■ Fiche de Révision – Préparation Technique Java & Architecture

Fondamentaux de Java

■ Concepts de base

- **Orienté Objet** : Paradigme basé sur objets.

Piliers OOP : Encapsulation, Abstraction, Héritage, Polymorphisme.

- **Classe vs Objet** : Classe = plan/blueprint ; Objet = instance concrète.

- **Interface** : Contrat définissant des méthodes abstraites.

Différence avec classe abstraite : Interface = uniquement comportements ; Classe abstraite = peut contenir attributs + implémentations partielles.

- **Héritage** : Transmission des propriétés d'une classe mère à une classe fille. Java supporte **simple héritage** (pas multiple pour les classes).

- **Polymorphisme** :

- **Surcharge (overloading)** : même nom de méthode, signatures différentes.

- **Redéfinition (overriding)** : redéfinir une méthode héritée.

■ Structures de données

- **Principales** : Tableaux, ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap.

- **Complexités** (moyenne) :

- ArrayList : accès O(1), insertion fin O(1), insertion milieu O(n).

- LinkedList : accès O(n), insertion début/fin O(1).

- HashSet/HashMap : ajout/recherche O(1).

- TreeSet/TreeMap : ajout/recherche O(log n).

■ Collections

- **Interfaces** : List, Set, Queue, Map.

- **Quand utiliser** :

- ArrayList → accès rapide, insertions rares.

- LinkedList → insertions/suppressions fréquentes.

- HashSet → ensemble unique sans ordre.

- HashMap → paires clé-valeur.

■ Exceptions

- **Exception** : événement perturbant le flot normal du programme.

- **Types** : Checked (IOException), Unchecked (NullPointerException), Errors (OutOfMemoryError).

- **Gestion** : `try-catch-finally`, `throws`, `throw`.

■ Concurrency

- **Thread** : unité d'exécution parallèle.

- **Création** : `extends Thread`, `implements Runnable`, `Executors`.

- **Synchronisation** : mot-clé `synchronized`, `Lock`, `Semaphore`, `volatile`.

■ JVM

- **JVM** : Machine virtuelle exécutant le bytecode Java.

- **Cycle de vie** : Compilation (Java → bytecode) → ClassLoader → Execution Engine.

- **Garbage Collector** : libère mémoire automatiquement en supprimant objets non référencés.

Programmation Avancée

■ Design Patterns

- **Principaux** : Singleton (instance unique), Factory (création d'objets), Observer (notification), Strategy (choix d'algorithme).

- **Exemple** : Singleton pour config système partagée.

■ Algorithmes

- **Tri** : QuickSort O(n log n), MergeSort O(n log n), BubbleSort O(n^2).

- **Recherche** : Linéaire O(n), Dichotomique O(log n).

■ SQL & Bases de données

- **Requêtes** : SELECT, JOIN, WHERE, GROUP BY.

- **Optimisation** : Index, requêtes préparées.

- **JDBC** : API Java pour connecter BD relationnelles.

■ Frameworks & Outils

- **Spring** (DI, REST, sécurité), **Hibernate** (ORM), **Maven/Gradle** (build & dépendances).

API & Web Services

- **REST vs SOAP** : REST → léger, JSON, HTTP. SOAP → protocole strict basé sur XML.

- **JSON vs XML** : JSON léger et lisible, XML plus verbeux mais extensible.

- **OAuth 2.0** : protocole d'authentification (Authorization Code, Client Credentials, Implicit, Password).

- **API Gateway** : point d'entrée unique, sécurité, load balancing.

- **Swagger/OpenAPI** : documenter, tester et générer clients API.

Conception & Architecture

- **SOLID** :

- S : Single Responsibility

- O : Open/Closed

- L : Liskov Substitution

- I : Interface Segregation

- D : Dependency Inversion

- **Microservices** : Avantages → scalabilité, indépendance. Inconvénients → complexité, cohérence des données.

- **TDD** : écrire tests avant le code → fiabilité, documentation vivante.

Microservices

- **Communication** : REST, gRPC, Messaging (Kafka, RabbitMQ).

- **Découverte de services** : Eureka, Consul, Kubernetes.

- **Scaling** : Horizontal (plus d'instances), Vertical (plus de ressources).

- **Résilience** : Retry, Circuit Breaker, Load balancing.

Veille Technologique

- **Nouveautés Java** : `var`, records, switch expressions, virtual threads (Java 21).

- **Frameworks** : Spring Boot, Quarkus, Micronaut.

- **Cloud** : AWS, Azure, GCP → élasticité, coût réduit, scalabilité.

- **DevOps** : CI/CD, Docker, Kubernetes, monitoring.

Questions Générales

- **Optimisation** : algorithmes efficaces, caches, pooling, profilage.

- **Gestion des erreurs** : exceptions contrôlées, logs, monitoring.

- **Threads & parallélisme** : exécution concurrente, `ExecutorService`, synchronisation.

- **Algorithmes & Structures maîtrisées** : tri, recherche, piles, files, graphes.