Les getters et setters en Dart

1. Objectifs du cours

- Comprendre le concept de getters et de setters en Dart.
- Savoir utiliser des getters pour récupérer des valeurs d'une classe.
- Savoir utiliser des setters pour modifier les valeurs d'une classe.
- Comprendre l'importance des getters et des setters dans l'encapsulation des données.
- Être en mesure d'appliquer les getters et les setters dans des scénarios pratiques.

2. Comprendre le concept de getters et de setters en Dart.

1. Présentation

- En POO, les **accesseurs** désignent des méthodes qui permettent d'accéder aux attributs privés d'une classe en lecture et écriture. Il y en a de deux types, les getters et les setters.
- Les getters et les setters sont également appelés accesseurs et les mutateurs permettent au programme d'initialiser et de récupérer les valeurs des champs de classe respectivement.
- Les getters ou les accesseurs sont définis à l'aide du mot-clé get, les setters ou les mutateurs sont définis à l'aide du mot-clé set.
- Obtenir et **Setter** fournir un accès explicite en lecture et en écriture aux propriétés d'un objet. Dans Dart, **get** et **set** sont les mots-clés utilisés pour créer un **getter** et un **setter**. **Getter** lit la valeur de la propriété et agit en tant qu'accesseur . **Setter** met à jour la valeur de la propriété et agit en tant que mutateur .

> ***Les getters sont des méthodes spéciales dans une classe qui permettent de récupérer la valeur d'un attribut privé depuis l'extérieur de la classe. Ils offrent un moyen sûr et contrôlé d'accéder à ces attributs privés sans exposer directement leur valeur.***

2. Définitions

- Les **setters** et les **getters** sont des composants essentiels de la programmation orientée objet, permettant aux développeurs de contrôler l'accès aux propriétés de classe et de fournir une logique personnalisée pour lire et modifier les valeurs.

apcpedagogie.com

- Dans Dart et Flutter, les **setters** et **getters** fournissent un mécanisme puissant pour encapsuler les données et ajouter des fonctionnalités supplémentaires.

1. **Getter**

    - **Syntax:**

      ```
      return_type get field_name{ // return the value }
      ```

    - Une méthode qui permet d'accéder à la valeur d'une propriété d'une classe.
    - Un getter est une méthode qui permet d'obtenir la valeur d'une propriété.
    - Pour définir un getter, utilisez le mot-clé get suivi du nom du getter.

      Solution

      ```
      class MaClasse {
        String _maPropriete = "Valeur initiale";

        String get maPropriete {
          return _maPropriete;
        }
      }
      ```

2. **Setter**

    - **Syntax:**

      ```
      set field_name { // set the value }
      ```

    - **Setter** : Une méthode qui permet de mettre à jour la valeur d'une propriété d'une classe.
    - Il est utilisé pour définir les données dans une variable reçue de la méthode **getter**. Toutes les classes ont une méthode de définition par défaut mais elle peut être remplacée explicitement.
    - La méthode setter peut être définie à l'aide du mot-clé set comme :

```
définir nom_champ{
    ...
}
```

3. **Encapsulation**

    - **encapsulation**: Le principe de regrouper les données (propriétés) et les méthodes (fonctions) qui les manipulent dans une seule unité, c'est-à-dire

apcpedagogie.com

une classe.

- Les **setters** et **getters** assurent l'encapsulation en masquant la représentation interne des données et en autorisant un accès contrôlé. Cela favorise la dissimulation des informations et améliore la maintenabilité du code.

3. **Exemple**

- Imaginons un Compte Bancaire et un Distributeur de billets.
- **La Classe CompteBancaire (Le coffre-fort)**
- Ici, le `setter` sert de sécurité : on ne peut pas retirer plus d'argent qu'on n'en a, et on ne peut pas déposer un montant négatif (ce qui serait un vol !).
  - **`compte_bancaire.dart`**
    - Ce fichier contient la logique métier et la sécurité des données.

```
// Fichier : compte_bancaire.dart
class CompteBancaire {
  final String titulaire;
  double _solde; // Privé

  CompteBancaire(this.titulaire, this._solde);

  // Accesseur (Lecture seule)
  double get solde => _solde;

  // Mutateur pour le dépôt
  set depot(double montant) {
    if (montant > 0) {
      _solde += montant;
      print("Dépôt de $montant€ réussi.");
    }
  }

  // Mutateur pour le retrait
  set retrait(double montant) {
    if (montant > 0 && montant <= _solde) {
      _solde -= montant;
      print("Retrait de $montant€ effectué.");
    } else {
      print("Opération refusée : Solde insuffisant.");
    }
```

