for (Groupe g : programmation) {
        System.out.println("\nGroupe: " + g.nom);
        System.out.println("  Cachet brut: " + g.getCachet() + "€");

        // Détection du type avec instanceof
        if (g instanceof GroupePro) {
            GroupePro gp = (GroupePro) g;
            System.out.println("  Commission " + gp.manager + ": "
                + gp.commissionManager() + "€");
        }

        System.out.println("  Part/musicen: " + g.partParMusicien() +
```

```
"€");
    }
    System.out.println("\nTOTAL: " + cachetTotal() + "€");
}
```

}

// TESTS EXERCICE 3 Organisateur orga = new Organisateur();

Groupe g1 = new Groupe(“Garage Band”, 3); g1.setCachet(1200);

GroupePro g2 = new GroupePro(“Super Star”, 4, “Big Boss”); g2.setCachet(15000);

Groupe g3 = new Groupe(“Les Copains”, 5); g3.setCachet(800);

orga.ajouter(g1); orga.ajouter(g2); orga.ajouter(g3);

orga.fichePaie();

Réponses aux questions

Q1 : Un GroupePro EST un Groupe. La liste accepte tout objet qui “est un” Groupe.

Q2 : Liaison dynamique. À l’exécution, la JVM regarde le type réel de l’objet et appelle sa méthode. C’est le polymorphisme.

Q3 : Non, instanceof viole le principe Ouvert/Fermé. On doit modifier Organisateur à chaque nouveau type de groupe.

// MODIFICATION DE GROUPE

public String detailsRepartition() {

return “Part égale:” + String.format(“%.2f”, partParMusicien()) + “€ par musicien”;

}

// MODIFICATION DE GROUPEPRO (ajoutez @Override)

@Override

public String detailsRepartition() {

return “Manager (” + manager + “):” + String.format(“%.2f”, commissionManager()) + “€,” + “musiciens:” + String.format(“%.2f”, partParMusicien()) + “€ chacun”; }

// NOUVELLE VERSION DANS ORGANISATEUR

public void fichePaieV2() {

System.out.println(“=== FICHES DE PAIE V2 (sans instanceof) ===”);

for (Groupe g : programmation) { System.out.println(g.nom + “:” + g.detailsRepartition()); } System.out.println(“TOTAL:” + cachetTotal() + “€”); }

// TEST orga.fichePaieV2();

Réponses aux questions

Q1 : Principe Ouvert/Fermé (Open/Closed) : ouvert à l'extension (nouveaux types de groupes), fermé à la modification (Organisateur ne change plus).

Q2 : Il suffirait de créer GroupeTribute extends Groupe et override detailsRepartition(). Organisateur fonctionnera immédiatement sans modification.
