# Programmation Orientée Objet ( POO ):

## Voici un cahier de charge :

Une entreprise de la place souhaite en place une application web en utilisant le langage PHP (langage conçu 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**) 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
- ✔ Surchargée une méthode
- ✔ modificateur de visibilité protected
- ✔ interface
- ✔ polymorphisme
- ✔ trait
- ✔ méthodes et propriétés statiques
- ✔ Le mot clé static
- ✔ Le mot clé self
- ✔ Opérateur ::
- ✔ Les constantes de classes
- ✔ Late static binding
- ✔ Les méthodes magiques
- ✔ Faire des recherches sur Les constantes de classes
- ✔ Savoir maîtriser Late Static Binding
- ✔ Comprendre comment les méthodes magiques fonctionnent : _ _get ( ) , _ _ set( ), _ _ call ( ), _ _ callStatic ( ), _ _ invoke( ) , _ _toString ( )
- ✔ Sérialisation des Objets avec la méthode **serialize ( )**
- ✔ Désérialisation des objets avec la méthode **unserialize ( )**
- ✔ clonage des objets
- ✔ Comparaison des objets
- ✔ Les classes anonymes
- ✔ Les names spaces
- ✔ Le mot clé use dans les namespaces
- ✔ le mot clé **as**(**a**lia**s**) dans les namespaces
- ✔ spl_autoload_register ( )
- ✔ la fonction magique _ _autoload( ) (obsolète)
- ✔ composer
- ✔ psr-4
- ✔ set_error_handler()
- ✔ set_exception_handler()
- ✔ try … catch ...finally
- ✔ Throwable

- ✔ Exception
- ✔ class_exists()
- ✔ method_exists ( )
- ✔ property_exists ( )

## 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 object ?
2. -A quoi sert la programmation orientée object ?
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 du compte) et **balance** (solde en français).Toutes les propriétés sont publiques , ce qui signifie qu'on peut les accéder partout (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 en **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 Users

- 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 cette belle erreur suivant : **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 **$bankCustomer1 .**(Comme rappelle vous avez déjà appris à inclure les scripts avec les fonctions suivantes : **include** , **incude_once** , **require** et **require_once** )
   Après avoir instancier une classe , faire un **var_dump** sur l'objet instancié pour voir ce qu'il contient (cela peut vous aider à comprendre ce qu'un objet et son contenu)
4. Après avoir instancié une classe , accédée à leur propriété **accountNumber** et **balance** avec l'opérateur de l'objet qui est une flèche comme ceci → (c 'est tiret de 6 suivit du signe supérieur ) puis assigné-les des valeurs qui sont le **numéro du compte client 1 et le son solde** puis afficher ces valeurs sur l'écran.
5. Faire la même chose que les numéros **3** et **4** mais cette fois-ci le nom de l'objet est **$bankCustomer2** puis les valeurs doivent être différentes à celle de **$bankCustomer1**
6. En premier lieu , vos propriétés sont public (accessible 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 ces questions suivantes :

   - Qu'est-ce qui se passe ?
   - Donnez-nous le conseil dans le choix des **modificateurs de visibilité.**
   - Quelles solutions proposez-nous 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 le **constructeur** et promotion de constructeur puis le **destructeur**?
2. Faut-il implémenter explicitement les constructeur ou le destructeur ?
3. A quoi sert le mot clé **$this**?
4. Y-a t-il le ramassage miette (**Garbage collection** comme en Java ) en PHP .

5. C’est quoi une méthode en POO et les **setters** et **getters** puis montrez comment elles sont nommées conventionnellement?
6. 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 dossir **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 dossir **src** ,définissez encore nouvelles deux (02) 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 , je peux vérifier si son solde est supérieur au solde du retrait ou bien de voir si il y a une limitation d’un cota que la banque limite ou bien si un client veut faire un dépôt je peux vérifier si le montant à déposer n’est pas inférieur à 0 Francs etc.