apcpedagogie.com

```
  }
}
```

- **utilisateur.dart**
  - Ce fichier gère l'interaction. Note l'utilisation de import pour pouvoir utiliser la classe CompteBancaire.

```
// Fichier : utilisateur.dart
import 'compte_bancaire.dart';

class Utilisateur {
  String nom;
  CompteBancaire monCompte;

  Utilisateur(this.nom, this.monCompte);

  void consulterEtDepenser(double achat) {
    print("--- Session de ${nom} ---");
    print("Solde actuel : ${monCompte.solde}€");
    monCompte.retrait = achat;
    print("Nouveau solde : ${monCompte.solde}€\n");
  }
}
```

- **main.dart**
  - C'est le point d'entrée de ton application qui orchestre le tout.

```
// Fichier : main.dart
import 'compte_bancaire.dart';
import 'utilisateur.dart';

void main() {
  // Création des objets
  var compteDeJean = CompteBancaire("Jean Dupont", 150.0);
  var jean = Utilisateur("Jean", compteDeJean);

  // Actions
  jean.consulterEtDepenser(60.0);

  // Utilisation directe du setter de dépôt
  compteDeJean.depot = 20.0;
```

apcpedagogie.com

```
  print("Après dépôt, solde de Jean : ${compteDeJean.solde}€");
}
```

4. **Exercice : Gestion de Carburant**

- Imaginons un système de gestion pour le réservoir d'une voiture avec une sécurité antidébordement.
- **La Classe Reservoir (La gestion du carburant)**
- Ici, le `setter` vérifie que l'on ne dépasse pas la capacité maximale du réservoir et le `getter` permet de surveiller le niveau de réserve.
  - **reservoir.dart**
    - Ce fichier contient la logique de sécurité pour empêcher le réservoir de déborder.

```
// Fichier : reservoir.dart
class Reservoir {
final double capaciteMax;
double _niveauActuel; // Privé

Reservoir(this.capaciteMax, this._niveauActuel);

// Accesseur pour lire le niveau
double get niveau => _niveauActuel;

// Getter calculé : vérifie si on est sur la réserve (< 5L)
bool get estBas => _niveauActuel < 5;

// Mutateur pour le remplissage avec sécurité
set remplissage(double quantite) {
if (_niveauActuel + quantite > capaciteMax) {
_niveauActuel = capaciteMax;
print("Plein effectué ! Le surplus a débordé.");
} else {
_niveauActuel += quantite;
print("Ajout de $quantite litres réussi.");
}
}

// Mutateur pour la consommation
set consommation(double litres) {
```

apcpedagogie.com

```
if (_niveauActuel - litres >= 0) {
_niveauActuel -= litres;
} else {
_niveauActuel = 0;
print("Panne sèche !");
}
}
}
```

- **voiture.dart**
  - Ce fichier gère l'interaction entre le conducteur et son réservoir.

```
// Fichier : voiture.dart
import 'reservoir.dart';

class Voiture {
String marque;
Reservoir reservoir;

Voiture(this.marque, this.reservoir);

void rouler(double distance) {
double essenceConsommee = distance * 0.06; // 6L/100km
reservoir.consommation = essenceConsommee;

print("La $marque a roulé sur $distance km.");
if (reservoir.estBas) {
  print("Alerte : Niveau de carburant critique
(${reservoir.niveau}L) !");
}
}
}
```

- **main.dart**
  - Ce fichier simule l'utilisation réelle du véhicule et du passage à la pompe.

```
// Fichier : main.dart
import 'reservoir.dart';
import 'voiture.dart';
```

apcpedagogie.com

```
void main() {
// Création d'un réservoir de 50L, actuellement à 10L
var monReservoir = Reservoir(50.0, 10.0);
var maAuto = Voiture("Peugeot", monReservoir);

// Test du setter : Tentative de mettre 60L (trop pour un
réservoir de 50L)
maAuto.reservoir.remplissage = 60.0;

// Test du getter de réserve : On roule beaucoup
maAuto.rouler(800.0);
}
```

