Examen Analyste Programmeur (Banque) – 8 Axes – Intermédiaire (Questions + Réponses)

UML

Q1. Association vs composition ?

Réponse : Association = lien logique (cycle de vie indépendant). Composition = relation tout/partie forte (la partie dépend du tout, cycle de vie lié).

Q2. Différence diagramme de classes vs diagramme de séquence ?

Réponse : Le diagramme de classes modélise la structure statique (types, attributs, relations). Le diagramme de séquence modélise les interactions dans le temps (messages, ordre d'appel) pour un scénario.

Q3. Quand utiliser un diagramme d'activités ?

Réponse : Pour modéliser un workflow/processus (décisions, parallélisme), ex. ouverture de compte, traitement de virement.

Q4. Qu'est-ce qu'un stéréotype UML (<>, <>…) ?

Réponse : Extension sémantique pour préciser le rôle d'un élément selon un profil (ex. UML pour logiciel).

Q5. Héritage vs interface (réalisation) en UML ?

Réponse : Héritage = généralisation (est-un). Interface = contrat de comportements sans implémentation (selon langage).

Q6. Diagramme d'états : cas d'usage ?

Réponse : Pour objets à cycle de vie riche (ex. Transaction: INITIÉE→AUTORISÉE→COMPTABILISÉE→ANNULÉE).

Q7. Que représente une dépendance UML (flèche en pointillés) ?

Réponse : Un élément utilise un autre (compile-time / usage). Changement du fournisseur peut impacter le client.

Networking

Q8. Qu'est-ce que le MTU et pourquoi c'est important ?

Réponse : Taille max d'un paquet IP sur un lien. MTU mal aligné ⇒ fragmentation/perf; MSS TCP dépend du MTU.

Q9. DNS : rôle et mécanisme haut niveau ?

Réponse : Résolution nom→IP via hiérarchie (root/TLD/autoritatifs) et cache (TTL).

Q10. Qu'est-ce qu'un reverse proxy ?

Réponse : Proxy côté serveur, termine TLS, fait cache, routage, protection, masquant les services internes.

Q11. TCP vs UDP : principales différences ?

Réponse : TCP est orienté connexion, fiable (ACK, retransmission, ordre). UDP est sans connexion, plus léger, pas de garantie d'ordre/fiabilité.

Q12. Différence entre latence et throughput ?

Réponse : Latence = délai (ms) par requête. Throughput = débit (req/s, Mb/s).

Q13. Load balancer L4 vs L7 ?

Réponse : L4 route sur IP/port (TCP/UDP). L7 comprend HTTP (URL, headers), permet routage avancé, WAF, etc.

Q14. Que fait un pare-feu stateful ?

Réponse : Suit l'état des connexions (table d'états) pour autoriser le trafic de retour et appliquer des règles contextuelles.

OOP

Q15. Héritage vs composition : règle pratique ?

Réponse : Préférer la composition (plus flexible) sauf vraie relation 'est-un' stable.

Q16. Qu'est-ce qu'une interface fonctionnelle ?

Réponse : Interface avec une seule méthode abstraite (Java), compatible lambda.

Q17. 4 piliers de la POO ?

Réponse : Encapsulation, Abstraction, Héritage, Polymorphisme.

Q18. SOLID : que signifie le 'D' ?

Réponse : Dependency Inversion: dépendre d'abstractions, pas de concrétions.

Q19. Encapsulation : bénéfice concret ?

Réponse : Masquer l'état interne, contrôler invariants via méthodes, réduire couplage.

Q20. Qu'est-ce qu'un objet valeur (Value Object) ?

Réponse : Objet identifié par ses valeurs (equals/hashCode), immutable; ex. IBAN, Money.

Q21. Inversion of Control : idée ?

Réponse : Le framework orchestre la création/injection (DI), l'application fournit des composants.

DSA

Q22. BFS vs DFS : quand choisir ?

Réponse : BFS pour plus court chemin non pondéré; DFS pour détection cycles, topologie, backtracking.

Q23. Arbre équilibré (AVL/RB) : intérêt ?

Réponse : Garantir hauteur O(log n) ⇒ opérations O(log n) stables.

Q24. Heap : usage typique ?

Réponse : Priority queue: extraire min/max en O(log n), utilisé dans Dijkstra, scheduling.

Q25. Tri rapide : moyenne vs pire ?

Réponse : Moyenne O(n log n), pire $O(n^2)$ si pivot mauvais; randomisation réduit risque.

Q26. Complexité Big-O : que mesure-t-elle ?

Réponse : Croissance asymptotique du coût (temps/espace) selon n, en ignorant constantes.

Q27. Tableau vs liste chaînée : trade-offs ?

Réponse : Tableau: accès O(1) mais insertion milieu coûteuse. Liste: insert/delete O(1) avec pointeur mais accès O(n).

Q28. Fenêtre glissante : pour quoi ?