4. Utiliser les mots clés **public** pour vos méthodes sinon vous n’allez plus pouvoir accéder ni vos propriétés ni 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 ces consignes suivantes :
9. Vos objets doivent en même temps avoir un numéro du compte et un solde minimale de votre choix.
10. Ensuite , appelée les méthodes **getters** et **setters** pour voir ses 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 ,sous les yeux 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 l’héritage multiple ?
4. Comment appelle t-on la classe qui a hérité une autre classe ? Et celle qu’on 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.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 correspondants 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 appliqué modificateur de visibilité **protected** des propriétés et c'est quand est-ce qu'il faut l'utiliser ?
2. C'est quoi le Polymorphisme ?
3. Interface c'est quoi ?

## 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 classe fille hérite 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 **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 et 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 ) :

**V**otre 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. Parler nous la différence entre les **propriétés de classe** et les **propriétés d'instance.**
3. Le mot clé **static**
4. Le mot clé **self** et l'opérateur **::**
5. Comment utiliser les méthodes de **classe à l'intérieur** de la classe et à **l'extérieur de la classe** ?
6. Expliquer nous 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 ces 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 $**totalBalance** ) et nommez-les **getNumberOfAccounts**() et **getTotalBalance**() .Ces méthodes renvoient respectivement le nombre **total de comptes bancaires** créés et le **solde total de tous les comptes bancaires** créés.N'oubliez pas d'utiliser le mot-clé **self** pour accéder aux propriétés et méthodes statiques à l'intérieur de la classe.
3. Créé deux comptes bancaires (**$client1** et **$client2** ) en utilisant le constructeur de la classe **BankAccount**. Ensuite appelé les méthodes statiques **getNumberOfAccounts**() et **getTotalBalance**()

Fin du GROUPE 6

# GROUPE 7 :

## Recherche :

1. Faire des recherches sur Les constantes de classes
2. Savoir maîtriser Late Static Binding
3. Comprendre comment les méthodes magiques fonctionnent : _ _get ( ) , _ _ set( ), _ _ call ( ), _ _ callStatic ( ), _ _ invoke( ) , _ _toString ( )

## TAF (Travail A Faire ) :

En reprenant le code du **GROUPE 2** mais toutes les propriétés doivent être en privée puis suivre ces étapes suivantes :

1. Définir la méthode **_ _contruct( )** en utilisant la technique de la promotion du constructeur qui vont initialiser les propriétés de la classe **BankAccount**
2. Dans cette même classe, définissez les méthodes magiques **_ _get( )**, **_ _set( )** et **_ _call( )** pour gérer l'accès aux propriétés et aux méthodes de la classe.La méthode **_ _get( )** doit être appelée automatiquement lorsque l'on tente d'accéder à une propriété qui **n'existe pas** ou qui est **protégée** (**protected**) ou **privée(private)**. Dans notre cas , il faut qu'elle renvoie simplement la valeur de la propriété si elle existe.Quant à la méthode **_ _set( )** elle doit être appelée automatiquement lorsque l'on tente de modifier la valeur d'une propriété qui **n'existe pas** ou qui est **protégée (protected)** ou **privée (private)**. Dans notre exemple, elle doit modifier simplement la valeur de la propriété si elle existe. Enfin, la méthode **_ _call()** doit être appelée automatiquement lorsque l'on tente d'appeler une méthode qui **n'existe pas** ou qui est **protégée** (protected)ou **privée**(private). Dans notre exemple, elle va permettre de définir un comportement personnalisé pour les méthodes **deposit**( ) et **withdraw**( ). La méthode **deposit**( ) ajoute le montant spécifié au solde du compte, tandis que la méthode **withdraw**() retire le montant

Fin du GROUPE 7

# GROUPE 8 :

# Recherche :

1. Sérialisation des Objets avec la méthode **serialize ( )**
2. Désérialisation des objets avec la méthode **unserialize ( )**
3. clonage des objets
4. Comparaison des objets
5. Les classes anonymes

# TAF (Travail A Faire ) :

Dans le code du **groupe 7** , on s'est contenté de définir notre classe **BankAccount** sans instancier les objets de cette classe et c'est ce que nous allons faire dans ce groupe mais en travaillant plus sur les objets.Voici ce que vous devez faire :

1. Instancier d'abord la classe **BankAccount** et nommé cet objet **$compte**
2. Sérialiser cet objet en utilisant la méthode **serialize**() qui permet de transformer l'objet en une chaîne de caractères et stocké çà dans la variable **$unserialized**
   **NB** :Il est important de noter que pour utiliser la méthode **serialize**( ), les propriétés de l'objet doivent être accessibles (c'est-à-dire **publics** ou avec des méthodes **d'accès**).
3. Afficher la chaîne de caractères sérialisée
4. Désérialiser ensuite cette chaîne de caractères en un objet **$compte** avec la méthode **unserialize( )**
5. Ré-afficher l'objet $**compte** désérialisé
6. Instancier encore la classe **BankAccount** et nommé cet objet **$compte1**
7. Cloner cet objet $**compte1** et nommé le nouveau objet **$compte2**
8. Modifier le solde du **$compte2**
9. Afficher les soldes des deux objets **$compte1** et **$compte2**
10. Que remarquez vous ?
11. Utiliser les deux (02) façons ( = = = et méthode **equals ( )** ) pour comparer les objets $**compte1** et $**compte2**
12. Comment insérer alors la classe **anonyme** pour modifier le comportement de votre classe **BankAccount ?**

Fin du GROUPE 8

# GROUPE 9 :

## Recherche :

1. Les namespaces
2. Le mot clé use dans namespaces
3. le mot clé alias

## TAF (Travail A Faire ) :

1. Reprenez le code de n’importe quel groupe et ajouter les **namespaces**
2. Après l’ajout des **namespaces** ,utiliser le mot clé **use** + **namespace** pour mettre en valeur le mot clé use d’une part et d’autre part appliquer aussi le mot clé **alias** puis dites nous dans quel cas qu’on utilise les **alias**.

Fin du GROUPE 9

# GROUPE 10 : (Très important )

## Recherche :

1. Chargement automatique des classes à l'aide de la fonction **spl_autoload_registrer( )**
2. Chargement automatique des classes à l'aide de la fonction _ **_autoload( )** mais obsolète
3. Chargement automatique des classes à l'aide du gestionnaire des paquets de PHP appelé **composer**
4. psr-4

## TAF (Travail A Faire ) :

**1 - J**e vous rappelle qu'il est très recommandé de conserver chaque classe PHP dans un fichier séparé et que le nom de la **classe** doit être le même que le nom du **fichier**.Si j'ai un fichier appelé **Person.php** , ce fichier doit contenir la classe **Person**. Jusqu'ici rien de nouveau car c'est juste un rappel.

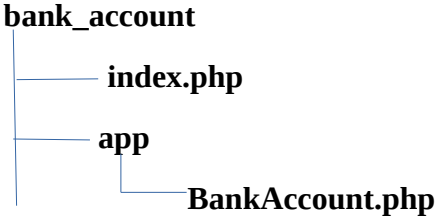
Par ailleurs, nous avons dit qu'avant d'utiliser la classe dans un autre fichier , on lui charge à l'aide de l'une des instructions suivantes : **require** , **require_once** , **include** ou **include_once**

Supposons que le nom de mon projet s'appelle **bank_account**

**D**ans ce projet , j'ai un dossier appelé **app** qui contient le fichier **BankAccount.php** lui même contient la classe **BankAccount**.

Ensuite, à la racine de ce projet ,j'ai un fichier appelé i**ndex.php**

Voici cette structure en schéma :




Voici le code du fichier **BankAccount.php** :

```
class BankAccount {
    private $accountNumber;
    private $balance;

    public function __construct($accountNumber, $balance) {
        $this->accountNumber = $accountNumber;
        $this->balance = $balance;
    }
```

```
    public function deposit($amount) {
        $this->balance += $amount;
    }

    public function withdraw($amount) {
        if ($this->balance >= $amount) {
            $this->balance -= $amount;
        } else {
            throw new Exception("Insufficient funds");
        }
    }

    public function getBalance() {
        return $this->balance;
    }

    public function getAccountNumber() {
        return $this->accountNumber;
    }
}
```

