Table des matières

1. Introduction à la Programmation Orientée Objet

1.1 Évolution historique

- Apparition dans les années 60
- Réponse aux limitations de la programmation structurée
- Évolution des langages : Simula → Smalltalk → C++ → Java

1.2 La programmation structurée (contexte historique)

- **Organisation classique** : - 🔹 Structures (types et variables) - 🔹 Opérations (procédures) - 🔹 Programme principal - **Limitations majeures** : - 🔸 Séparation artificielle données/traitements - 🔸 Difficulté de maintenance - 🔸 Faible réutilisabilité du code

2. Concepts Fondamentaux

2.1 L'Abstraction

- **Définition** : Processus d'identification des caractéristiques essentielles - **Composants** : - 🔷 Classe : modèle conceptuel - 🔷 Objet : instance concrète - **Structure d'un objet** : - 🔹 Identité (identificateur unique) - 🔹 État (données/attributs) - 🔹 Comportement (méthodes)

2.2 L'Encapsulation

- **Principe** : "Masquage de l'information" - **Structure** : - 🔷 Interface publique (visible) - 🔷 Implémentation privée (cachée) - **Avantages** : - 🔹 Protection des données - 🔹 Contrôle d'accès - 🔹 Maintenance facilitée

2.3 La Classification

A) Généralisation - 🔷 Regroupement par points communs - 🔷 Création de classes abstraites - 🔷 Exemple : Voiture, Moto → Véhicule

B) Spécialisation

- 🔷 Création de sous-classes spécifiques
- 🔷 Ajout de caractéristiques particulières
- 🔷 Exemple : Animal → Mammifère → Chien

C) Héritage

- 🔷 Relation "est un"
- 🔷 Transmission des propriétés
- 🔷 Support du polymorphisme

D) Surcharge

- 🔷 Redéfinition des propriétés
- 🔷 Adaptation des comportements
- 🔷 Extension des fonctionnalités

3. Principes Avancés

3.1 Le Polymorphisme

- **Types** : - 🔷 Polymorphisme d'héritage - 🔷 Polymorphisme d'interface - 🔷 Polymorphisme paramétrique - **Caractéristiques** : - 🔷 Liaison dynamique - 🔷 Comportements multiples - 🔷 Flexibilité du code

3.2 L'Agrégation

- **Définition** : Relation "partie de" - **Caractéristiques** : - 🔷 Composition d'objets - 🔷 Dépendance forte/faible - 🔷 Cycle de vie lié/indépendant

3.3 La Coopération

- **Principes** : - 🔷 Interaction entre objets - 🔷 Messages et responsabilités - 🔷 Cohésion et couplage - **Application** : - 🔷 Définition des interfaces - 🔷 Protocoles de communication - 🔷 Gestion des dépendances

Questions de Cours Approfondies (30 points)

Questions Conceptuelles (15 points)

1. Expliquez la différence entre classe et objet avec un exemple concret. (2 pts)
2. Comment l'encapsulation contribue-t-elle à la sécurité du code ? (2 pts)
3. Quels sont les avantages et inconvénients de l'héritage multiple ? (2 pts)
4. Expliquez le concept de liaison dynamique avec un exemple. (2 pts)
5. Comment le polymorphisme facilite-t-il l'extension du code ? (2 pts)
6. Quelle est la différence entre agrégation et composition ? (2 pts)
7. Pourquoi dit-on que la POO est plus proche de la réalité ? (3 pts)

Questions Pratiques (15 points)

1. Donnez un exemple concret de surcharge de méthode. (2 pts)
2. Comment implémenteriez-vous une relation "est un" vs "a un" ? (2 pts)
3. Expliquez comment vous concevriez une hiérarchie de classes pour un système de gestion de bibliothèque. (3 pts)
4. Comment utiliseriez-vous le polymorphisme pour gérer différents types de paiements dans un système e-commerce ? (3 pts)
5. Proposez une solution utilisant l'agrégation pour modéliser une université avec ses départements et professeurs. (3 pts)
6. Décrivez un cas où l'abstraction serait particulièrement utile dans un système de réservation. (2 pts)

Réponses aux questions précédentes

Réponses détaillées

1. La classe est un modèle, l'objet une instance concrète. Exemple : Classe `Voiture` (modèle) → Objet `maVoiture` (instance spécifique avec ses propres caractéristiques).
2. L'encapsulation cache les détails d'implémentation et protège les données en contrôlant leur accès via des méthodes dédiées.
3. **Avantages et inconvénients de l'héritage multiple :**
   - **Avantages :**
     - Réutilisation du code provenant de plusieurs classes parentales.
     - Modularité accrue.
     - Réduction de la redondance.
   - **Inconvénients :**
     - Conflits d'héritage.
     - Complexité accrue.
     - Certains langages comme Java ne le supportent pas.
4. **Concept de liaison dynamique avec un exemple :**
   - La liaison dynamique résout l'appel d'une méthode à l'exécution.

```
class Animal {
    void faireSon() {
        System.out.println("L'animal fait un bruit.");
    }
}
class Chien extends Animal {
```

```
    void faireSon() {
        System.out.println("Le chien aboie.");
    }
}
public class Test {
    public static void main(String[] args) {
        Animal monAnimal = new Chien();
        monAnimal.faireSon(); // "Le chien aboie."
    }
}
```

