Exercice

Énoncé

Créer une classe CompteBancaire qui simule un compte bancaire, en appliquant les principes d'encapsulation pour protéger les données à l'aide d'attributs privés, de getters, de setters (via @property), et de méthodes publiques avec des validations rigoureuses.

Consignes

Définir la classe CompteBancaire avec les attributs suivants, tous privés :

- titulaire : Nom du titulaire du compte (chaîne).
- numero_compte : Numéro unique du compte (chaîne).
- solde : Solde actuel du compte (nombre, initialisé à 0 par défaut).
- historique : Liste des transactions effectuées (liste de chaînes).

Utiliser @property pour retourner les informations suivantes :

- titulaire (lecture seule).
- numero_compte (lecture seule).
- historique (retourne une copie de la liste pour éviter les modifications externes).

Implémenter un setter via @property qui valide que :

- Le numéro de compte est un nombre positif (entier ou flottant).
- Chaque modification est enregistrée dans l'historique.

Ajouter des méthodes publiques :

- deposer(montant) : Ajoute un montant au solde, avec les validations suivantes :
  - Le montant est un nombre positif.
  - Enregistre la transaction dans l'historique (ex. : "Dépôt: +500, Solde: 1500").
- retirer(montant) : Retire un montant du solde, avec les validations suivantes :
  - Le montant est un nombre positif.
  - Le solde est suffisant pour le retrait.
  - Enregistre la transaction dans l'historique (ex. : "Retrait: -200, Solde: 1300").

Redéfinir __str__ :

Retourne une représentation textuelle du compte, par exemple : "Le propriétaire du Compte numéro 123456789 est Alain. Le solde actuel est 1300$".

Tester le code :

- Créer un compte avec un titulaire, un numéro, et un solde initial.
- Effectuer des dépôts et retraits valides.
- Tester des cas d'erreur (retrait trop élevé, dépôt négatif, solde négatif).
- Afficher le solde et l'historique.
- Vérifier l'encapsulation en essayant d'accéder directement aux attributs privés.