Règles appliques pour passer de C a Java

1. Structures de données

En C

- **Structure** : Utilisation de `struct Supermat` pour représenter une matrice, avec des champs pour le nombre de lignes, de colonnes et un pointeur vers un tableau 2D dynamique.
- **Allocation** : Utilisation de `malloc` / `free` pour gérer la mémoire.

En Java

- **Classe** : Utilisation d'une classe `Supermat` avec des attributs `nl` (nombre de lignes), `nc` (nombre de colonnes) et `lignes` (tableau 2D de `double`).
- **Allocation** : Utilisation de `new` pour créer des objets et des tableaux. Pas besoin de `free`, le ramasse-miettes s'en occupe.

Règle appliquée :
Chaque `struct` C devient une `class` Java. Les champs sont traduits en attributs publics ou privés selon le besoin.

---

2. Fonctions utilitaires

En C

- Fonctions du type `Supermat* allouerSupermat(int nl, int nc)` pour créer une matrice.
- Fonctions pour le produit matriciel, la permutation de lignes, l'extraction de sous-matrice, etc.
- Passage des pointeurs pour modifier les matrices en place.

En Java

- Méthodes statiques dans une classe utilitaire `SupermatUtils`.
- Retour d'objets `Supermat` ou modification directe des objets passés en paramètre (car tout est référence).
- Pas de pointeurs, mais passage par référence des objets.

Règle appliquée :
Les fonctions C deviennent des méthodes statiques Java, regroupées dans une classe utilitaire. Les pointeurs sont remplacés par des références d'objets.

---

3. Gestion de la mémoire

En C

- Nécessité d'allouer et de libérer explicitement la mémoire (`malloc`, `free`).
- Risque de fuites mémoire.

En Java

- Allocation via `new`.
- Libération automatique par le garbage collector.
- Ajout d'une méthode `rendreSupermat` qui met le tableau à `null` pour simuler la libération (optionnel en Java).

Règle appliquée :
Suppression de tout code de libération mémoire explicite, sauf pour simuler le comportement C.

---

4. Tableaux et accès mémoire

En C

- Tableaux dynamiques via pointeurs.
- Accès via `mat->lignes[i][j]`.

En Java

- Tableaux 2D natifs (`double[][]`).
- Accès via `mat.lignes[i][j]`.

Règle appliquée :
Traduction directe des accès mémoire, mais sans gestion manuelle des pointeurs.

---

5. Gestion des erreurs

En C

- Retour de `NULL` ou de codes d'erreur (`-1` , etc.) en cas de problème.

En Java

- Retour de `null` pour les objets, ou de codes d'erreur pour les méthodes retournant un entier.

Règle appliquée :
Même logique de gestion d'erreur, mais adaptée à la gestion des objets Java.

---

6. Programmation orientée objet

En C

- Pas d'objets, tout est structuré autour de fonctions et de structures.

En Java

- Utilisation de classes pour encapsuler les données et les méthodes.
- Possibilité d'ajouter des méthodes d'instance (non utilisé ici, mais possible).

Règle appliquée :
Adoption du paradigme objet, même si la logique reste procédurale via des méthodes statiques.

---

7. Impression et affichage

En C

- Utilisation de `printf` pour afficher les matrices.

En Java

- Utilisation de `System.out.println` et d'une méthode utilitaire `printSupermat` .

---

8. Différences syntaxiques et idiomatiques

Aspect	C	Java
Déclaration	`struct Supermat *mat;`	`Supermat mat;`
Allocation	`mat = malloc(sizeof(Supermat));`	`mat = new Supermat(...);`
Libération	`free(mat);`	`mat = null;` (garbage collector)
Tableaux	`double **lignes;`	`double[][] lignes;`
Fonctions	`Supermat* produitSupermat(...)`	`Supermat produitSupermat(...)`
Passage de paramètres	Par valeur (pointeurs pour modifier)	Par référence (objets)
Gestion erreurs	`NULL`, codes d'erreur	`null`, codes d'erreur

---

9. Règles de migration principales

1. **Struct → Classe** : Chaque structure devient une classe Java.
2. **Fonctions → Méthodes statiques** : Les fonctions C deviennent des méthodes statiques dans une classe utilitaire.
3. **Pointeurs → Références** : Les pointeurs sont remplacés par des références d'objets.
4. **Allocation/libération mémoire** : Utilisation de `new` et suppression de `free`.
5. **Gestion des erreurs** : Conservation de la logique de retour d'erreur, adaptée à Java.
6. **Affichage** : Utilisation de `System.out.println` au lieu de `printf`.
7. **Encapsulation** : Possibilité d'encapsuler davantage les données (non obligatoire mais recommandé en Java).

---

10. Conclusion

La migration du projet SUPERMAT de C vers Java a permis de :

- Simplifier la gestion mémoire.
- Bénéficier de la programmation orientée objet.
- Améliorer la lisibilité et la sécurité du code.
- Réduire les risques de bugs liés aux pointeurs et à la mémoire.

Les règles de conversion ont été appliquées de façon systématique pour garantir une correspondance fonctionnelle entre les deux versions, tout en profitant des avantages du langage Java.