Cette classe représente un compte bancaire avec un numéro de **compte** et un **solde**. Elle contient également des méthodes pour effectuer des **dépôts** et des **retraits**, ainsi que pour obtenir le **solde** et **le numéro de compte.**

Pour utiliser la classe **BankAccount** dans le fichier **index.php** ,on doit importer le fichier **BankAccount.php** qui contient cette classe (sinon on doit avoir une erreur **Class "BankAccount" not found** ) et pour importer cette classe nous utilisons l'une des instructions suivante : **require** , **require_once** , **include** ou **include_once** comme ceci :

```
require_once 'app/BankAccount.php';
```

Voici un exemple complet du fichier **index.php :**

```
<?php
require_once 'app/BankAccount.php';
$client1 = new BankAccount("250454744777177",25000);
echo $client1->getBalance();
```

Bien sûr cette solution fonctionne bien dans le cas où nous avons un petit nombre de fichiers.Dans un grand projet (si on veut faire ***require de 50 classes***) ,vous allez vous rendre compte que ceci devient fastidieux et souvent vous allez vous perdre,c'est pourquoi cette méthode est limitée.Pour pallier à ce problème nous allons utiliser une fonction intégrée de PHP appelé **spl_autoload_registrer ( )** au détriment de la fonction magique **_ _autoload( )** obsolète depuis **7.2.0** c'est-à-dire n'est plus utilisée.La question est la suivante :
Utiliser la fonction **spl_autolod_registrer ( )** pour inclure la classe **BankAccount** du projet que je viens de créer dans ce **groupe 10** c'est-à-dire le projet **bank_account**

**2 –** Par ailleurs ,je tiens à vous dire que la méthode précédente fonctionne aussi très très bien car elle résout beaucoup de problème et si vous remarquez bien , nous utilisons toujours **require** ou **include** avec la fonction **spl_autoload_registrer ( )** mais pas trop que la première méthode. Peut-être vous n'avez pas encore développé un site complexe,vous allez remarquer très vitre la limite de la fonction **spl_autoload_registrer ( )** ,donc pour être à l'aise dans vos projets web PHP , je vous conseille d'utiliser un outil simple ,fiable et extrêmement puissant du nom **composer.**
**Composer** est partout et si vous êtes développeur PHP ,c'est l'un des outils que vous devrez avoir dans vos outils de développement .Il est partout et les framework PHP l'utilisent beaucoup.Donc vous devez comprendre tous les écosystèmes autour de celui-ci
En effet **Composer** est ce que l'on appelle le **gestionnaire de dépendances** pour le langage PHP. Au faite , pour vous faire comprendre le **gestionnaire de dépendance** ,je m'en vais pour vous donner un exemple .Souvent dans nos sites web,nous avons besoin d'effectuer certaines taches courantes dans le projet comme mettre en place un système **d'autorisation** ,**d'authentification** ,**envoi des e-mails** ,**généré les couleurs aléatoirement** pour ne citer que ceux-là… En pratique , vous devez vous même réfléchir et écrire tout le code mais en programmation il y a une expression courante qui dit que **'' Eviter de réinventer la roue "** c'est-à-dire au lieu de perdre le temps ou histoire de faire des erreurs , il y a déjà des autres développeurs avant vous qui ont déjà écrit ce même code puis tester et apprécier par d'autres développeurs aussi. Ces codes sont appelés des **librairies** (**bibliothèque** en français ) ou **packages** (**paquets** en français )que vous pouvez utiliser vous aussi dans vos projets d'où votre code **dépend** de ces librairies et c'est **Composer** qui va vous permettre de télécharger ces packages.Si vous avez déjà développé en **JavaScript** , son gestionnaire de dépendance est **npm** ; pour **python** c'est **pip** ou **pip3** et d'ailleurs le fameux **apt** de **linux** pour installer les logiciels est aussi un gestionnaire dépendance … donc chaque langage a son gestionnaire de dépendance.Ce que vous devez retenir **Composer cherche** (installe ) , **supprime** ou **mettre à jour** les dépendances de votre projet .A présent , nous allons d'abord installer **composer** avant de voir comment l'utiliser pour faire **autoloading** de nos classes

