Examen Analyste Programmeur – Niveau Intermédiaire

AVEC RÉPONSES COMPLÈTES

UML

1. Quelle est la différence entre un diagramme de classes et un diagramme de séquence ?

Réponse: - **Diagramme de classes:** Structure statique du système (classes, attributs, méthodes, relations) - **Diagramme de séquence:** Comportement dynamique (interactions temporelles entre objets)

Exemple bancaire: - Classes: `Compte`, `Client`, `Transaction` - Séquence: Flow d'un virement entre comptes

---

2. À quoi sert une relation d'agrégation en UML ?

Réponse: Relation "a un" où la partie peut exister indépendamment du tout (losange vide).

Exemple: `Banque -- Compte` - Si la banque ferme, les comptes peuvent être transférés ailleurs

---

3. Quand utiliser un diagramme d'activités ?

Réponse: Pour modéliser des workflows et processus métier.

Cas d'usage: - Processus d'ouverture de compte - Workflow d'approbation de prêt - Flux de traitement de transaction

---

4. Expliquez la notion d'héritage en UML.

Réponse: Relation "est un" où une classe enfant hérite des attributs/méthodes du parent (flèche creuse).

Exemple:

```
Compte (parent)
  ↑
    CompteCourant
    CompteEpargne
```

---

5. Quelle est l'utilité des interfaces en UML ?

Réponse: Contrat définissant des méthodes que les classes doivent implémenter (sans implémentation).

Exemple: `interface Payable { pay(amount) }` - Implémenté par: `CreditCard`, `BankTransfer`, `Cash`

---

6. Différence entre association et composition ?

Réponse: - **Association (——):** Relation simple entre classes - **Composition (——):** Relation forte, la partie ne peut exister sans le tout

Exemple: - Association: `Client -- Compte` (client peut exister sans compte) - Composition: `Compte -- Transaction` (transaction n'existe pas sans compte)

---

7. Que représente une multiplicité ?

Réponse: Nombre d'instances dans une relation.

Exemples: - `1..1` : exactement 1 - `0..1` : 0 ou 1 - `1..*` : au moins 1 - `0..*` ou `*` : plusieurs

Cas: `Client 1 -- 0..* Compte` (1 client peut avoir plusieurs comptes)

---

8. Diagramme UML le plus adapté pour modéliser un workflow ?

Réponse: Diagramme d'activités (avec swimlanes pour montrer les responsabilités).

Alternative: Diagramme d'états pour les états d'une entité spécifique.

Networking

9. Expliquez le modèle OSI.

Réponse: Modèle en 7 couches pour la communication réseau:

Couche	Nom	Rôle
7	Application	HTTP, FTP, DNS
6	Présentation	Chiffrement, compression

Couche	Nom	Rôle
5	Session	Gestion sessions
4	Transport	TCP, UDP
3	Réseau	Routage IP
2	Liaison	Ethernet
1	Physique	Câbles

Mnémonique: “Please Do Not Throw Sausage Pizza Away”

---

10. Différence entre TCP et UDP.

Réponse:

TCP	UDP
Connexion établie	Sans connexion
Fiable (garanties)	Pas de garantie
Ordonné	Peut être désordonné
Plus lent	Plus rapide
Web, Email	Streaming, DNS

Cas bancaire: TCP pour transactions (fiabilité critique)

---

11. À quoi sert le DNS ?

Réponse: Domain Name System: Traduit noms de domaine en adresses IP.

Exemple: `www.banque.ht` → `192.168.1.100`

---

12. Qu’est-ce qu’une adresse IP privée ?

Réponse: Adresse utilisée dans un réseau local (non routable sur Internet).

Plages: `- 10.0.0.0 - 10.255.255.255 - 172.16.0.0 - 172.31.255.255 - 192.168.0.0 - 192.168.255.255`

---

13. Différence entre HTTP et HTTPS ?

Réponse:

HTTP	HTTPS
Port 80	Port 443
Non chiffré	Chiffré (SSL/TLS)
Vulnérable	Sécurisé

Bancaire: HTTPS obligatoire pour protéger données sensibles.