5. **Comment le polymorphisme facilite l'extension du code :**
   - Il permet d'écrire du code générique qui fonctionne avec plusieurs types d'objets, facilitant ainsi l'ajout de nouvelles fonctionnalités sans modifier le code existant.
6. **Différence entre agrégation et composition :**
   - **Agrégation :** Relation "a un", l'objet contenu peut exister indépendamment.
   - **Composition :** Relation "fait partie de", l'objet contenu dépend entièrement de l'objet contenant.
7. **Pourquoi la POO est plus proche de la réalité ?**
   - Elle reflète la manière dont nous percevons le monde en utilisant des objets, leurs états et comportements.
8. **Exemple concret de surcharge de méthode :**

```
class Calculateur {
    int additionner(int a, int b) { return a + b; }
    double additionner(double a, double b) { return a + b; }
}
```

9. **Implémentation d'une relation "est un" vs "a un" :**
   - **"Est un"** → Héritage :

```
class Vehicule { ... }
class Voiture extends Vehicule { ... }
```

   - **"A un"** → Agrégation/Composition :

```
class Moteur { ... }
class Voiture { private Moteur moteur; }
```

10. **Hiérarchie de classes pour un système de gestion de bibliothèque :**

```
class Document { ... }
class Livre extends Document { ... }
class Magazine extends Document { ... }
```

11. **Utilisation du polymorphisme pour gérer différents types de paiements dans un système e-commerce :**

```
interface Paiement { void effectuerPaiement(); }
class PaiementCarte implements Paiement {
    public void effectuerPaiement() { System.out.println("Paiement par carte."); }
}
class PaiementPaypal implements Paiement {
    public void effectuerPaiement() { System.out.println("Paiement via PayPal."); }
}
```

12. **Solution utilisant l'agrégation pour modéliser une université :**

```
class Departement { ... }
class Professeur { ... }
class Universite {
    private List<Departement> departements;
    private List<Professeur> professeurs;
}
```

13. **Cas où l'abstraction serait utile dans un système de réservation :**

    - Un système de réservation peut utiliser une classe abstraite `Reservation` avec des sous-classes comme `ReservationAvion`, `ReservationHotel`, `ReservationTrain` pour assurer une gestion uniforme.

Évaluation Complète (50 points)

Partie 1 : QCM Initial (20 points)

1. Qu'est-ce qui caractérise la programmation structurée ? (2pts)
   - a) L'encapsulation des données
   - b) La séparation des données et des traitements ✓
   - c) L'héritage entre classes
   - d) Le polymorphisme
2. Un objet est constitué de : (2pts)
   - a) Une classe et des méthodes
   - b) Une identité, un état et un comportement ✓
   - c) Des attributs uniquement
   - d) Des méthodes uniquement
3. Le concept d'encapsulation permet de : (2pts)
   - a) Créer des objets
   - b) Cacher l'implémentation interne ✓
   - c) Hériter des propriétés
   - d) Agréger des classes
4. Le polymorphisme représente : (2pts)
   - a) L'héritage multiple
   - b) La capacité à avoir plusieurs classes
   - c) La faculté d'une opération à s'appliquer à des objets de classes différentes ✓
   - d) La création d'objets
5. L'agrégation correspond à une relation de type : (2pts)
   - a) "est un"
   - b) "partie de" ✓
   - c) "hérite de"
   - d) "implémente"
6. Dans l'approche objet, la fonction est considérée comme : (2pts)
   - a) Le premier principe d'intelligibilité
   - b) Un critère de perception
   - c) Un critère de pertinence ✓
   - d) Le principe fondamental
7. Les avantages de l'approche objet incluent : (2pts)
   - a) La rigidité du code
   - b) La réutilisabilité ✓
   - c) La centralisation des données
   - d) La programmation linéaire
8. Une des difficultés de l'approche objet est : (2pts)
   - a) La simplicité excessive
   - b) Le manque de flexibilité
   - c) La modélisation des aspects dynamiques ✓
   - d) L'impossibilité d'héritage
9. La classification en POO s'appuie sur : (2pts)
   - a) Uniquement la généralisation
   - b) Uniquement la spécialisation

