**Gestion des Publicités en Ligne**

Une entreprise spécialisée dans la publicité digitale souhaite développer une application permettant de gérer différentes campagnes publicitaires.
Chaque publicité a un **nom, un budget et un canal de diffusion** (ex : réseaux sociaux, moteurs de recherche, e-mailing, influenceurs).

Il existe plusieurs types de publicités :

1. **Publicité Google Ads** (modèle basé sur le **Coût Par Clic (CPC)**).
2. **Publicité Facebook Ads** (modèle basé sur le **Coût Par Mille (CPM)**).
3. **Publicité YouTube Ads** (modèle basé sur le **Coût Par Vue (CPV)**).
4. **Publicité d'Influenceurs** (modèle basé sur le **Coût Par Engagement (CPE)**).

L'application devra être développée en **programmation orientée objet** et utiliser **Tkinter** pour son interface graphique.

**Partie 1 : Conception des classes et principes de la POO (10 pts)**

1. **(1 pt)** Définissez une **classe abstraite AdCampaign** contenant les attributs communs (nom, budget, canal) et une méthode abstraite calculer_portée().
2. **(2 pts)** Ajoutez une **méthode d'instance afficher_details()** dans AdCampaign qui retourne une chaîne contenant les informations de la campagne sous la forme :

```
Campagne: Promo Noël | Budget: 2000€ | Canal: Réseaux sociaux
```

Exemple d'utilisation :

```
campagne = AdCampaign("Promo Noël", 2000, "Réseaux sociaux")
print(campagne.afficher_details())
```

3. **(2 pts)** Implémentez la classe GoogleAdsCampaign qui hérite de AdCampaign, en ajoutant un attribut cpc (coût par clic) et en définissant calculer_portée() pour estimer **le nombre de clics** (budget / cpc).
4. **(2 pts)** Implémentez la classe FacebookAdsCampaign avec un attribut cpm (coût pour 1000 impressions) et une méthode calculer_portée() retournant **le nombre d'impressions** ((budget / cpm) * 1000).

5. **(1 pt)** Implémentez la classe YouTubeAdsCampaign avec deux attributs : cpv (coût par vue) et budget, et la méthode calculer_portée() qui retourne **le nombre de vues estimées** (budget / cpv).
6. **(2 pts)** Ajoutez une **méthode magique __str__()** dans AdCampaign pour afficher une campagne sous la forme :

| Campagne: Promo Noël \| Budget: 2000€ \| Canal: Réseaux sociaux |
|---|

7. **(1 pt)** Ajoutez une **méthode magique __eq__()** permettant de comparer si deux campagnes sont identiques (même nom, même budget, même canal).

**Partie 2 : Gestion des erreurs et encapsulation (5 pts)**

8. **(2 pts)** Créez une **exception personnalisée InvalidBudgetError** qui sera levée si le budget est négatif ou nul.
9. **(1 pt)** Modifiez le constructeur de AdCampaign pour utiliser des **getters et setters** qui empêchent l'affectation d'un budget négatif.
10. **(2 pts)** Modifiez calculer_portée() pour qu'elle lève une **exception NotImplementedError** si elle n'est pas implémentée dans une sous-classe.

**Partie 3 : Interface Graphique avec Tkinter (5 pts)**

11. **(3 pts)** Créez une fenêtre Tkinter avec :
    - Un **champ de saisie** pour le nom de la publicité.
    - Un **champ de saisie** pour le budget.
    - Une **liste déroulante** permettant de choisir le type de publicité (Google Ads, Facebook Ads, YouTube Ads, Influenceurs).
    - Un **bouton "Ajouter"** qui crée une campagne et l'affiche dans une Listbox.
12. **(2 pts)** Ajoutez une gestion des erreurs pour **empêcher l'ajout d'une publicité avec un budget négatif** et afficher un message d'erreur Tkinter (messagebox.showerror).
13. **(3 pts)** Ajoutez une Treeview pour afficher les publicités avec les colonnes :
    - Nom,
    - Budget (€),
    - Canal,
    - Type,

- Portée estimée.

Le bouton "Ajouter" doit également **mettre à jour la Treeview** avec les informations de la publicité.

14. **(1 pt)** Ajoutez un bouton permettant de **supprimer une publicité** sélectionnée dans la Treeview.
15. **(1 pt)** Ajoutez une fonctionnalité permettant de **modifier une publicité existante** (ex : changer son budget).