Analyse UML et Codage C++

Compte Bancaire

~ TD5 – Complément C++ ~

Version 1.0 – janvier 2019

Objectif

- Codage C++
  - Flux d'entrée – sortie standard (cin cout)
  - Classe string
  - Héritage
  - Accès protégé
  - Polymorphisme
- UML
  - Diagramme de classe
  - Relations entre classes : Héritage

Conditions de réalisation

Ressources utilisées :

Matériel	**Logiciel**
Un PC sous Linux	NetBeans Modelio

1. LES FLUX D'ENTRÉE - SORTIE STANDARD

La librairie <iostream> offre la possibilité d'écrire sur la console et de lire le clavier en utilisant des flux.

Exemples de flux de sortie :

```
#include <iostream>

using namespace std;

int main(int argc, char** argv)
{
    cout << "Voici un message" << endl;

    return 0;
}
```

Librairie à utiliser pour les flux d'entrée-sortie standard.

Utilise l'espace de nommage **std**
Evite d'écrire **std::cout**.

Flux de sortie vers la console

Remplace **'\n'** du **printf** pour passer à la ligne.

Le flux **cout** permet de visualiser des variables de type simple entier, réel, caractère, chaîne de caractères…

De même, un nouveau type apparaît en C++, le type **string**. Il s'agit d'un objet de type chaîne de caractères avec une gestion de la mémoire et des méthodes de gestion intégrées. Il est également dans l'espace de nommage **std**.

1. *Sous Netbeans, dans un projet nommé **Banque** de type application console C++, complétez le programme principal suivant pour faire apparaître le menu :*

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char** argv)
{
    string menu[] = {"Déposer", "Retirer", "Solde", "Quitter"};
    for(int i= 0 ; i < 4 ; i++)
    {
        cout <<
    }
    cout <<

