Examen Conception de Systèmes (Banque) – UML à produire – Intermédiaire (Scénarios + Réponses attendues)

Conception UML – Scénarios bancaires (à dessiner manuellement)

Q1. Concevoir le système de 'Virement Interne' : produire un diagramme de classes (domain) + un diagramme de séquence pour le scénario 'virement validé'.

Réponse : Classes attendues: Customer, Account, Transfer, LedgerEntry, Money (Value Object), TransferService, FraudCheckService (option), NotificationService (option). Relations: Customer 1..* Account; Transfer relie from_account/to_account; LedgerEntry 1..* par Transfer. Séquence: Client→API→TransferService: validate(idempotencyKey), lock accounts, check balance, persist transfer+ledger, commit, publish event.

Q2. Concevoir 'Ouverture de compte' : diagramme d'activités (workflow) incluant KYC/AML et validation de documents.

Réponse : Activités: Saisie demande → Vérif identité (KYC) → Screening AML/PEP → Vérif documents → Décision (accept/reject) → Création Customer/Account → Notification. Ajouter branches d'erreur, timers (documents manquants), et swimlanes (Client, Agent, Services).

Q3. Concevoir 'Carte bancaire' : diagramme d'états pour Card (émise, activée, suspendue, bloquée, expirée).

Réponse : États: CREATED→ISSUED→(ACTIVATED)→(SUSPENDED)→(BLOCKED) ; ISSUED/ACTIVATED→EXPIRED. Transitions: activate(), suspend(reason), resume(), block(fraud), expire(date). Garder invariants: BLOCKED terminal (sauf reissue).

Q4. Concevoir 'Système d'alertes anti-fraude' : diagramme de composants + interfaces, incluant event bus.

Réponse : Composants: API Gateway, Core Banking Service, Transfer Service, Fraud Service, Notification Service, Reporting, Event Bus (Kafka/Rabbit). Interfaces: publish(TransactionPosted), consume, rule engine. TechLead: ajouter observabilité, DLQ, retries, circuit breaker.

Q5. Concevoir 'Relevé de compte' : diagramme de séquence pour la génération (période, pagination, export PDF).

Réponse : Séquence: Client→StatementAPI: request(period) → StatementService: validateAuth, fetch ledger entries (cursor pagination), compute running balance, generate PDF via Template/Renderer, store in Object Storage, return download link. Ajouter caching et async job si gros volume.

Q6. Concevoir le système de 'Virement Interne' : produire un diagramme de classes (domain) + un diagramme de séquence pour le scénario 'virement validé'.

Réponse : Classes attendues: Customer, Account, Transfer, LedgerEntry, Money (Value Object), TransferService, FraudCheckService (option), NotificationService (option). Relations: Customer 1..* Account; Transfer relie from_account/to_account; LedgerEntry 1..* par Transfer. Séquence: Client→API→TransferService: validate(idempotencyKey), lock accounts, check balance, persist transfer+ledger, commit, publish event.

Q7. Concevoir 'Ouverture de compte' : diagramme d'activités (workflow) incluant KYC/AML et validation de documents.

Réponse : Activités: Saisie demande → Vérif identité (KYC) → Screening AML/PEP → Vérif documents → Décision (accept/reject) → Création Customer/Account → Notification. Ajouter branches d'erreur, timers (documents manquants), et swimlanes (Client, Agent, Services).

Q8. Concevoir 'Carte bancaire' : diagramme d'états pour Card (émise, activée, suspendue, bloquée, expirée).

Réponse : États: CREATED→ISSUED→(ACTIVATED)→(SUSPENDED)→(BLOCKED) ; ISSUED/ACTIVATED→EXPIRED. Transitions: activate(), suspend(reason), resume(), block(fraud), expire(date). Garder invariants: BLOCKED terminal (sauf reissue).

Q9. Concevoir 'Système d'alertes anti-fraude' : diagramme de composants + interfaces, incluant event bus.

Réponse : Composants: API Gateway, Core Banking Service, Transfer Service, Fraud Service, Notification Service, Reporting, Event Bus (Kafka/Rabbit). Interfaces: publish(TransactionPosted), consume, rule engine. TechLead: ajouter observabilité, DLQ, retries, circuit breaker.

Q10. Concevoir 'Relevé de compte' : diagramme de séquence pour la génération (période, pagination, export PDF).

Réponse : Séquence: Client→StatementAPI: request(period) → StatementService: validateAuth, fetch ledger entries (cursor pagination), compute running balance, generate PDF via Template/Renderer, store in Object Storage, return download link. Ajouter caching et async job si gros volume.

Q11. Concevoir le système de 'Virement Interne' : produire un diagramme de classes (domain) + un diagramme de séquence pour le scénario 'virement validé'.

Réponse : Classes attendues: Customer, Account, Transfer, LedgerEntry, Money (Value Object), TransferService, FraudCheckService (option), NotificationService (option). Relations: Customer 1..* Account; Transfer relie from_account/to_account; LedgerEntry 1..* par Transfer. Séquence: Client→API→TransferService: validate(idempotencyKey), lock accounts, check balance, persist transfer+ledger, commit, publish event.

Q12. Concevoir 'Ouverture de compte' : diagramme d'activités (workflow) incluant KYC/AML et validation de documents.

Réponse : Activités: Saisie demande → Vérif identité (KYC) → Screening AML/PEP → Vérif documents → Décision (accept/reject) → Création Customer/Account → Notification. Ajouter branches d'erreur, timers (documents manquants), et swimlanes (Client, Agent, Services).

Q13. Concevoir 'Carte bancaire' : diagramme d'états pour Card (émise, activée, suspendue, bloquée, expirée).

Réponse : États: CREATED→ISSUED→(ACTIVATED)→(SUSPENDED)→(BLOCKED) ; ISSUED/ACTIVATED→EXPIRED. Transitions: activate(), suspend(reason), resume(), block(fraud), expire(date). Garder invariants: BLOCKED terminal (sauf reissue).

Q14. Concevoir 'Système d'alertes anti-fraude' : diagramme de composants + interfaces, incluant event bus.

Réponse : Composants: API Gateway, Core Banking Service, Transfer Service, Fraud Service, Notification Service, Reporting, Event Bus (Kafka/Rabbit). Interfaces: publish(TransactionPosted), consume, rule engine. TechLead: ajouter observabilité, DLQ, retries, circuit breaker.

Q15. Concevoir 'Relevé de compte' : diagramme de séquence pour la génération (période, pagination, export PDF).

Réponse : Séquence: Client→StatementAPI: request(period) → StatementService: validateAuth, fetch ledger entries (cursor pagination), compute running balance, generate PDF via Template/Renderer, store in Object Storage, return download link. Ajouter caching et async job si gros volume.

Q16. Concevoir le système de 'Virement Interne' : produire un diagramme de classes (domain) + un diagramme de séquence pour le scénario 'virement validé'.

Réponse : Classes attendues: Customer, Account, Transfer, LedgerEntry, Money (Value Object), TransferService, FraudCheckService (option), NotificationService (option). Relations: Customer 1..* Account; Transfer relie from_account/to_account; LedgerEntry 1..* par Transfer. Séquence: Client→API→TransferService: validate(idempotencyKey), lock accounts, check balance, persist transfer+ledger, commit, publish event.

Q17. Concevoir 'Ouverture de compte' : diagramme d’activités (workflow) incluant KYC/AML et validation de documents.

Réponse : Activités: Saisie demande → Vérif identité (KYC) → Screening AML/PEP → Vérif documents → Décision (accept/reject) → Création Customer/Account → Notification. Ajouter branches d’erreur, timers (documents manquants), et swimlanes (Client, Agent, Services).

Q18. Concevoir 'Carte bancaire' : diagramme d’états pour Card (émise, activée, suspendue, bloquée, expirée).

Réponse : États: CREATED→ISSUED→(ACTIVATED)→(SUSPENDED)→(BLOCKED) ; ISSUED/ACTIVATED→EXPIRED. Transitions: activate(), suspend(reason), resume(), block(fraud), expire(date). Garder invariants: BLOCKED terminal (sauf reissue).

Q19. Concevoir 'Système d’alertes anti-fraude' : diagramme de composants + interfaces, incluant event bus.

Réponse : Composants: API Gateway, Core Banking Service, Transfer Service, Fraud Service, Notification Service, Reporting, Event Bus (Kafka/Rabbit). Interfaces: publish(TransactionPosted), consume, rule engine. TechLead: ajouter observabilité, DLQ, retries, circuit breaker.

Q20. Concevoir 'Relevé de compte' : diagramme de séquence pour la génération (période, pagination, export PDF).

Réponse : Séquence: Client→StatementAPI: request(period) → StatementService: validateAuth, fetch ledger entries (cursor pagination), compute running balance, generate PDF via Template/Renderer, store in Object Storage, return download link. Ajouter caching et async job si gros volume.

Q21. Concevoir le système de 'Virement Interne' : produire un diagramme de classes (domain) + un diagramme de séquence pour le scénario 'virement validé'.

Réponse : Classes attendues: Customer, Account, Transfer, LedgerEntry, Money (Value Object), TransferService, FraudCheckService (option), NotificationService (option). Relations: Customer 1..* Account; Transfer relie from_account/to_account; LedgerEntry 1..* par Transfer. Séquence: Client→API→TransferService: validate(idempotencyKey), lock accounts, check balance, persist transfer+ledger, commit, publish event.

Q22. Concevoir 'Ouverture de compte' : diagramme d’activités (workflow) incluant KYC/AML et validation de documents.

Réponse : Activités: Saisie demande → Vérif identité (KYC) → Screening AML/PEP → Vérif documents → Décision (accept/reject) → Création Customer/Account → Notification. Ajouter branches d’erreur, timers (documents manquants), et swimlanes (Client, Agent, Services).

Q23. Concevoir 'Carte bancaire' : diagramme d’états pour Card (émise, activée, suspendue, bloquée, expirée).

Réponse : États: CREATED→ISSUED→(ACTIVATED)→(SUSPENDED)→(BLOCKED) ; ISSUED/ACTIVATED→EXPIRED. Transitions: activate(), suspend(reason), resume(), block(fraud), expire(date). Garder invariants: BLOCKED terminal (sauf reissue).

Q24. Concevoir 'Système d’alertes anti-fraude' : diagramme de composants + interfaces, incluant event bus.

Réponse : Composants: API Gateway, Core Banking Service, Transfer Service, Fraud Service, Notification Service, Reporting, Event Bus (Kafka/Rabbit). Interfaces: publish(TransactionPosted), consume, rule engine. TechLead: ajouter observabilité, DLQ, retries, circuit breaker.

Q25. Concevoir 'Relevé de compte' : diagramme de séquence pour la génération (période, pagination, export PDF).

Réponse : Séquence: Client→StatementAPI: request(period) → StatementService: validateAuth, fetch ledger entries (cursor pagination), compute running balance, generate PDF via Template/Renderer, store in Object Storage, return download link. Ajouter caching et async job si gros volume.