

## Sujet:Variante n° 7

groupeN7:-BAJADDA Ahmed

-MAZOUZ Oussama

-JERAF Yassir

## Presentation breve du sujet:

- Le projet consiste à créer un système pour gérer des comptes bancaires. Chaque compte bancaire est caractérisé par un solde, un code sous forme de chaîne de caractères et un titulaire qui est une personne physique. Le solde peut être positif ou négatif et un compte bancaire peut être créé avec un solde initial, un titulaire et un code initial. Les opérations autorisées sur un compte bancaire sont les dépôts, les retraits et la consultation du solde.
- La première étape consiste à créer une classe PersonnePhysique pour représenter les titulaires de compte. Cette classe aura un nom et un prénom et comprendra des méthodes pour consulter ces informations.

- Ensuite, une classe CompteBancaire sera créée pour représenter les comptes bancaires. Cette classe comprendra des constructeurs pour créer des comptes avec ou sans solde initial, un titulaire et un code initial. Elle aura également des méthodes pour modifier le code, consulter le solde, effectuer des dépôts et des retraits.
- La banque souhaite autoriser certains clients à avoir un découvert. Pour cela, une méthode decouvertAutorise sera ajoutée pour spécifier la valeur du découvert autorisé. Si le solde du compte dépasse le découvert autorisé, la méthode de retrait lancera une exception de la classe RetraitInterditException. Le nombre de comptes bancaires existants et le nombre de comptes débiteurs seront également enregistrés.
- Enfin, la banque souhaite gérer des comptes "rémunérés" avec un taux d'intérêt.
  Les titulaires de ces comptes pourront savoir leur solde dans x années et le temps
  nécessaire pour doubler leur solde sans aucune opération effectuée sur le
  compte.

#### Presentation du code:

• Ce programme en C++ simule un système bancaire simple. Il définit trois classes : RetraitInterditException, PersonnePhysique et CompteBancaire.

RetraitInterditException est une classe d'exception qui est levée lorsqu'une opération de retrait ne peut être effectuée en raison d'un solde insuffisant sur le compte.

```
class RetraitInterditException
{
   public:
       RetraitInterditException() {}
       string affiche_err()
       {
        return "Retrait impossible : solde insuffisant.";
       }
   };
```

PersonnePhysique est une classe qui représente une personne. Elle possède deux variables membres privées, nom\_m et prenom\_m, qui sont respectivement le nom et le prénom de la personne. Elle possède également deux fonctions membres publiques, getNom() et getPrenom(), qui renvoient respectivement le nom et le prénom de la personne.

```
class PersonnePhysique
private:
    string nom m;
    string prenom m;
public:
    PersonnePhysique(string nom, string prenom)
        nom m = nom;
        prenom m = prenom;
    string getNom()
        return nom m;
    string getPrenom()
        return prenom m;
```

CompteBancaire est une classe qui représente un compte bancaire. Elle est dérivée de PersonnePhysique. Elle possède quatre variables membres privées, solde\_m, code\_m, decouvertAutorise\_m et deux variables membres statiques nombreComptes\_m et nombreComptesDebiteurs m.

class CompteBancaire:public PersonnePhysique
{
private:
 double solde\_m;
 string code\_m;
 int decouvertAutorise\_m;
 static int nombreComptes\_m;
 static int nombreComptesDebiteurs\_m;

 La variable solde\_m représente le solde actuel du compte, code\_m représente le code utilisé pour accéder au compte, et decouvertAutorise\_m représente le montant autorisé du découvert. nombreComptes\_m est une variable statique qui permet de suivre le nombre total de comptes bancaires créés, et nombreComptesDebiteurs\_m est une variable statique qui permet de suivre le nombre de comptes bancaires ayant un solde négatif. • La classe possède cinq fonctions membres publiques : Un constructeur qui prend quatre arguments : nom, prenom, code et soldeInitial, ainsi qu'un cinquième argument optionnel decouvertAutorise qui est par défaut nul. Ce constructeur initialise les variables membres nom m, prenom m, code m et solde m avec les valeurs passées. Il incrémente également nombre Comptes m de 1 et stocke l'adresse de l'objet créé dans le vecteur List CompteBancaires. Si le solde initial est négatif, il incrémente nombreComptesDebiteurs m de 1. Un deuxième constructeur qui ne prend que deux arguments : nom et prenom. Ce constructeur initialise les variables membres nom met prenom m avec les valeurs passées. Il initialise également les variables membres code m, solde m et decouvertAutorise m avec des valeurs par défaut. Il incrémente nombre Comptes m de 1 et stocke l'adresse de l'objet créé dans le vecteur List CompteBancaires.

```
CompteBancaire(string nom="inconnu", string prenom="inconnu"):PersonnePhysique(nom, prenom)
   solde m = 0;
   code m = "NULL";
   decouvertAutorise m = 0;
   nombreComptes m++;
   List_CompteBancaires.push back(this);// Stocker l'objet créé dans le vecteur 'people'
CompteBancaire (string nom, string prenom, string code, double soldeInitial, int decouvertAutorise=0):PersonnePhysique (nom, prenom
   solde m = soldeInitial;
   code m = code;
   decouvertAutorise m=decouvertAutorise;
   nombreComptes m++;
   List_CompteBancaires.push_back(this);// Stocker l'objet créé dans le vecteur 'people'
   if (soldeInitial < 0)</pre>
        nombreComptesDebiteurs m++;
```