    return 0;
}
```

```
Sortie - Banque (Run)
1 - Déposer
2 - Retirer
3 - Solde
4 - Quitter

Votre choix :
```

Le flux **cin** permet de lire les données en provenance du clavier. Les variables de type simple entier, réel, caractère, chaîne de caractères sont affectées directement.

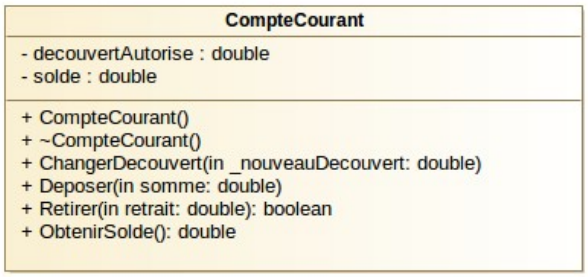
```
int choix;
cin >> choix ;
```

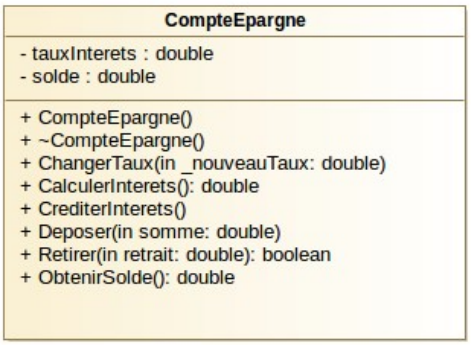
La variable **choix** reçoit ici un entier, car elle est déclarée en tant qu'entier. Si choix avait été déclarée en tant que caractère, le contenu de choix serait un caractère.

À noté, le caractère **&** représentant l'adresse de la variable n'est plus nécessaire comme il l'était avec le **scanf**.

2. NOTION D'HÉRITAGE : LA GÉNÉRALISATION

On donne les deux classes suivantes :





2. *Sous modelio, regroupez dans la classe **CompteBancaire** les éléments communs à ces deux classes. Puis reconstituez un nouveau diagramme de classes représentant la relation entre les classes **CompteBancaire**, **CompteCourant** et **CompteEpargne**. Vous aurez pris soin de supprimer, des deux autres classes, les éléments regroupés dans la classe **CompteBancaire**.*

   Attention de prendre soin à l'accès aux données membres de la classe de base.

3. *Sous Netbeans, dans votre projet **Banque**, réalisez le codage de la classe **CompteBancaire**.*

   - Le constructeur initialise le solde à 0,
   - la méthode **Deposer** ajoute le **montant** au **solde**.
   - La méthode **Retirer** impute la valeur du **retrait** au **solde** si cela est possible et retourne **vrai**, sinon elle retourne **faux**.
   - La méthode **ObtenirSolde** retourne la valeur du **solde**.

4. *Instanciez votre classe **CompteBancaire** dans le programme principal que vous compléterez pour pouvoir tester cette classe. Seuls les flux **cin** et **cout** sont autorisés pour la saisie et l'affichage des valeurs.*

5. *Dans le même projet sous Netbeans, réalisez le codage de la classe **CompteEpargne**. Modifiez le programme principal pour vérifier son fonctionnement.*

6. *Ajoutez dans chaque constructeur et chaque destructeur un message indiquant le passage dans la méthode.*

   *Exemple : **cout << " Constructeur CompteBancaire " << endl ;***

   Relevez l'ordre de l'appel de chacun. Pourquoi en est-il ainsi ?

7. *Procédez de la même manière pour le codage de la classe **CompteCourant**. À la suite de vos tests, vous devez constater que la méthode **Retirer** ne tient pas compte du découvert autorisé.*

   *Surchargez la méthode **Retirer** dans la classe **CompteCourant** pour quelle tienne compte du découvert autorisé. Vérifiez à nouveau le fonctionnement.*

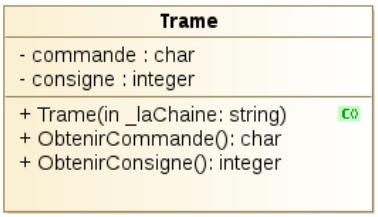
3. GESTION DES CHAÎNES DE CARACTÈRES : std::string

De nombreux challenges ont eu recours au traitement de chaînes de caractères en provenance d'une liaison série. Bien souvent, la trame reçue est constituée d'une **commande** suivie d'une **consigne**. On propose ici l'étude de l'extraction de chacune de ces deux valeurs pour les traiter séparément par la suite.

Pour le robot rover, nous pouvons envisager des trames du style

À 100	Avance vitesse= 100		**D 50**	Tourne à droite vitesse = 50
R 60	Recule vitesse = 60		**G 80**	Tourne à gauche vitesse = 80
S	Arrêt du robot, remarque ici la commande n'est pas suivie d'une consigne			

Chacune de ces trames est une chaîne de caractères composée d'une lettre, d'un espace et éventuellement un chiffre représenté sous forme de caractères.



On propose la modélisation de cette trame sous la forme d'une classe comme le montre la figure ci-contre.

Le constructeur reçoit la chaîne de caractères.

Les deux autres méthodes permettent de récupérer respectivement les valeurs de la commande et de la consigne.

1. Sous NetBeans, créez nouveau projet nommé **TestTrame** de type application console en C++. Ajoutez à ce projet la classe **Trame** en respectant la modélisation proposée.
2. Codez le constructeur, il possède un paramètre de type **std::string**. Son rôle est d'extraire les deux parties de la trame pour affecter les attributs de la classe. Pour une trame sans consigne, celle-ci doit être fixée à 0.

   Pour la commande, rien de compliqué, la classe **std::string** possède **l'opérateur [ ]**, à la manière d'un tableau on accède à chaque caractère de la chaîne. Ici, c'est le premier caractère de la chaîne.

   Pour la consigne, il est nécessaire de rechercher la position du caractère espace dans la chaîne et d'extraire la sous chaîne à la suite contenant les nombres. En effet, certains programmeurs ont envisagé de coder la trame « Avance 100 »… notre classe Trame doit toujours fonctionner.

   Pour vous aider, sur le site developpez.com étudiez les rubriques :

   14.1.4. Accès aux données de la chaîne de caractères
   14.1.5.2. Extraction de données d'une chaîne de caractères
   14.1.7. Recherche dans les chaînes
3. Codez les deux autres méthodes, puis, dans le programme principal, testez votre classe en l'instanciant avec les différents types de paramètres comme le montre le tableau en haut de page et avec des trames du style : « Avance 100 ».

   Tous les affichages utiliseront le flux de sortie standard **cout**.

   Pour convertir une chaîne de caractère ASCII en entier la fonction **atoi()** peut-être envisagée. La conversion **string → tableau de caractères** est réalisée par la méthode **string::c_str()**.