**Installation sur windows :**

Rendez – vous sur le site de composer **https://getcomposer.org** puis cliquer sur **Gettting Started** et télécharger **composer-setup.exe** et après le téléchargement **double-cliquer** sur le fichier télécharger puis **suivant suivant** jusqu'à la fin. Pour voir si le **composer** bien est installé , lancer votre **invite de commende** ou **cmd** puis taper simplement : **composer** et si vous obtenez le résultat ,félicitation **composer** est installé sur votre ordinateur

**Installation sur linux :**

Rendez – vous sur le site de composer **https://getcomposer.org** puis cliquer sur **Gettting Started.** Cliquer sur l'option **Installer sur linux** et suivre les étapes d'installation.

Revenons à nos moutons.Notre objectif,c'est d'utiliser **autoloading** de composer pour inclure nos classes,donc la question est la suivante :
Utiliser le fichier **autoload.php** de **composer** qui se trouve dans le dossier **vendor** pour inclure la classe **BankAccount** ( toujours en utilisant le projet du **groupe 10** )
3-Dites nous le lien entre **psr-4** et autoloading de **composer**

Fin du Groupe 10 :

# GROUPE 11 :

## Recherche :

1. Gestion des erreurs
2. set_error_handler()
3. set_exception_handler()
4. try … catch ...finally
5. Throw
6. la classe Exception

## TAF (Travail A Faire ) :

1- En PHP, les erreurs peuvent être gérées à l'aide de la gestion d'erreurs intégrée de PHP, qui utilise des fonctions comme **set_error_handler()** et **set_exception_handler()** pour capturer les erreurs et les **exceptions** dans votre code.
Imaginons que vous avez crée un site web qui calcule ls somme ,différence , quotient et le produit de deux (02) nombres.Vous espérez que l'utilisateur saisissent les nombres afin que votre site puisse les calculer et lui donner le résultat mais celui-ci saisit à la place des nombres les lettres. Qu'allez -vous faire ? La solution est simple ,utilisez alors le bloc **try ...catch** pour définir vos fonctions qui font les opérations des nombres.On dit qu'on lève une **exception** .Si vous jugez que le code que vous voulez implémenter pourrait générer une erreur lors de son utilisation , lèvez tout simplement une exception . Mais souvent on veut exécuter un code après que le bloc **try** ou le bloc **catch** a été exécuté, indépendamment de la présence ou non d'une exception., la clause **finally** peut être utilisée dans ce cas. Voici alors la question : en reprenant le code du **groupe 2** ,utiliser le bloc **try ... catch** ...**finally** au niveau de la méthode **widthdraw ( )**

2- Parallèlement ,faites une recherche sur le mot clé **throw** , les fonctions **set_error_handler()** et **set_exception_handler()** pour gérer aussi les erreurs .
Enfin ,ce que vous devez retenir ,pour bien gérer les erreurs , vous devez faire une recherche sur la classe **Exception** et ses classes filles

Fin du GROUPE 11

# GROUPE 12 : ( Fin )

## Recherche :

1. class_exists()
2. method_exists ( )
3. property_exists ( )

## TAF (Travail A Faire ) :

En PHP, il existe trois fonctions pour vérifier si une **classe**, une **méthode** ou une **propriété** existe : **class_exists()**, **method_exists()** et **property_exists()**.
**1- class_exists( )** vérifie si une classe existe ,utiliser cette méthode pour vérifier si la classe **BankAccount** existe.

2-**method_exists( )** vérifie si une méthode existe dans une **classe** ou dans un **objet,** utiliser cette méthode pour vérifier si la méthode **withdraw** ( ) existe dans une classe **BankAccount** ou un **objet** de la classe **BankAccount**

**3 – property_exists( )** vérifie si une propriété existe dans une **classe** ou dans un **objet**, utiliser cette méthode pour vérifier si la propriété **$balance** existe dans l'objet de la classe **BankAccount**.

Fin du GROUPE 12

# Bonne recherche et suite