---

14. Qu'est-ce qu'un pare-feu ?

Réponse: Système filtrant le trafic réseau selon des règles de sécurité.

Types: - Pare-feu matériel - Pare-feu logiciel - WAF (Web Application Firewall)

---

15. Expliquez le rôle d'un load balancer.

Réponse: Distribue le trafic entre plusieurs serveurs pour: - Éviter surcharge - Haute disponibilité - Améliorer performance

Algorithmes: Round-robin, least connections, IP hash

---

16. Qu'est-ce que la latence réseau ?

Réponse: Délai de transmission des données entre source et destination (en millisecondes).

Facteurs: - Distance physique - Congestion réseau - Qualité connexion

---

OOP

17. Expliquez les quatre piliers de la POO.

Réponse:

1. **Encapsulation:** Cacher les détails internes

```python
class Compte:
    def __init__(self):
        self.__solde = 0  # Privé
```

2. **Héritage:** Réutiliser code parent

   ```
   class CompteEpargne(Compte):
       pass
   ```

3. **Polymorphisme:** Même interface, comportements différents

   ```
   compte.calculer_interets()  # Différent selon type
   ```

4. **Abstraction:** Masquer complexité

   ```
   compte.retirer(100)  # Cache vérifications internes
   ```

---

18. Différence entre abstraction et encapsulation ?

Réponse: - **Abstraction:** Cacher complexité (QUOI faire) - **Encapsulation:** Cacher implémentation (COMMENT faire)

Exemple: - Abstraction: Interface `Payable` avec `pay()` - Encapsulation: Méthode privée `__validate_payment()`

---

19. Qu'est-ce que le polymorphisme ?

Réponse: Capacité d'un objet à prendre plusieurs formes.

Types: 1. **Surcharge (overloading):** Mêmes méthodes, paramètres différents 2. **Redéfinition (overriding):** Réimplémenter méthode parent

```
class Compte:
    def calculer_frais(self):
        return 5.0

class ComptePremium(Compte):
    def calculer_frais(self):  # Override
        return 0.0
```

---

20. Quand utiliser une classe abstraite ?

Réponse: Quand on veut: - Définir un contrat avec implémentation partielle - Empêcher instanciation directe - Partager code commun

```
from abc import ABC, abstractmethod

class Compte(ABC):
    @abstractmethod
    def calculer_interets(self):
```

```
        pass

    def afficher_solde(self):  # Implémenté
        print(self.solde)
```

---

21. Différence entre héritage et composition ?

Réponse:

Héritage	Composition
"est un"	"a un"
Couplage fort	Couplage faible
Moins flexible	Plus flexible

Exemple:

```
# Héritage
class CompteCourant(Compte):
    pass

# Composition
class Compte:
    def __init__(self):
        self.historique = Historique()
```

Principe: Préférer composition à l'héritage.

---

22. Qu'est-ce qu'un constructeur ?

Réponse: Méthode spéciale appelée lors de la création d'un objet.

```
class Compte:
    def __init__(self, numero, titulaire):
        self.numero = numero
        self.titulaire = titulaire
        self.solde = 0.0

compte = Compte("123", "Jean")
```

---

23. Avantages de la POO ?

Réponse: 1. **Réutilisabilité:** Héritage, composition 2. **Maintenabilité:** Code modulaire 3. **Scalabilité:** Facile à étendre 4. **Sécurité:** Encapsulation 5. **Modélisation naturelle:** Objets du monde réel

---

24. Exemple de surcharge de méthode.

Réponse:

```
class Transaction:
    def creer(self, montant):
        # Avec montant seulement
        pass

    def creer(self, montant, description):
        # Avec montant et description
        pass
```

Note Python: Utiliser paramètres par défaut ou `*args`

DSA

25. Différence entre tableau et liste chaînée.

Réponse:

Tableau	Liste chaînée
Taille fixe	Taille dynamique
Accès O(1)	Accès O(n)
Insertion O(n)	Insertion O(1)
Contigü en mémoire	Non contigü

Utilisation: Tableau pour accès fréquent, liste pour insertions.

