Formation

JAVA

m2iformation.fr

Clean Code en Java

Principes Clés

- Écriture de code lisible et maintenable
- Réduction de la complexité
- Respect des bonnes pratiques de conception

Nommage Clair

- Utiliser des noms explicites pour les variables, méthodes et classes
- Éviter les abréviations obscures
- Exemples :

```
// Mauvais
int x = 10;

// Bon
int nombreDeClients = 10;
```

Méthodes Simples et Courtes

- Une seule responsabilité par méthode
- Ne pas dépasser 20-30 lignes
- Exemples :

```
// Mauvais
void process() {
    // beaucoup de logique imbriquée
}

// Bon
void validerUtilisateur() {}
void enregistrerUtilisateur() {}
```

KISS (Keep It Simple, Stupid)

- Garder le code aussi simple que possible
- Éviter les complexités inutiles
- Exemples :

```
// Mauvais
public int calculerSomme(int a, int b) {
    return a + b;
}

// Bon
public int somme(int a, int b) {
    return a + b;
}
```

DRY (Don't Repeat Yourself)

- Éviter la duplication de code
- Utiliser des méthodes et classes réutilisables
- Exemples :

```java
// Mauvais
public double calculerPrixTTC(double prix) {
    return prix * 1.2;
}

public double calculerPrixRemise(double prix) {
    return (prix * 0.9) * 1.2;
}

// Bon
public double appliquerTVA(double prix) {
    return prix * 1.2;
}

public double appliquerRemise(double prix) {
    return appliquerTVA(prix * 0.9);
}
```

Éviter les Commentaires Inutiles

- Le code doit être auto-descriptif
- Utiliser des commentaires uniquement si nécessaire

```
// Mauvais
// Vérifie si l'utilisateur est actif
if (user.isActive()) {}

// Bon
if (user.estActif()) {}
```

Gestion des Exceptions

- Éviter les `try-catch` excessifs
- Utiliser des exceptions spécifiques

```
// Mauvais
try {
    utilisateur.save();
} catch (Exception e) {
    System.out.println("Erreur");
}

// Bon
try {
    utilisateur.save();
} catch (DatabaseException e) {
```

Tests Unitaires

- Écrire des tests clairs et isolés
- Utiliser des frameworks comme JUnit

```
@Test
void shouldReturnTrueWhenUserIsActive() {
    User user = new User(true);
    assertTrue(user.isActive());
}
```

Structuration des Projets Spring

Architecture et structuration modulaire

- Séparer les responsabilités en modules
- Adopter une approche hexagonale ou en couches
- Exemples :
  - `com.exemple.controller`
  - `com.exemple.service`
  - `com.exemple.repository`

Organisation des Couches

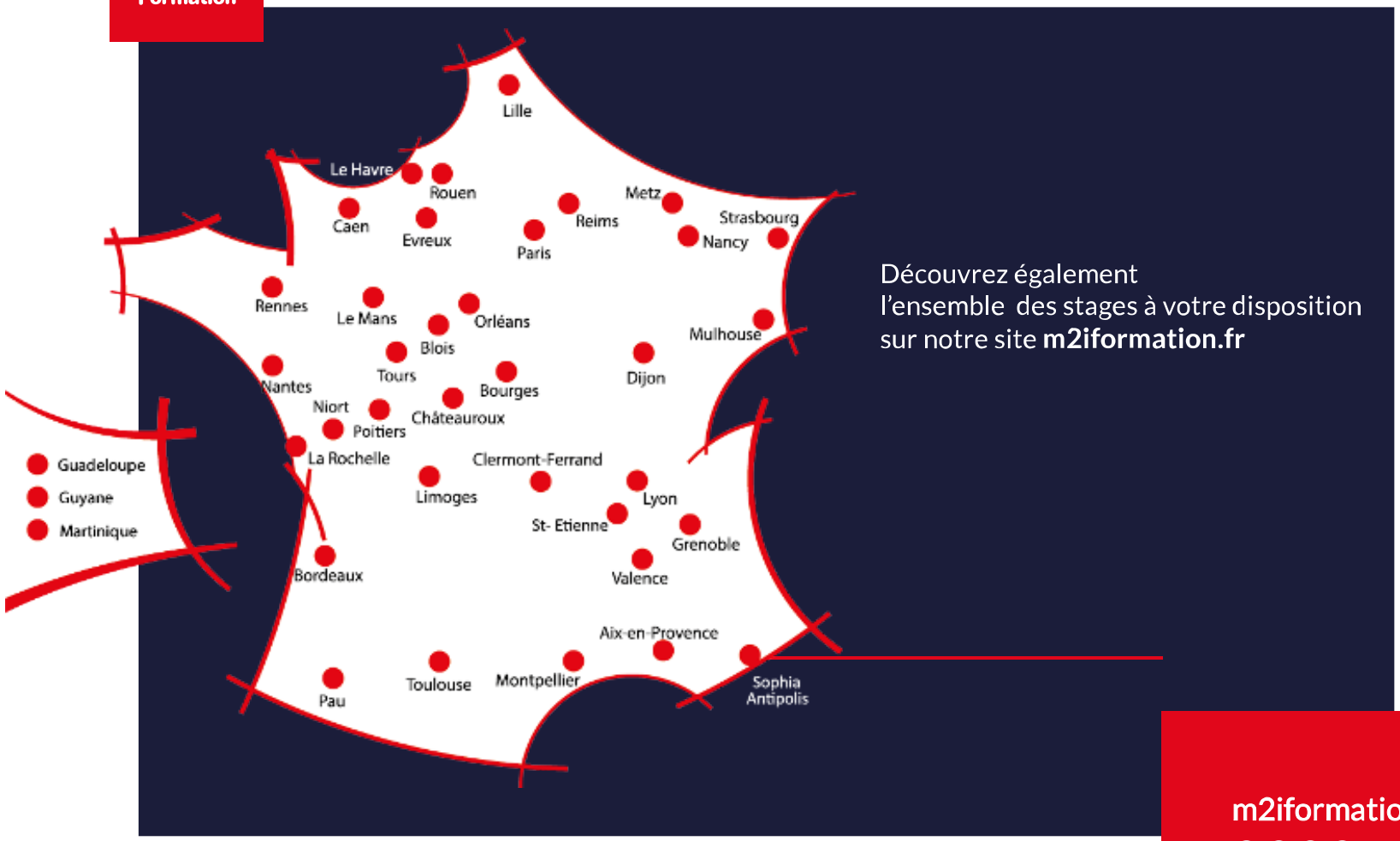
- **Controller** : Gère les requêtes HTTP
- **Service** : Contient la logique métier
- **Repository** : Interagit avec la base de données

```java
@RestController
@RequestMapping("/users")
public class UserController {
    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping("/{id}")
    public ResponseEntity<UserDTO> getUser(@PathVariable Long id) {
        return ResponseEntity.ok(userService.getUserById(id));
    }
}
```

Intégration des DTO 📦

- Éviter d'exposer directement les entités
- Protéger les données sensibles et optimiser les réponses API

```
public class UserDTO {
    private String name;
    private String email;

    public UserDTO(User user) {
        this.name = user.getName();
        this.email = user.getEmail();
    }
}
```



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr