Programmation Orientée Objet (POO):

Cahier de charges :

Une entreprise de la place souhaite mettre en place une application web en utilisant le langage PHP (langage conçue pour le web) avec le **paradigme de la programmation orientée objet**.Vue l'envergure de ladite application, l'entreprise souhaite partager les taches entre les groupes de développeurs et votre groupe (**groupe 1**) en fait parti. Pour votre mission regarde la partie mentionnant le numéro de votre groupe (**groupe** 1) et le cadre coloré en vert comme couleur de fond.

Les notions en POO :

GROUPE 1

- ✔ Classe
- ✔ Object
- ✔ Les propriétés
- ✔ Les modificateurs de visibilité : public , private
- ✔ Les constructeurs
- ✔ Les destructeurs
- ✔ Le mot clé $**this**
- ✔ Les méthodes
- ✔ Interaction entre les objets
- ✔ Héritage
- ✔ Appel du constructeur parent
- ✔ Surcharge d'une méthode

Tous les GROUPES :

1. Créer un nouveau projet sur votre serveur et nommé le **ifntiaccountbank**
2. Créer un répertoire et nommé le **src** à la racine de votre projet
3. Créer un fichier et nommé le **index.php** à la racine de votre projet

GROUPE 1:

Recherche :

1. Qu'est-ce que la programmation orientée objet ?
2. A quoi sert la programmation orientée objet ?
3. POO est un nouveau langage de programmation ?
4. Différence entre la programmation orientée objet et la programmation procédurale
5. Qu'est-ce qu'une classe ? Objet ?
6. Faire une équivalence entre variable et propriété en PHP.

TAF (Travail A Faire) :

1. Dans le dossier **src** de votre projet, créer une classe (mais on dit définir une classe) donc utiliser alors définir une classe au lieu de créer une classe. Le nom de votre classe s'appelle **BankAccount** qui signifie en français compte bancaire (Prenez l'habitude d'utiliser l'anglais dans vos projet)
2. Dans votre classe, définir deux (02) propriétés appelées respectivement **accountNumber** (numéro de compte) et **balance** (solde en français). Toutes les propriétés sont publiques, ce qui signifie qu'on peut y accéder depuis n'importe où (utiliser le mot clé **public** pour les rendre publique). Noter aussi si vous n'avez rien précisé un mot clé **public**, **private** ou **protected**, c'est le public qui est utilisé par **défaut.**

Remarque :
Suivez ces règles lorsque vous voulez définir une classe :

- Les noms des classes sont **upper camel case** c'est-à-dire chaque mot commence par la lettre **majuscule.**
  Exemple : **M**y**S**ite**U**ser , **I**fnti**S**tudent , **T**ogo**P**opulation ...
- Si le nom de la classe est un **nom**, mettez le au **singulier**
  Exemple :
  Si j'ai une classe qui défini les noms des utilisateurs, je vais appeler le nom de ma classe comme ceci : User au lieu de User~~s~~
- Définissez chaque classe dans un fichier séparé sauf dans certains cas
- Le nom du fichier doit être le même que le nom de la classe.
  Exemple :
  Si le nom de la classe est **IfntiStudent** alors le nom du fichier sera **IfntiStudent**

3. Dans votre fichier **index.php**, créer un objet de la classe mais on dit instancier une classe donc utiliser **instancier** une classe au lieu de créer un objet de la classe. N'oublier pas d'inclure le fichier **BankAccount.php** qui contient la classe à instancier et qui se trouve dans le dossier **src** de votre projet. Si vous oubliez de le faire, vous aurez la belle erreur suivante : **Fatal error: Uncaught Error: Class "BankAccount" not found** qu'il y a une erreur fatal car la classe **BankAccount** n'est pas trouvée. Nommer votre objet **$bankCustomer1.** (Comme rappelle vous avez déjà appris à inclure les scripts avec les directives suivantes : **include**, **incude_once**, **require** et **require_once**)

Après avoir instancier une classe, faire un **var_dump** sur l'objet instancié pour voir ce qu'il contient (cela peut vous aider à comprendre ce qu'est un objet et son contenu)