---

26. Complexité temporelle de la recherche linéaire.

Réponse: O(n) - Parcourir tous les éléments dans le pire cas.

```
def recherche_lineaire(arr, cible):
    for i in range(len(arr)):  # O(n)
        if arr[i] == cible:
```

```
            return i
    return -1
```

---

27. Qu'est-ce qu'une pile (stack) ?

Réponse: Structure LIFO (Last In, First Out).

Opérations: - `push()`: Ajouter au sommet - O(1) - `pop()`: Retirer du sommet - O(1) - `peek()`: Voir sommet - O(1)

Cas d'usage: Historique navigation, undo/redo

---

28. Qu'est-ce qu'une file (queue) ?

Réponse: Structure FIFO (First In, First Out).

Opérations: - `enqueue()`: Ajouter à la fin - O(1) - `dequeue()`: Retirer du début - O(1)

Cas d'usage: File d'attente transactions, message queue

---

29. Complexité moyenne du tri rapide.

Réponse: O(n log n) en moyenne

Pire cas: $O(n^2)$ si pivot mal choisi

Avantages: Efficace, tri en place

---

30. À quoi sert une table de hachage ?

Réponse: Structure pour stockage clé-valeur avec accès rapide.

Complexité: O(1) en moyenne pour recherche/insertion/suppression

Exemple:

```
comptes = {
    "123": Compte("Jean"),
    "456": Compte("Marie")
}
```

---

31. Différence entre BFS et DFS.

Réponse:

BFS	DFS
Parcours en largeur	Parcours en profondeur
File (queue)	Pile (stack)
Plus court chemin	Explorer branches

Cas: BFS pour réseau bancaire (trouver chemin le plus court)

---

32. Qu'est-ce qu'un arbre binaire ?

Réponse: Arbre où chaque nœud a au maximum 2 enfants (gauche, droit).

Types: - **BST (Binary Search Tree):** Gauche < Parent < Droit - **Complet:** Tous niveaux remplis sauf dernier - **Équilibré:** Hauteur minimale

---

Design Patterns

33. À quoi sert le pattern Singleton ?

Réponse: Garantir qu'une classe n'a qu'une seule instance.

```
class Config:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance
```

Cas: Configuration bancaire globale

---

34. Différence entre Factory et Abstract Factory.

Réponse: - **Factory:** Créer objets d'un type - **Abstract Factory:** Créer familles d'objets cohérentes

Exemple: - Factory: `CompteFactory.create("COURANT")` - Abstract Factory: `BanqueRetailFactory` crée `CompteCourant` + `CarteDebit`

---

35. Quand utiliser Observer ?

Réponse: Quand un objet doit notifier automatiquement plusieurs observateurs.

Cas bancaire: Notification de transaction - Subject: `Compte` - Observers: `EmailNotifier`, `SMSNotifier`, `PushNotifier`

---

36. Avantages du pattern Strategy.

Réponse: 1. Algorithmes interchangeables à l'exécution 2. Évite conditions multiples (if/else) 3. Open/Closed principle

```
class Transaction:
    def __init__(self, fee_strategy):
        self.fee_strategy = fee_strategy

    def calculate_fee(self):
        return self.fee_strategy.calculate()
```

---

37. Pattern MVC : rôle du contrôleur ?

Réponse: Intermédiaire entre Modèle et Vue: - Reçoit requêtes utilisateur - Appelle la logique métier (Modèle) - Met à jour la Vue

Flux: Vue → Contrôleur → Modèle → Contrôleur → Vue

---

38. Pattern Adapter : cas d'usage.

Réponse: Adapter une interface incompatible.