Les fonctions getCode() et setCode(string code), qui renvoient et définissent la variable membre code\_m, respectivement

```
void setCode(string code)
{
    code_m = code;
}
string getCode()
{
    return code_m;
}

double getSolde()
{
    return solde_m;
}
```

La fonction depot(double montant, string code), qui dépose montant sur le compte si le paramètre code correspond à la variable membre code\_m du compte. Si le montant déposé ramène le solde au-dessus de zéro, elle décrémente nombreComptesDebiteurs\_m de 1

```
void depot(double montant, string code)
{
    if (code_m != code)
    {
        cout << "Code incorrect" << endl;
        return;
}

solde_m += montant;

if (solde_m >= 0)
    {
        nombreComptesDebiteurs_m--;
}
```

La fonction retrait (double montant, string code), qui retire montant du compte si le paramètre code correspond à la variable membre code\_m du compte et que le retrait ne fera pas tomber le solde en dessous du découvert autorisé.

```
void retrait(double montant, string code)
    if (code m != code)
        cout << "Code incorrect\n" << endl;
        return;
    if (montant > solde m + decouvertAutorise m)
        throw RetraitInterditException();
        return;
    solde m -= montant;
    if (solde m < 0 )</pre>
        nombreComptesDebiteurs m++;
```

Une fonction calcule le solde d'un compte bancaire après une période de temps donnée avec un taux d'intérêt annuel fixe (TI). La fonction solde\_xannee prend trois paramètres en entrée: S qui représente le solde initial, TI qui est le taux d'intérêt annuel et annee qui est le nombre d'années pour lequel le calcul de solde est souhaité.

La fonction commence par initialiser une variable solde\_var à la valeur initiale de S. Elle utilise ensuite une boucle for pour calculer le solde après chaque année en ajoutant les intérêts annuels. La valeur des intérêts est calculée en multipliant le taux d'intérêt (TI) par le solde actuel et en ajoutant le résultat à la valeur de solde\_var. Après avoir parcouru le nombre d'années spécifié, la fonction retourne le solde final.

```
double solde_xannee(double S,float TI,int annee)
{
    double solde_var=S;
    for (int i=1;i<=annee;i++)
    {
        solde_var=solde_var+TI/100*solde_var;
    }
    return solde_var;
}</pre>
```

- La deuxième fonction doublesolde\_annee calcule le nombre d'années nécessaires pour que le solde d'un compte bancaire double avec un taux d'intérêt annuel fixe (TI). Cette fonction prend également deux paramètres en entrée: S qui est le solde initial et TI qui est le taux d'intérêt annuel.
- La fonction commence par initialiser deux variables: solde\_double est le double du solde initial (2\*S) et solde\_var est initialisé à la valeur initiale de S. Ensuite, elle utilise une boucle while pour augmenter le solde chaque année jusqu'à ce qu'il atteigne la valeur de solde\_double. La valeur de solde\_var est mise à jour à chaque itération en ajoutant les intérêts annuels calculés en multipliant le taux d'intérêt (TI) par le solde actuel et en ajoutant le résultat à la valeur de solde\_var. À chaque itération, l'année est incrémentée jusqu'à ce que la condition soit remplie. Finalement, la fonction retourne le nombre d'années nécessaires pour doubler le solde initial.

```
int doublesolde_annee(double S, float TI)
{
   int annee=0;
   double solde_double=2*S;
   double solde_var=S;
   while (solde_var<solde_double)
   {
      solde_var=solde_var+solde_var*TI/100;
      annee++;
   }
   return annee;
}</pre>
```

```
friend double solde_xannee(double S, float TI, int annee);
friend int doublesolde_annee(double S, float TI);
```