4. Après avoir instancié une classe, accéder à ses propriétés **accountNumber** et **balance** avec l'opérateur de l'objet qui est une flèche comme ceci → (c 'est tiret de 6 suivit du signe supérieur) puis assigné-les des valeurs qui sont le **numéro du compte client 1 et son solde** puis afficher ces valeurs sur l'écran.
5. Faire la même chose que les numéros **3** et **4** mais cette fois-ci le nom de l'objet est **$bankCustomer2** puis les valeurs doivent être différentes de celles de **$bankCustomer1**
6. En premier lieu, vos propriétés sont publics (accessibles partout), essayer maintenant de les mettre **privé** en utilisant le mot clé **private** puis essayer de les accéder ou tenter d'afficher leurs valeurs. Répondez alors aux questions suivantes :

   - Qu'est-ce qui se passe ?
   - Donnez-nous le conseil dans le choix des **modificateurs de visibilité.**
   - Quelles solutions proposez-vous pour pouvoir utiliser nos propriétés ?

Note : C'est ces solutions qui permettront au groupe suivant (Groupe 2) de continuer à développer le projet.

Fin GROUPE 1

GROUPE 2:

Recherche :

1. C’est quoi un **constructeur** et promotion de constructeur puis un **destructeur** ?
2. Faut-il implémenter explicitement les constructeurs ou les destructeurs ?
3. A quoi sert le mot clé **$this**?
4. Y-a t-il un ramassage miette (**Garbage collector** comme en Java) en PHP .
5. C’est quoi une méthode en POO ?
6. C’est quoi les **setters** et les **getters ?** Montrez comment elles sont nommées conventionnellement.
7. Faire une équivalence entre fonction et méthode en PHP.

TAF (Travail A Faire) :

Nous remarquons que le **groupe 1** a changé la visibilité des propriétés qui sont passées de publique (via le mot clé **public**) en privé (par le mot clé **private**), c’est ce qui nous empêche d’accéder directement à nos propriétés, c’est pourquoi ils nous ont proposé d’utiliser les fonctions pour pouvoir les utiliser.Voici ce que vous allez faire pour y arriver :

1. Dans le fichier **BankAccount** qui se trouve dans le dossier **src**, définissez deux (02) méthodes appelées respectivement **getAccountNumber** qui permet de voir ou d’obtenir le numéro du compte bancaire et **setAccount** qui permet de modifier le numéro du compte bancaire.
2. Dans le fichier **BankAccount** qui se trouve dans le dossier **src**, ajouter deux nouvelles (02) méthodes appelées respectivement **getBalance** qui permet de voir ou d’obtenir le solde du compte bancaire et **setAccount** qui permet de modifier le solde du compte bancaire. Au total nous avons déjà quatre (04) méthodes.
3. Dans le fichier **BankAccount** qui se trouve dans le dossier **src**, définissez encore deux (02) nouvelles méthodes appelées respectivement **deposit** qui permet de faire le dépôt d’argent sur le numéro du compte bancaire et **withdraw** qui permet de retirer l’argent sur le numéro du compte bancaire. Au total nous avons déjà six (06) méthodes.

Attention : L’objectif d’utiliser les mots clés **private** sur vos propriétés, c’est de pouvoir les contrôler donc n’oublier pas de faire toutes les vérifications lorsque vous définissez vos méthodes.Voici un exemple du traitement (ou vérification) : Si un client veut faire un retrait d’argent, l’on peux vérifier si son solde est supérieur au montant du retrait ou bien s’il a atteint le quota de limitation de retrait fixé par la banque. Également si un client veut faire un dépôt l’on peux vérifier si le montant à déposer n’est pas inférieur à 0 Francs etc...

4. Utiliser les mots clés **public** pour vos méthodes et propriétés sinon vous n'allez plus pouvoir accéder ni à vos propriétés ni à vos méthodes.
5. Utiliser le mot clé **$this** pour implémenter vos méthodes
6. En réalité, lorsqu'un client de la banque crée un compte, il doit en avoir le même jour un **numéro du compte bancaire** et au **moins une somme minimale sur son compte**. Donc pour que cela soit possible, on doit définir un constructeur qui exige à ce qu'une fois un compte bancaire est crée, il doit en avoir numéro du compte et solde minimale.
7. Savez vous que le langage PHP tout comme les autres évolue vite? Si oui, depuis PHP 8 (version 8) ,il y a un nouveau concept appelé promotion de constructeur. Montrez nous comment les implémenter en POO (Si vous utilisez PHP inférieur à la version 8, montrez votre compétence pour le mettre à jour quelques soit votre système d'exploitation).
8. Testez à nouveau le fonctionnement de votre classe en instanciant deux (02) objets de la classe **BankAccount.** Nommez les deux (02) objets respectivement **$client2** et **$client3** et suivez les consignes suivantes :
9. Vos objets doivent en même temps avoir un numéro du compte et un solde minimale de votre choix.
10. Ensuite, appeler les méthodes **getters** et **setters** pour voir leur importances.
11. Utiliser les méthodes **deposit** et **withdraw** pour montrer encore plus l'importance de ces méthodes.
12. Vous venez de voir que vous avez appelé les méthodes séparément, au vue de certains développeurs cette manière d'appeler ces méthodes, donc rappelez ces méthodes en utilisant **chaînage des méthodes**
13. Montrer nous enfin comment définir un destructeur et comment l'utiliser pour détruire nos objets. N'oubliez pas de nous dire s'il est important de définir obligatoirement le destructeur et pourquoi.