5. **Activités**

1. **Activité: 01**

- Considérez la classe Dart suivante :

```
class Personne {
  String name;
  String _prenom;
  int age;

  Personne(this.name, this._prenom, this.age);

  void afficher_nom() {
    print("Je m'appelle: $name, mon prénom est $_prenom, j'ai $age
ans");
  }
}
```

- Ajoutez les méthodes **getter** et **setter** pour les variables d'instance **name, _prenom et age** de la classe **Personne**.
- Les méthodes **getter** doivent retourner la valeur respective de chaque variable et les méthodes **setter** doivent modifier la valeur de chaque variable.
- Vérifiez que les méthodes **getter** et **setter** fonctionnent correctement en effectuant les étapes suivantes :
  - Créez une instance de la classe **Personne**.

apcpedagogie.com

- Utilisez les méthodes **setter** pour modifier les valeurs des variables d'instance.
- Utilisez les méthodes **getter** pour accéder aux valeurs des variables d'instance.
- Appelez la méthode **afficher_nom()** pour afficher le **nom**, le **prénom** et l'**age**.

- Voici un exemple de code pour vous aider à démarrer :

```
void main() {
  Personne personne = Personne("Doe", "John", 30);

  // Utiliser les méthodes setter
  personne.name = "Smith";
  personne._prenom = "Jane";
  personne.age = 35;

  // Utiliser les méthodes getter
  String nom = personne.name;
  String prenom = personne._prenom;
  int age = personne.age;

  // Afficher les informations
  personne.afficher_nom();
  print("Nom : $nom");
  print("Prénom : $prenom");
  print("Âge : $age");
}
```

- Vous devez compléter la classe **Personne** en ajoutant les méthodes **getter** et **setter** appropriées pour chaque variable d'instance. Assurez-vous de comprendre comment fonctionnent les méthodes **getter** et setter en Dart. Solution

```
class Personne {
 // Variable d'instance pour le nom, précédée d'un underscore pour
indiquer qu'elle est privée
  String _name;
// Variable d'instance pour le prénom, précédée d'un underscore
pour indiquer qu'elle est privée
  String _prenom;
// Variable d'instance pour l'âge, précédée d'un underscore pour
```

apcpedagogie.com

```
indiquer qu'elle est privée
  int _age;

  // Constructeur de la classe Personne prenant en paramètres le
nom, le prénom et l'âge
  Personne(this._name, this._prenom, this._age);

  // Méthode getter pour récupérer la valeur du nom
  String get name => _name;
  // Méthode setter pour définir la valeur du nom
  set name(String value) => _name = value;

  // Méthode getter pour récupérer la valeur du prénom
  String get prenom => _prenom;
  // Méthode setter pour définir la valeur du prénom
  set prenom(String value) => _prenom = value;

  // Méthode getter pour récupérer la valeur de l'âge
  int get age => _age;
  // Méthode setter pour définir la valeur de l'âge
  set age(int value) => _age = value;

  // Méthode pour afficher le nom, le prénom et l'âge de la
personne
  void afficher_nom() {
    print("Je m'appelle: $_name, mon prénom est $_prenom, j'ai
$_age ans");
  }
}

void main() {
  // Création d'une instance de la classe Personne avec le nom
"Doe", le prénom "John" et l'âge 30
  Personne personne = Personne("Doe", "John", 30);

  // Utilisation des méthodes setter pour modifier les valeurs des
variables d'instance
  personne.name = "Smith";
  personne.prenom = "Jane";
  personne.age = 35;

  // Utilisation des méthodes getter pour récupérer les valeurs
des variables d'instance
```

apcpedagogie.com

```
  String nom = personne.name;
  String prenom = personne.prenom;
  int age = personne.age;

  // Affichage des informations en utilisant la méthode
afficher_nom() et les valeurs récupérées avec les méthodes getter
  personne.afficher_nom();
  print("Nom : $nom");
  print("Prénom : $prenom");
  print("Âge : $age");
}
```