Réponse : Sous-tableau/sous-chaîne optimale en O(n) (ex. somme max longueur k).

Design Patterns

Q29. Decorator : bénéfice ?

Réponse : Ajouter responsabilités dynamiquement sans modifier la classe (ex. logging, métriques).

Q30. Adapter : quand ?

Réponse : Intégrer API legacy/tiers avec interface différente (ex. switch de paiement).

Q31. Strategy : cas d'usage bancaire ?

Réponse : Sélection de calcul de frais/commission selon type client/produit.

Q32. Factory Method vs Abstract Factory ?

Réponse : Factory Method crée un produit via héritage. Abstract Factory fournit familles de produits compatibles.

Q33. Command : cas d'usage ?

Réponse : Encapsuler une requête (virement) pour queue/retry/audit.

Q34. Singleton : pourquoi souvent critiqué ?

Réponse : Cache dépendances, rend tests difficiles, état global, problèmes concurrence/ordre d'init.

Q35. Circuit Breaker : pourquoi en microservices ?

Réponse : Éviter surcharge/cascades; ouvrir circuit après erreurs pour donner temps de récupération.

Backend Patterns

Q36. Gestion erreurs API : bonnes pratiques ?

Réponse : Codes HTTP, message clair, trace-id, pas de fuite sensible, mapping exceptions.

Q37. Idempotence : exemple en banque ?

Réponse : POST virement avec idempotency-key : répéter la requête ne double pas l'opération.

Q38. Pagination : offset vs cursor ?

Réponse : Offset simple mais instable/perf; cursor stable et efficace (index, 'seek method').

Q39. Observabilité : 3 piliers ?

Réponse : Logs, métriques, traces distribuées (OpenTelemetry).

Q40. Saga : pourquoi ?

Réponse : Gérer transactions distribuées via compensations (ex. réservation + débit + confirmation).

Q41. Rate limiting : comment ?

Réponse : Token bucket/leaky bucket; protéger API contre abus/DDoS, contrôler coûts.

Q42. AuthN/AuthZ : différence ?

Réponse : AuthN = prouver identité; AuthZ = droits/permissions.

SQL & Bases de données

Q43. DELETE vs TRUNCATE ?

Réponse : DELETE journalise ligne par ligne (WHERE possible). TRUNCATE vide la table rapidement, souvent DDL, réinitialise identity.

Q44. Clé primaire vs unique ?

Réponse : PK identifie ligne (non NULL, 1 par table). UNIQUE impose unicité (peut être multiple, NULL selon SGBD).

Q45. INNER JOIN vs LEFT JOIN ?

Réponse : INNER: intersection. LEFT: toutes lignes de gauche + NULL si pas de match à droite.

Q46. Transactions : ACID ?

Réponse : Atomicité, Cohérence, Isolation, Durabilité.

Q47. Partitionnement : quand ?

Réponse : Très grandes tables (transactions), améliore maintenance/perf par pruning.

Q48. Deadlock : c'est quoi ?

Réponse : Cycle d’attente entre transactions; SGBD choisit une victime et rollback.

Q49. Index : coût caché ?

Réponse : Accélère lecture mais ralentit écritures (INSERT/UPDATE/DELETE) et consomme espace.

Frontend Patterns

Q50. Accessibility : exemple ?

Réponse : Labels associés aux inputs, ARIA quand nécessaire, navigation clavier.

Q51. Sécurité front : XSS ?

Réponse : Injection de script; éviter innerHTML, échapper/sanitizer, CSP.

Q52. Lazy loading : bénéfice ?

Réponse : Réduire bundle initial, améliorer performance (LCP) en chargeant à la demande.

Q53. Performance : re-render inutile ?

Réponse : Memoization, keys stables, découper composants, éviter state global abusif.

Q54. SPA vs MPA ?

Réponse : SPA charge app JS et navigue côté client; MPA recharge pages côté serveur, plus simple SEO par défaut.

Q55. CSRF : mitigation ?

Réponse : SameSite cookies, tokens anti-CSRF, double submit, vérifier origin.

Q56. State management : quand nécessaire ?

Réponse : État partagé complexe (auth, panier), synchronisation entre composants, offline cache.

Questions bonus

Q57. [Bonus] Deadlock : c’est quoi ? (Intermédiaire)

Réponse : Cycle d’attente entre transactions; SGBD choisit une victime et rollback.

Q58. [Bonus] Idempotence : exemple en banque ? (Intermédiaire)

Réponse : POST virement avec idempotency-key : répéter la requête ne double pas l’opération.

Q59. [Bonus] NAT : pourquoi l’utiliser ? (Intermédiaire)

Réponse : Traduction d’adresses privées vers publique, économise IPv4 et masque topologie interne.

Q60. [Bonus] Factory Method vs Abstract Factory ? (Intermédiaire)

Réponse : Factory Method crée un produit via héritage. Abstract Factory fournit familles de produits compatibles.