Fin GROUPE 2

GROUPE 3 :

Recherche :

1. C’est quoi l’héritage en PHP dans POO ?
2. A quoi sert le mot clé **extends** ?
3. PHP supporte t-il l’héritage multiple ?
4. Comment appelle t-on la classe qui a hérité d’une autre classe ? Et celle dont a hérité ?
5. Expliquer nous la notion de surcharge de méthodes
6. Expliquer nous la notion **d’abstraction**
7. Quand est-ce qu’on utilise le modificateur de visibilité **protected** en PHP

TAF (Travail A Faire) :

Si vous avez bien suivi la présentation, le projet utilise une seule classe qui est **BankAccount**, maintenant vous êtes solliciter à étendre cette classe.Votre mission est la suivante :

1. Définissez une autre méthode et nommez-la **transfer** qui prend en paramètre un objet **BankAccount** représentant le compte bénéficiaire et le montant à transférer. Cette méthode permet de transférer l’argent entre deux objets **BankAccount**. Mais attention, il faut vérifier d’abord que le solde du compte **expéditeur** est suffisant pour effectuer le transfert, puis effectue le retrait et le dépôt correspondant sur les deux comptes.
2. Testez cette nouvelle méthode sur les objets **$client2** et **$client3** que le **groupe 2** vient d’instancier.
3. **Rappelle** : souvenez-vous bien lors de la présentation du **groupe 1**, il a été dit que chaque fichier contient une seule classe et que le nom de chaque classe doit correspondre à son ficher. Donc dans le dossier **src** créer deux fichiers et nommée les respectivement C**heckingAccount** et **SavingsAccount.** La classe **CheckingAccount** ajoute une propriété privée **$transactionFee** qui représente les frais de transaction pour un compte courant. Elle contient également une méthode **deductTransactionFee()** qui déduit les frais de transaction du solde du compte. La classe **SavingsAccount** ajoute une propriété privée **$interestRate** qui représente le taux d'intérêt pour un compte d'épargne. Elle contient également une méthode **addInterest()** qui calcule l'intérêt du compte et l'ajoute au solde. Ces deux classes filles étendent la classe **BankAccount** et **héritent** de ses propriétés et méthodes. En ajoutant des propriétés et des méthodes spécifiques à chaque type de compte, elles permettent de créer des objets de compte courant et d'épargne avec des fonctionnalités supplémentaires.
4. Tester vos nouvelles classes en instanciant les deux(02) classes dans le fichier **index**.**php**
5. Implémentez le constructeur de la classe parente dans les classes filles .
6. Si on souhaite remplacer la fonctionnalité d'une méthode héritée dans une classe fille, on peut utiliser la technique **d'override** de méthode. Pour cela, on redéfinit simplement la

méthode dans la classe fille en utilisant le même nom que la méthode héritée. Ainsi, lorsqu'on appelle cette méthode sur un objet de la classe fille, c'est la méthode redéfinie qui est appelée, et non plus la méthode héritée : illustrer alors cela !

7. Si notre classe a des classes filles, on peut la rendre abstraite pour indiquer que cette classe doit être héritée pour être utilisée, et que des instances de cette classe ne devraient pas être créées directement. Alors, appliquer l'abstraction sur la classe **BankAccount.** Sans détaille, quelle sera à nouveau le modificateur de visibilité à utiliser sur les propriétés ?

FIN GROUPE 3

GROUPE 4 :

Recherche :

1. Comment appliquer le modificateur de visibilité **protected** des propriétés et quand est-ce qu'il faut l'utiliser ?
2. C'est quoi le Polymorphisme ?
3. C'est quoi l'Interface ?