2. **Activité: 02**

- Dans cet exercice, vous devez mettre en pratique vos connaissances sur les méthodes **getter** et **setter** en Dart en réalisant un ensemble de classes pour un **système de gestion d'une bibliothèque**.
- **Instructions :**
  - Définissez une classe Livre avec les propriétés suivantes :
    - **titre**: une chaîne de caractères représentant le titre du livre.
    - **auteur** : une chaîne de caractères représentant le nom de l'auteur du livre.
    - **anneePublication**: un entier représentant l'année de publication du livre.
  - Ajoutez les méthodes getter et setter pour chaque propriété de la classe Livre. Les méthodes getter doivent retourner la valeur respective de chaque propriété et les méthodes setter doivent modifier la valeur de chaque propriété.
  - Définissez une classe Bibliotheque avec les propriétés suivantes :
    - **nom**: une chaîne de caractères représentant le nom de la bibliothèque.
    - **livres**: une liste de livres (instances de la classe Livre) représentant les livres présents dans la bibliothèque.
  - Ajoutez les méthodes getter et setter pour la propriété nom de la classe Bibliotheque. Les méthodes getter doivent retourner la valeur de la propriété et les méthodes setter doivent modifier la valeur de la propriété.

apcpedagogie.com

- Ajoutez une méthode ajouterLivre à la classe Bibliotheque qui prend en paramètre un livre et l'ajoute à la liste des livres de la bibliothèque.
- Ajoutez une méthode afficherLivres à la classe Bibliotheque qui affiche les informations de tous les livres présents dans la bibliothèque (titre, auteur, année de publication).
- Votre objectif est de créer les classes Livre et Bibliotheque, d'implémenter les méthodes getter et setter appropriées, d'ajouter des livres à la bibliothèque et d'afficher les informations des livres.
    - Le fichier `livre.dart`

      livre.dart

```
class Livre {
  String _titre;
  String _auteur;
  int _anneePublication;

  Livre(this._titre, this._auteur,
this._anneePublication);

  String get titre => _titre;
  set titre(String value) => _titre = value;

  String get auteur => _auteur;
  set auteur(String value) => _auteur = value;

  int get anneePublication => _anneePublication;
  set anneePublication(int value) => _anneePublication
= value;
}
```

- Le fichier `bibliotheque.dart`

  bibliotheque.dart

```
import 'livre.dart';
class Bibliotheque {
  String _nom;
  List _livres = [];

  Bibliotheque(this._nom);
```

apcpedagogie.com

```
  String get nom => _nom;
  set nom(String value) => _nom = value;

  void ajouterLivre(Livre livre) {
    _livres.add(livre);
  }

  void afficherLivres() {
    print("Livres dans la bibliothèque $_nom :");
    for (var livre in _livres) {
      print("Titre: ${livre.titre}, Auteur:
${livre.auteur}, Année de publication:
${livre.anneePublication}");
    }
  }
}
```

- Le fichier `main.dart`

  main.dart

```
import 'bibliotheque.dart';
import 'livre.dart';
void main() {
  Bibliotheque biblio = Bibliotheque("Ma Biblio");

  Livre livre1 = Livre("Harry Potter", "J.K. Rowling",
1997);
  Livre livre2 = Livre("Le Seigneur des Anneaux",
"J.R.R. Tolkien", 1954);
  Livre livre3 = Livre("1984", "George Orwell", 1949);

  biblio.ajouterLivre(livre1);
  biblio.ajouterLivre(livre2);
  biblio.ajouterLivre(livre3);

  biblio.afficherLivres();
}
```

3. **Activité: 03**

   - Vous devez implémenter un système de gestion de compte bancaire en Dart en utilisant des méthodes **getter** et **setter** avancées.

apcpedagogie.com

- Définissez une classe **CompteBancaire** avec les propriétés suivantes :
  - **solde** : un double représentant le solde du compte.
  - **plafond**: un double représentant le plafond de découvert autorisé pour le compte.
- Ajoutez une méthode **getter** pour la propriété **solde**. Cette méthode doit afficher le solde actuel du compte et indiquer si le compte est à découvert.
- Ajoutez une méthode **setter** pour la propriété **solde**. Cette méthode doit vérifier si le nouveau solde dépasse le plafond de découvert autorisé. Si c'est le cas, elle doit refuser la transaction et afficher un message approprié.
- Ajoutez une méthode depot à la classe CompteBancaire qui prend en paramètre un montant et l'ajoute au solde du compte.
- Ajoutez une méthode retrait à la classe CompteBancaire qui prend en paramètre un montant et le retire du solde du compte. Assurez-vous que cette méthode respecte le plafond de découvert autorisé.
- Créez une instance de la classe **CompteBancaire** et effectuez une série de dépôts et de retraits pour tester votre implémentation. Assurez-vous que les méthodes **getter** et setter fonctionnent correctement et que les règles du plafond de découvert sont respectées.

Solution

```
class CompteBancaire {
  double _solde; // Variable privée représentant le solde du compte
  double _plafond; // Variable privée représentant le plafond de
découvert autorisé

  // Constructeur de la classe CompteBancaire prenant en paramètres le
solde initial et le plafond de découvert
  CompteBancaire(this._solde, this._plafond);

  // Méthode getter pour la propriété solde, elle permet de récupérer la
valeur du solde
  double get solde => _solde;

  // Méthode setter pour la propriété solde, elle permet de modifier le
solde tout en vérifiant les règles du plafond de découvert
  set solde(double value) {
    if (value < -_plafond) { // Vérifie si le nouveau solde dépasse le
plafond de découvert autorisé
      print("Opération refusée : dépassement du plafond de découvert
```

```
autorisé."); // Affiche un message d'erreur si la transaction est
refusée
    } else {
      _solde = value; // Affecte la nouvelle valeur au solde si la
transaction est autorisée
    }
  }

  // Méthode pour effectuer un dépôt sur le compte
  void depot(double montant) {
    _solde += montant; // Ajoute le montant au solde
  }

  // Méthode pour effectuer un retrait sur le compte
  void retrait(double montant) {
    double nouveauSolde = _solde - montant; // Calcule le nouveau solde
après le retrait
    if (nouveauSolde < -_plafond) { // Vérifie si le nouveau solde
dépasse le plafond de découvert autorisé
      print("Opération refusée : dépassement du plafond de découvert
autorisé."); // Affiche un message d'erreur si la transaction est
refusée
    } else {
      _solde = nouveauSolde; // Met à jour le solde avec le nouveau
solde calculé
    }
  }
}
void main() {
  // Création d'une instance de CompteBancaire avec un solde de 1000 et
un plafond de découvert de -500
  CompteBancaire compte = CompteBancaire(1000, 500);

  // Affichage du solde initial
  print("Solde initial : ${compte.solde}");

  // Effectuer des dépôts
  compte.depot(200);
  compte.depot(300);
  print("Solde après dépôts : ${compte.solde}");

  // Effectuer des retraits
  compte.retrait(800);
```

apcpedagogie.com

```
  compte.retrait(1000); // Ce retrait devrait être refusé car il dépasse
le plafond de découvert
  print("Solde après retraits : ${compte.solde}");
}
```

3. Pourquoi utiliser les getters et setters

- Les getters sont utilisés dans ce cas pour encapsuler l'accès aux attributs de la classe Rectangle. Voici quelques raisons pour lesquelles les getters sont utilisés :
- **Encapsulation** : Les getters permettent d'encapsuler l'accès aux données de la classe en fournissant une interface pour récupérer les valeurs des attributs. Cela permet de cacher les détails d'implémentation de la classe et de limiter l'accès direct aux attributs.
- **Contrôle d'accès** : En utilisant des getters, vous pouvez contrôler l'accès aux attributs de la classe. Par exemple, vous pouvez ajouter des conditions ou des vérifications lors de la récupération des valeurs, ce qui peut être utile pour garantir l'intégrité des données.
- **Flexibilité**: Les getters permettent de modifier la manière dont les valeurs des attributs sont récupérées sans modifier l'interface publique de la classe. Si vous décidez de changer la façon dont les attributs sont stockés ou calculés à l'avenir, vous pouvez le faire en modifiant uniquement le code du getter, sans avoir à modifier les autres parties du code qui utilisent la classe.
- **Encapsulation des changements** : En encapsulant l'accès aux attributs derrière des getters, vous pouvez modifier l'implémentation interne de la classe sans affecter le code client. Cela permet de réduire la dépendance entre les différentes parties du code et facilite la maintenance et l'évolution du logiciel.