- c) La généralisation et la spécialisation ✓
- d) Ni l'un ni l'autre

10. L'abstraction permet de : (2pts)

- a) Uniquement cacher les données
- b) Représenter les caractéristiques essentielles d'une entité ✓
- c) Créer uniquement des interfaces
- d) Supprimer des fonctionnalités

Partie 2 : Questions de Cours Approfondies (30 points)

Réponses Détaillées aux Questions Conceptuelles

1. **Différence entre classe et objet (2 pts)**
   - La classe est un modèle abstrait qui définit :
     - La structure (attributs)
     - Les comportements (méthodes)
   - L'objet est une instance concrète de la classe :
     - Exemple : Classe Voiture (modèle) → Objet maVoiture (Renault Clio rouge de 2020)
     - Les objets ont des valeurs spécifiques pour chaque attribut
2. **Contribution de l'encapsulation à la sécurité (2 pts)**
   - Protection des données :
     - Accès contrôlé via des méthodes (getters/setters)
     - Validation des données avant modification
   - Masquage de l'implémentation :
     - L'utilisateur ne voit que l'interface publique
     - Prévient les manipulations directes dangereuses
3. **Avantages et inconvénients de l'héritage multiple (2 pts)**
   - Avantages :
     - Réutilisation maximale du code
     - Modélisation de concepts complexes
   - Inconvénients :
     - Conflits de noms (problème du diamant)
     - Complexité accrue de la maintenance
     - Risque de confusion dans la hiérarchie
4. **Liaison dynamique avec exemple (2 pts)**

```
class Animal {
    void faireBruit() { System.out.println("???"); }
}
class Chien extends Animal {
    void faireBruit() { System.out.println("Wouf!"); }
}
class Chat extends Animal {
    void faireBruit() { System.out.println("Miaou!"); }
}
```

```
Animal animal1 = new Chien(); // Le choix de la méthode se fait à l'exécution
animal1.faireBruit(); // Affiche "Wouf!"
```

5. **Polymorphisme et extension du code (2 pts)**
   - Permet d'ajouter de nouveaux comportements sans modifier le code existant
   - Facilite l'ajout de nouvelles classes dérivées
   - Maintient la cohérence des interfaces
   - Exemple : Ajout d'un nouveau type de paiement sans modifier le système existant
6. **Différence entre agrégation et composition (2 pts)**
   - Agrégation (relation faible) :
     - Les objets peuvent exister indépendamment
     - Exemple : Université et Professeurs
   - Composition (relation forte) :
     - L'objet composé ne peut exister sans ses composants
     - Exemple : Maison et Pièces
7. **POO et réalité (3 pts)**
   - Modélisation naturelle :
     - Les objets du monde réel deviennent des classes
     - Les interactions deviennent des méthodes
   - Organisation intuitive :
     - Hiérarchie reflétant la réalité
     - Relations entre objets explicites
   - Maintenance facilitée :
     - Modifications localisées
     - Évolution naturelle du système

Réponses aux Questions Pratiques

1. **Exemple de surcharge de méthode (2 pts)**

```
class Calculator {
    int add(int a, int b) { return a + b; }
    double add(double a, double b) { return a + b; }
    int add(int a, int b, int c) { return a + b + c; }
}
```

2. **Implémentation "est un" vs "a un" (2 pts)**

```
// "est un" (héritage)
class Voiture extends Vehicule { }

// "a un" (agrégation)
class Voiture {
    private Moteur moteur;
}
```

3. **Système de gestion de bibliothèque (3 pts)**

```
abstract class Document {
    private String titre;
    private String reference;
    abstract void emprunter();
}
```

```
class Livre extends Document {
    private String auteur;
    private String ISBN;
}

class Revue extends Document {
    private int numero;
    private Date datePublication;
}
```

4. **Polymorphisme pour système de paiement (3 pts)**

```
interface Paiement {
    boolean processer(double montant);
}

class PaiementCarte implements Paiement {
    public boolean processer(double montant) {
        // Logique spécifique carte
    }
}

class PaiementPayPal implements Paiement {
    public boolean processer(double montant) {
        // Logique spécifique PayPal
    }
}
```

5. **Modélisation d'université avec agrégation (3 pts)**

```
class Universite {
    private List<Departement> departements;
    private List<Professeur> professeurs;
}

class Departement {
    private String nom;
    private List<Professeur> professeurs;
}

class Professeur {
    private String nom;
    private Departement departementPrincipal;
}
```

13. **Abstraction dans système de réservation (2 pts)**

```
abstract class Reservation {
    protected Date date;
    protected String client;

    abstract boolean verifierDisponibilite();
    abstract void confirmer();
    abstract void annuler();
}

class ReservationHotel extends Reservation {
    private int numeroChambre;
```

```
        // Implémentation spécifique
    }
```