Exemple:

```
class OldPaymentSystem:
    def process(self, amount):
        pass

class PaymentAdapter:
    def __init__(self, old_system):
        self.old_system = old_system

    def pay(self, amount):  # Nouvelle interface
        return self.old_system.process(amount)
```

---

39. Différence entre State et Strategy ?

Réponse: - **State:** Comportement change selon l'état interne - **Strategy:** Algorithme choisi par le client

Exemple: - State: `Compte` (ACTIF, SUSPENDU, FERMÉ) - Strategy: Calcul de frais (STANDARD, PREMIUM, VIP)

---

40. Pourquoi éviter l'abus de Singleton ?

Réponse: Problèmes: 1. État global (couplage fort) 2. Difficile à tester (mock) 3. Concurrence (thread-safety) 4. Violation Single Responsibility

Alternative: Dependency Injection

Backend Patterns

41. Qu'est-ce qu'une architecture REST ?

Réponse: Style d'architecture pour APIs basé sur HTTP.

Principes: - Stateless - Ressources (URIs) - Méthodes HTTP (GET, POST, PUT, DELETE) - Représentations (JSON)

Exemple:

```
GET    /comptes/123
POST   /comptes
PUT    /comptes/123
DELETE /comptes/123
```

---

42. Différence entre monolithe et microservices.

Réponse:

Monolithe	Microservices
Application unique	Services indépendants
Déploiement complet	Déploiement par service
Simple au début	Scalabilité granulaire
Couplage fort	Couplage faible

---

43. À quoi sert un API Gateway ?

Réponse: Point d'entrée unique pour les microservices.

Rôles: - Routage des requêtes - Authentification centralisée - Rate limiting - Load balancing - Agrégation de réponses

---

44. Qu'est-ce que la pagination côté backend ?

Réponse: Diviser résultats en pages pour performance.

```
{
  "items": [...],
  "page": 1,
  "page_size": 20,
  "total": 1000,
  "total_pages": 50
}
```

Implémentation:

```
skip = (page - 1) * page_size
results = query.offset(skip).limit(page_size)
```

---

45. Gestion des erreurs dans une API ?

Réponse: Utiliser codes HTTP appropriés + messages clairs.

```
{
  "error": "INSUFFICIENT_BALANCE",
  "message": "Solde insuffisant pour cette transaction",
  "details": {
    "available": 500,
    "requested": 1000
  }
}
```

Codes: 400 (client), 500 (serveur)

---

46. Qu'est-ce que l'idempotence ?

Réponse: Opération donnant le même résultat si exécutée plusieurs fois.

Idempotent: - GET, PUT, DELETE (plusieurs appels = même effet)

Non-idempotent: - POST (crée à chaque fois)

---

47. Rôle des middlewares.

Réponse: Fonctions exécutées entre requête et réponse.

Exemples: - Authentification - Logging - CORS - Rate limiting - Compression

```python
@app.middleware("http")
async def log_requests(request, call_next):
    # Avant traitement
    response = await call_next(request)
    # Après traitement
    return response
```

---

48. Qu'est-ce qu'un service stateless ?

Réponse: Service ne conservant pas d'état entre requêtes.

Avantages: - Scalabilité horizontale facile - Pas de session serveur - Load balancing simple

État stocké: Base de données, cache (Redis)

SQL & Base de données

49. Différence entre DELETE et TRUNCATE.

Réponse:

DELETE	TRUNCATE
Supprime lignes spécifiques	Supprime toutes lignes
Avec WHERE	Pas de WHERE
Peut rollback	Pas de rollback
Plus lent	Plus rapide
Trigger activé	Pas de trigger

---

50. À quoi sert une clé primaire ?

Réponse: Identifiant unique pour chaque ligne.

Propriétés: - Unique - Non NULL - Un seul par table - Index automatique

```sql
CREATE TABLE comptes (
    id INT PRIMARY KEY,
    numero VARCHAR(20) UNIQUE
);
```

---

51. Qu'est-ce qu'une clé étrangère ?

Réponse: Référence à la clé primaire d'une autre table (intégrité référentielle).

```sql
CREATE TABLE transactions (
    id INT PRIMARY KEY,
    compte_id INT,
    FOREIGN KEY (compte_id) REFERENCES comptes(id)
);
```

---

52. Différence entre INNER JOIN et LEFT JOIN.

Réponse:

INNER JOIN	LEFT JOIN
Lignes communes	Toutes lignes gauche
Exclut non-match	NULL si pas de match

```sql
-- INNER: Seulement clients avec comptes
SELECT * FROM clients c
INNER JOIN comptes co ON c.id = co.client_id;

-- LEFT: Tous clients, même sans compte
SELECT * FROM clients c
LEFT JOIN comptes co ON c.id = co.client_id;
```

---

53. Qu'est-ce qu'une transaction ?

Réponse: Séquence d'opérations traitées comme une unité (tout ou rien).

```sql
BEGIN;
UPDATE comptes SET solde = solde - 1000 WHERE id = 1;
UPDATE comptes SET solde = solde + 1000 WHERE id = 2;
COMMIT;
-- Si erreur: ROLLBACK;
```

ACID: Atomicity, Consistency, Isolation, Durability

---

54. Niveaux d'isolation des transactions.

Réponse:

Niveau	Description	Problème évité
READ UNCOMMITTED	Lit données non validées	-
READ COMMITTED	Lit données validées	Dirty read
REPEATABLE READ	Lecture répétable	Non-repeatable read
SERIALIZABLE	Isolation complète	Phantom read

Défaut PostgreSQL: READ COMMITTED

---

55. Index : avantages et inconvénients.

Réponse:

Avantages: - Accélère recherches (WHERE, JOIN) - Améliore performance SELECT

Inconvénients: - Ralentit INSERT/UPDATE/DELETE - Consomme espace disque - Maintenance nécessaire

```
CREATE INDEX idx_compte_numero ON comptes(numero);
```

---

56. Qu'est-ce que la normalisation ?

Réponse: Organiser données pour réduire redondance.

Formes normales: - **1NF:** Valeurs atomiques - **2NF:** Pas de dépendance partielle - **3NF:** Pas de dépendance transitive

Objectif: Éviter anomalies d'insertion/mise à jour/suppression

---

Frontend Patterns

57. Qu'est-ce que MVC côté frontend ?

Réponse: Pattern séparant logique en 3 composants: - **Model:** Données - **View:** Interface utilisateur - **Controller:** Logique de contrôle

Framework: AngularJS (historique)

---

58. Différence entre SPA et MPA.

Réponse:

SPA	MPA
Single Page Application	Multi-Page Application
Chargement initial	Rechargement pages
Rapide après chargement	Plus lent
Complexe (JS)	Plus simple

Exemple SPA: React, Vue, Angular

---

59. Rôle d'un state manager.

Réponse: Gérer l'état global de l'application de manière centralisée.

Avantages: - État prévisible - Debugging facilité - Time-travel debugging

Outils: Redux, Vuex, Zustand

---

60. Qu'est-ce que le data binding ?

Réponse: Synchronisation automatique entre données et UI.

Types: - **One-way:** Données → UI - **Two-way:** Données UI

```
// Vue.js two-way binding
<input v-model="solde">
```

---

61. Avantages des composants réutilisables.

Réponse: 1. **DRY:** Don't Repeat Yourself 2. **Maintenabilité:** Modifier un endroit 3. **Testabilité:** Tester isolément 4. **Cohérence:** UI uniforme

Exemple: Composant `<Button>` réutilisé partout

---

62. Qu'est-ce que le lazy loading ?

Réponse: Charger ressources seulement quand nécessaire.

Avantages: - Performance initiale améliorée - Économie bande passante - Meilleure UX

```
// React lazy loading
const Dashboard = lazy(() => import('./Dashboard'));
```

---

63. Gestion des formulaires côté frontend.

Réponse: Étapes: 1. Validation en temps réel 2. Messages d'erreur clairs 3. Désactiver bouton pendant soumission 4. Feedback utilisateur

Librairies: Formik, React Hook Form, Vee-Validate

---

64. Différence entre props et state.

Réponse:

Props	State
Données passées par parent	Données internes
Immuables	Mutables
Lecture seule	Lecture/écriture

```
// Props
<Compte solde={1000} />

// State
const [solde, setSolde] = useState(1000);
```

FIN DE L'EXAMEN - Niveau Intermédiaire

Date: Janvier 2026