4. Exemples

class CompteBancaire

```
class CompteBancaire {
  // Attribut privé
  double _solde = 0.0;

  // Getter pour lire l'attribut privé _solde
  double get solde => _solde;
//// Getter pour lire l'attribut privé _solde
  //double get solde {
   // return _solde;
```

apcpedagogie.com

```
  //}

  // Méthode pour effectuer un dépôt
  void deposer(double montant) {
    _solde += montant;
  }

  // Méthode pour effectuer un retrait
  void retirer(double montant) {
    if (_solde >= montant) {
      _solde -= montant;
    } else {
      throw Exception("Solde insuffisant");
    }
  }
}

void main() {
  // Création d'une instance de la classe CompteBancaire
  CompteBancaire compte = CompteBancaire();
  // Dépôt d'argent
  compte.deposer(1000);
  // Affichage du solde
  print("Solde après le dépôt: ${compte.solde}");
  // Retrait d'argent
  compte.retirer(500);
  // Affichage du solde
  print("Solde après le retrait: ${compte.solde}");
}
```

- Ce code définit une classe **CompteBancaire** avec des méthodes pour effectuer des dépôts et des retraits, ainsi qu'un attribut privé **_solde** pour stocker le solde du compte.
- Des **getters** sont utilisés pour permettre la lecture du solde sans accès direct à l'attribut privé. L'exemple dans la fonction main crée une instance de CompteBancaire, effectue un dépôt, affiche le solde, effectue un retrait, puis affiche à nouveau le solde mis à jour. class Person

```
class Person {
  // Champs privés
  String _name; // Champ privé, préfixé par "_"
  int _age;
```

```
  // Getter pour récupérer la valeur du champ _name
  String get name {
    return _name;
  }

  // Setter pour modifier la valeur du champ _name
  set name(String newName) {
    _name = newName;
  }

  // Getter pour récupérer la valeur du champ _age
  int get age {
    return _age;
  }

  // Setter pour modifier la valeur du champ _age
  set age(int newAge) {
    if (newAge >= 0) { // Vérification si l'âge est positif
      _age = newAge;
    } else {
      print("L'âge ne peut pas être négatif.");
    }
  }
}

void main() {
  var person = Person();

  // Utilisation du setter pour définir le nom et l'âge
  person.name = "John";
  person.age = 30;

  // Utilisation du getter pour récupérer le nom et l'âge
  print("Nom: ${person.name}, Âge: ${person.age}");
}
```

- Dans cet exemple :
  - On définit une classe Person avec deux champs privés _name et _age.
  - On crée des getters (get) et des setters (set) pour chaque champ privé. Ces méthodes permettent d'accéder et de modifier les champs de la classe respectivement.
  - Dans la méthode main(), on instancie un objet person de la classe Person.

apcpedagogie.com

- On utilise les setters pour définir les valeurs des champs _name et _age.
- On utilise les getters pour récupérer les valeurs des champs _name et _age.
- Remarquez comment dans le setter de l'âge, une vérification est effectuée pour s'assurer que l'âge est positif. Cela démontre que les setters peuvent également effectuer des validations ou des opérations supplémentaires avant de modifier la valeur d'un champ.

5. Applications

1. **App01**

- Créez une classe nommée "**Cercle**" avec une propriété privée "**rayon**" de type double.
- Implémentez un getter nommé "**getRayon**" pour récupérer la valeur de la propriété "**rayon**".
- Implémentez un setter nommé "**setRayon**" qui vérifie si la valeur passée est supérieure à zéro. Si c'est le cas, mettez à jour la valeur de "**rayon**". Sinon, affichez un message d'erreur.
- Testez votre classe en créant une instance de "**Cercle**", en définissant le rayon avec le setter, puis en récupérant sa valeur avec le getter.