TAF (Travail A Faire) :

1. Si vous remarquez bien, le groupe 3 nous a proposé d'appliquer le modificateur de visibilité **protected** au niveau de nos propriétés si nous voulons que nos classes filles héritent de ces propriétés mais ils n'ont pas détaillé. Si ce n'est ainsi, on vous invite à le faire : mettez toutes les propriétés de la classe **BankAccount** en **protected.**
2. Transformer notre classe **BankAccount** en interface et implémenter cette classe au niveau des classes **CheckingAccount** et **SavingsAccount (**Ces classes n'héritent plus de **BankAccount** mais l'implémentent)
3. Si vous avez implémenté l'interface BankAccount, appliquer le polymorphisme au niveau des méthodes **getBalance** et **deposit.**
4. Pour voir l'importance du polymorphisme, il faut instancier ces classes qui viennent implémenter l'interface **BankAccount** et appeler les classes **getBalance** et **deposit.** Quelle remarque faites-vous ?

Fin du GROUPE 4

GROUPE 5 :

Recherche :

1. Quel avantage offre le **trait** ?
2. A quoi sert le mot clé **use** dans le cas du **trait** ?

TAF (Travail A Faire) :

Votre mission est de nous parler de **trait** en PHP. Donc utiliser les trois éléments (**BankAccount** , **ChekingAccount** et **SavingsAccount**) pour implémenter le **trait**. Définissez ce **trait** au niveau du nouveau **trait** appelé **BankTransaction** et utilisé ce **trait** dans **ChekingAccount** et **SavingsAccount**. Qui implémentent l'interface **BankAccount.** Suivez ces étapes pour illustrer la notion du **trait** :

1. Dans le dossier **src**, définissez un **trait BankTransaction** qui contient les méthodes **deposit**() et **withdraw**().
2. Les classes **ChekingAccount** et **SavingsAccount** implémentent l'interface **BankAccount**
3. **BankAccount** contient deux (02) méthodes : **getAccount** et **getBalance**
4. Ensuite utilisé ce **trait** dans les classes **CheckingAccount** et **SavingsAccount** en utilisant la syntaxe **use BankTransaction**. (N'oublier pas que ces deux (02) classes implémentent l'interface **BankAccount** donc utilisent forcément ses méthodes)
5. Instancier des objets de type **CheckingAccount** et **SavingsAccount** dans le fichier **index.php** pour utiliser les méthodes **deposit**() et **withdraw**() du trait **BankTransaction**, car elles ont été héritées par les classes. Cela va vous permettre d'éviter de répéter du code identique dans les deux classes, et de bénéficier de la flexibilité offerte par les **traits**.

FIN DU GROUPE 5

GROUPE 6:

Recherche:

1. Parler nous des **Méthodes** et **propriétés** statiques et à quelle occasion les utiliser ?
2. Donner la différence qui existe entre les **propriétés de classe** et les **propriétés d'instance.**
3. Donner une explication du mot clé **static**
4. Donner une explication du mot clé **self** et de l'opérateur **::**
5. Comment utiliser les méthodes de **classe à l'intérieur** de la classe et à **l'extérieur de la classe** ?
6. Donner l'importance des **constantes**
7. Parler nous de **late static binding** et comment l'utiliser efficacement.

TAF (Travail A Faire) :

Reprenez le code complet du client **GROUPE 1** et suivre les consignes suivantes :

1. Ajouter deux propriétés statiques **$numberOfAccounts** et **$totalBalance**. La première propriété (**$numberOfAccounts)** contient le nombre total de comptes bancaires créés, et la seconde contient le solde total de tous les comptes bancaires créés (**$totalBalance**).
2. Ajouter deux (02) méthodes statiques de ces nouvelles propriétés ($**numberOfAccounts** et $**totalBalance**) et nommez-les **getNumberOfAccounts**() et **getTotalBalance**(). Ces méthodes renvoient respectivement le nombre **total de comptes bancaires** créés et le **solde total de tous les comptes bancaires** créés. N'oubliez pas d'utiliser le mot-clé **self** pour accéder aux propriétés et méthodes statiques à l'intérieur de la classe.
3. Créer deux comptes bancaires (**$client1** et **$client2**) en utilisant le constructeur de la classe **BankAccount**. Ensuite appeler les méthodes statiques **getNumberOfAccounts**() et **getTotalBalance**()

Fin du GROUPE 6