- La première méthode "SetDecouvertAutorise" permet de modifier le montant autorisé du découvert, la seconde méthode "getDecouvertAutorise" permet de récupérer la valeur du découvert autorisé.
- Les deux autres méthodes statiques "getNombreComptes" et "getNombreComptesDebiteurs" permettent de récupérer respectivement le nombre total de comptes bancaires créés et le nombre de comptes bancaires débiteurs. Ces deux méthodes sont statiques, ce qui signifie qu'elles appartiennent à la classe elle-même plutôt qu'à une instance de la classe.

```
void SetDecouvertAutorise(double montant)
   decouvertAutorise_m = montant;
int getDecouvertAutorise()
   return decouvertAutorise m;
static int getNombreComptes()
   return nombreComptes m;
static int getNombreComptesDebiteurs()
   return nombreComptesDebiteurs_m;
```

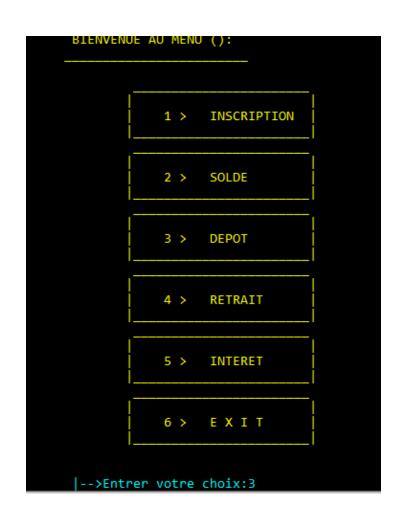
# Pour l'interface de l'application:

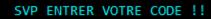


• Le programme principal utilise les classes pour créer plusieurs comptes bancaires et stocke ces comptes dans un vecteur de pointeurs. Ensuite, il affiche un menu à l'utilisateur, permettant de créer un nouveau compte, d'afficher les informations d'un compte existant, de déposer ou de retirer de l'argent, ou de quitter le programme.

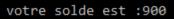
- Le programme utilise des boucles for pour parcourir le vecteur de comptes bancaires et recherche le compte correspondant au code entré par l'utilisateur. Il utilise également une exception personnalisée pour gérer les cas où un retrait est supérieur au solde disponible sur le compte.
- Le programme utilise également la fonction system("cls") pour effacer la console et rendre le programme plus facile à utiliser. Cependant, l'utilisation de la fonction goto peut rendre le code difficile à lire et à comprendre.

### Exemple:





|-->Entrer votre code ici:0000



combien voulez vous deposer:9000

votre solde est :900

combien voulez vous deposer:9000

return to menu?[1 si oui anything else]

Ce code permet de sauvegarder des objets de la classe CompteBancaire dans un fichier texte. On ouvre le fichier en mode écriture, vérifie qu'il a été ouvert correctement, écrit les données de chaque objet dans le fichier en les séparant avec des barres verticales et des retours à la ligne, et enfin on ferme le fichier.

```
// ouverture du fichier en mode écriture
   std::ofstream fichier("ah.txt");
    // vérification que le fichier a été ouvert correctement
   if (!fichier.is open())
                                                                                                  ah.txt - Bloc-notes
        cout << "Erreur : impossible d'ouvrir le fichier" << std::endl;</pre>
                                                                                               Fichier Edition Format Affichage Aide
                                                                                               JACK | HALMES | 900 |
       sauvgarder les objets dans le fichier
    for (int i=0; i<CompteBancaire::List CompteBancaires.size(); i++)</pre>
        fichier <<CompteBancaire::List CompteBancaires[i]->getNom()<<" | ";</pre>
                                                                                               AHMED | BAJADDA | 900 |
        fichier <<CompteBancaire::List CompteBancaires[i]->getPrenom()<<" | ";</pre>
        fichier <<CompteBancaire::List CompteBancaires[i]->getSolde()<<" | \n\n";</pre>
    // fermeture du fichier
   fichier.close();
```

#### **CONCLUSION:**

• En conclusion, notre groupe a travaillé avec diligence pour développer un programme de comptes bancaires efficace et convivial. Nous avons passé des heures à planifier, concevoir et tester chaque fonctionnalité pour nous assurer que le programme répond aux besoins de nos utilisateurs. Grâce à notre collaboration et notre engagement envers l'excellence, nous sommes fiers de présenter un produit de haute qualité qui offrira une expérience de gestion de compte bancaire agréable et facile pour tout utilisateur.