Solution

```
class Cercle {
  double _rayon;

  double getRayon() {
    return _rayon;
  }

  void setRayon(double nouveauRayon) {
    if (nouveauRayon > 0) {
      _rayon = nouveauRayon;
    } else {
      print("Le rayon doit être supérieur à zéro.");
    }
  }
}

void main() {
```

apcpedagogie.com

```
  var cercle = Cercle();
  cercle.setRayon(5.0);
  print(cercle.getRayon()); // Affiche "5.0"
  cercle.setRayon(-2.0); // Affiche "Le rayon doit être supérieur à zéro."
}
```

- Dans cet exercice, la classe "**Cercle**" a une propriété privée "**_rayon**" et les méthodes "**getRayon**" et "**setRayon**" pour y accéder. Le setter "**setRayon**" vérifie si la valeur passée est supérieure à zéro avant de mettre à jour la valeur de "**_rayon**". Si la valeur est inférieure ou égale à zéro, un message d'erreur est affiché.

2. **App02**

- Créez une classe **Temperature** avec une variable privée **_celsius** pour représenter une température en degrés Celsius.
- La classe devrait avoir un getter celsius qui renvoie la valeur en degrés Celsius et un setter celsius qui permet de définir la valeur en degrés Celsius.
- Assurez-vous que la valeur en degrés Celsius est supérieure à -273.15, car cela représente le zéro absolu.
- Ensuite, ajoutez un getter fahrenheit qui renvoie la valeur en degrés Fahrenheit (la formule de conversion est : *Fahrenheit = Celsius * 9/5 + 32*).

Solution

```
class Temperature {
  // Attribut privé pour stocker la température en degrés Celsius
  double _celsius;

  // Constructeur pour initialiser la température en degrés Celsius
  Temperature(this._celsius) {
    // Vérifier que la température est supérieure au zéro absolu
    if (_celsius < -273.15) {
      throw ArgumentError("La température en degrés Celsius ne peut pas être
inférieure à -273.15 (zéro absolu).");
    }
  }

  // Getter pour récupérer la température en degrés Celsius
  double get celsius => _celsius;
```

apcpedagogie.com

```
  // Setter pour définir la température en degrés Celsius
  set celsius(double value) {
    // Vérifier que la nouvelle température est supérieure au zéro absolu
    if (value < -273.15) {
      throw ArgumentError("La température en degrés Celsius ne peut pas être
inférieure à -273.15 (zéro absolu).");
    }
    _celsius = value;
  }

  // Getter pour récupérer la température en degrés Fahrenheit
  double get fahrenheit => _celsius * 9/5 + 32;
}

void main() {
  // Création d'un objet Temperature avec une température initiale de 20 degrés
Celsius
  Temperature temperature = Temperature(20.0);

  // Affichage de la température en degrés Celsius et Fahrenheit
  print("Température en degrés Celsius : ${temperature.celsius}");
  print("Température en degrés Fahrenheit : ${temperature.fahrenheit}");
}
```

3. **App03**

   - Créez une classe **Person** avec les propriétés privées **_firstName** et **_lastName**. Ajoutez des getters et des setters pour ces propriétés.
   - Ensuite, ajoutez une propriété calculée fullName qui renvoie la concaténation du prénom et du nom de famille. Assurez-vous de mettre à jour la valeur de fullName à chaque fois que les getters ou les setters des propriétés **_firstName** ou **_lastName** sont appelés.

     Solution

```
class Person {
  // Attributs privés pour stocker le prénom et le nom de famille
  String _firstName;
  String _lastName;

  // Constructeur pour initialiser le prénom et le nom de famille
```

apcpedagogie.com

```
  Person(this._firstName, this._lastName);

  // Getter pour récupérer le prénom
  String get firstName => _firstName;

  // Setter pour définir le prénom
  set firstName(String value) {
    _firstName = value;
  }

  // Getter pour récupérer le nom de famille
  String get lastName => _lastName;

  // Setter pour définir le nom de famille
  set lastName(String value) {
    _lastName = value;
  }

  // Propriété calculée pour renvoyer le nom complet
  String get fullName => '$_firstName $_lastName';
}

void main() {
  // Création d'un objet Person avec un prénom "John" et un nom de famille
"Doe"
  Person person = Person("John", "Doe");

  // Affichage du prénom, du nom de famille et du nom complet
  print("Prénom : ${person.firstName}");
  print("Nom de famille : ${person.lastName}");
  print("Nom complet : ${person.fullName}");

  // Modification du prénom et du nom de famille
  person.firstName = "Jane";
  person.lastName = "Smith";

  // Affichage du prénom, du nom de famille et du nom complet mis à jour
  print("Prénom : ${person.firstName}");
  print("Nom de famille : ${person.lastName}");
  print("Nom complet : ${person.fullName}");
}
```

apcpedagogie.com

4. **App04**

- Créez une classe Product pour représenter un produit avec les propriétés privées **_name** et **_price**. Ajoutez des getters et des setters pour ces propriétés.
- Ensuite, ajoutez une propriété calculée discountedPrice qui renvoie le prix réduit du produit. Vous pouvez supposer que le prix réduit est de 10% du prix d'origine.

