Guide Complet de la Programmation Orientée Objet en Dart

1. Classes et Objets

1.1 Définition d'une classe

dart
Copy
```dart
class Personne {
  // Propriétés (attributs)
  String nom;
  int age;

  // Constructeur
  Personne(this.nom, this.age);

  // Méthode
  void sePresenter() {
    print('Je m\'appelle $nom et j\'ai $age ans');
  }
}
```

1.2 Création d'objets

dart
Copy
```dart
void main() {
  var personne1 = Personne('Alice', 25);
  personne1.sePresenter();
}
```

2. Constructeurs

2.1 Constructeur par défaut

dart
Copy
```dart
class Point {
  double x = 0;
  double y = 0;

  Point(); // Constructeur par défaut
```

```
}
```

2.2 Constructeur nommé

```
dart
Copy
class Rectangle {
  double largeur;
  double longueur;

  Rectangle(this.largeur, this.longueur);

  // Constructeur nommé
  Rectangle.carre(double cote) :
      largeur = cote,
      longueur = cote;
}
```

2.3 Constructeur avec initialiseurs

```
dart
Copy
class Cercle {
  final double rayon;
  final double diametre;

  Cercle(this.rayon) : diametre = rayon * 2;
}
```

3. Encapsulation

3.1 Propriétés privées

```
dart
Copy
class CompteBancaire {
  // Propriété privée (commence par _)
  double _solde = 0;

  // Getter
  double get solde => _solde;

  // Setter
  set solde(double montant) {
```

```
    if (montant >= 0) {
      _solde = montant;
    }
  }
}
```

3.2 Méthodes privées

dart
Copy

```
class Calculator {
  double _result = 0;

  void addNumber(double num) {
    _result = _calculate(_result, num);
  }

  double _calculate(double a, double b) {
    return a + b;
  }
}
```

4. Héritage

4.1 Classe de base et classe dérivée

dart
Copy

```
class Animal {
  String nom;

  Animal(this.nom);

  void faireBruit() {
    print('L\'animal fait un bruit');
  }
}

class Chat extends Animal {
  Chat(String nom) : super(nom);

  @override
  void faireBruit() {
```

```
    print('Le chat miaule');
  }
}
```

4.2 Méthodes abstraites

```
dart
Copy
abstract class Forme {
  double calculerAire();
  double calculerPerimetre();
}

class Carre extends Forme {
  double cote;

  Carre(this.cote);

  @override
  double calculerAire() => cote * cote;

  @override
  double calculerPerimetre() => 4 * cote;
}
```

5. Interfaces et Mixins

5.1 Interfaces

```
dart
Copy
abstract class Volant {
  void voler();
}

class Oiseau implements Volant {
  @override
  void voler() {
    print('L\'oiseau vole avec ses ailes');
  }
}
```

5.2 Mixins

dart
Copy
```dart
mixin Nageur {
  void nager() {
    print('Je nage dans l\'eau');
  }
}

class Poisson with Nageur {
  String nom;

  Poisson(this.nom);
}
```

6. Génériques

6.1 Classes génériques

dart
Copy
```dart
class Pile<T> {
  List<T> _elements = [];

  void empiler(T element) {
    _elements.add(element);
  }

  T? depiler() {
    if (_elements.isEmpty) return null;
    return _elements.removeLast();
  }
}
```

6.2 Méthodes génériques

dart
Copy
```dart
T trouverMax<T extends Comparable>(List<T> liste) {
  if (liste.isEmpty) throw Exception('Liste vide');

  T max = liste[0];
  for (var element in liste.skip(1)) {
    if (element.compareTo(max) > 0) {
```

```
            max = element;
        }
    }
    return max;
}
```

7. Bonnes Pratiques

7.1 Principes SOLID

- **S**ingle Responsibility Principle
- **O**pen/Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

7.2 Exemple d'application des principes SOLID

dart
Copy

```dart
// Interface segregation principle
abstract class Imprimable {
  void imprimer();
}

abstract class Scannable {
  void scanner();
}

// Single responsibility principle
class GestionnaireFichiers {
  void sauvegarder(String contenu, String chemin) {
    // Logique de sauvegarde
  }
}

// Open/closed principle
abstract class Forme {
  double calculerAire();
}

class Rectangle implements Forme {
  double largeur;
  double longueur;
```

```
  Rectangle(this.largeur, this.longueur);

  @override
  double calculerAire() => largeur * longueur;
}
```

8. Gestion des Exceptions

dart
Copy

```
class SoldeInsuffisantException implements Exception {
  String message;
  SoldeInsuffisantException(this.message);
}

class CompteBancaire {
  double _solde = 0;

  void retirer(double montant) {
    if (montant > _solde) {
      throw SoldeInsuffisantException('Solde insuffisant');
    }
    _solde -= montant;
  }
}

void main() {
  try {
    var compte = CompteBancaire();
    compte.retirer(100);
  } on SoldeInsuffisantException catch (e) {
    print('Erreur: ${e.message}');
  } finally {
    print('Opération terminée');
  }
}
```