Solution

```
class Product {
  // Attributs privés pour stocker le nom et le prix du produit
  String _name;
  double _price;

  // Constructeur pour initialiser le nom et le prix du produit
  Product(this._name, this._price);

  // Getter pour récupérer le nom du produit
  String get name => _name;

  // Setter pour définir le nom du produit
  set name(String value) {
    _name = value;
  }

  // Getter pour récupérer le prix du produit
  double get price => _price;

  // Setter pour définir le prix du produit
  set price(double value) {
    _price = value;
  }

  // Propriété calculée pour renvoyer le prix réduit du produit
  double get discountedPrice => _price * 0.9; // Réduction de 10%
}

void main() {
  // Création d'un objet Product avec un nom "T-shirt" et un prix de 20
  Product product = Product("T-shirt", 20);

  // Affichage du nom et du prix du produit
```

apcpedagogie.com

```
  print("Nom du produit : ${product.name}");
  print("Prix du produit : ${product.price}");

  // Calcul et affichage du prix réduit du produit
  print("Prix réduit du produit : ${product.discountedPrice}");
}
```

5. **Exercice : Gestion d'un Éclairage Connecté**

- Imaginons un système qui contrôle l'intensité des ampoules intelligentes d'une maison.
- **La Classe Ampoule (Le matériel)**
- Ici, le `setter` s'assure que la luminosité reste entre 0 et 100%, et le `getter` détermine automatiquement si l'ampoule consomme beaucoup d'énergie.
  - **`ampoule.dart`**
    - Ce fichier définit les limites techniques de l'ampoule.

```
// Fichier : ampoule.dart
class Ampoule {
final String piece;
int _intensite = 0; // Privé (0 à 100)

Ampoule(this.piece);

// Accesseur pour lire l'intensité actuelle
int get intensite => _intensite;

// Getter calculé : vrai si l'intensité dépasse 80%
bool get consommationElevee => _intensite > 80;

// Mutateur avec validation des bornes
set intensite(int valeur) {
if (valeur < 0) {
_intensite = 0;
print("[$piece] Intensité réglée au minimum (0%).");
} else if (valeur > 100) {
_intensite = 100;
print("[$piece] Intensité bridée au maximum (100%).");
} else {
_intensite = valeur;
```

apcpedagogie.com

```
print("[$piece] Intensité réglée à $_intensite%.");
}
}
}
```

- **application.dart**
  - Ce fichier représente l'interface de l'utilisateur pour contrôler la maison.
- **main.dart**
  - Point d'entrée pour tester le comportement du système.

```
// Fichier : main.dart
import 'ampoule.dart';
import 'application.dart';

void main() {
var lampeSalon = Ampoule("Salon");
var monApp = ApplicationMaison("Admin");

// Test du setter avec une valeur trop haute
monApp.configurerAmbiance(lampeSalon, 150);

// Test du getter de consommation
print("Intensité réelle : ${lampeSalon.intensite}%");
print("Mode énergivore : ${lampeSalon.consommationElevee}");
}
```

```
// Fichier : application.dart
import 'ampoule.dart';

class ApplicationMaison {
String utilisateur;

ApplicationMaison(this.utilisateur);

void configurerAmbiance(Ampoule lampe, int niveau) {
print("--- Action de $utilisateur ---");
lampe.intensite = niveau; // Utilise le setter

if (lampe.consommationElevee) {
```

apcpedagogie.com

```
  print("Attention : L'ampoule de la ${lampe.piece} consomme beaucoup !");
}
}
}
```

Cours Dart

Source : https://apcpedagogie.com/les-getters-et-setters-en-dart · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com