TP Western C++ n°3

Sommaire

Les objectifs de ce tp sont de découvrir la programmation orientée objet en C++.
On désire réaliser un programme C++ permettant d'écrire facilement des histoires de Western.
Dans nos histoires, nous aurons des brigands, des cowboys, des shérifs, des barmen et des dames en détresses ... (à partir d'une idée de Laurent Provot)

Histoire n°3 : Brigands, cowboys et dames en détresses ...

Objectifs

On désire réaliser un programme orienté objet en C++ qui **racontera une histoire dans laquelle une dame mais se fait kidnapper par un brigand avant d'être libérée par un cowboy**.

```
(Lucky Luke) -- Bonjour, je suis le vaillant Lucky Luke et
j'aime le coca-cola
(Jenny) -- Bonjour, je suis Miss Jenny et j'ai une jolie robe
blanche
** Miss Jenny hurle
(Jenny) -- Au secours, je me fais kidnapper !
(Joe) -- Ah ah ! Miss Jenny, tu es mienne désormais !
(Lucky Luke) -- Bonjour, je suis le vaillant Lucky Luke et
j'aime le coca-cola
(Joe) -- Bonjour, je suis Joe le méchant et j'aime le tord-
boyaux.
(Joe) -- J'ai l'air méchant et j'ai déjà kidnappé 1 dames !
(Joe) -- Ma tête est mise à prix 100 $ !
** Le vaillant Lucky Luke tire
(Lucky Luke) -- Prends ça, rascal !
(Joe) -- Damned, je suis fait ! Lucky Luke, tu m'as eu !
** Le vaillant Lucky Luke libère Miss Jenny
(Jenny) -- Merci Lucky Luke, je suis enfin libre !
```

La programmation orientée objet en action

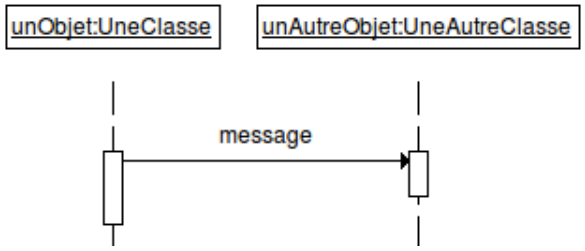
Rappels

La programmation orientée objet consiste à **définir des objets logiciels et à les faire interagir entre eux**.

Notion de messages

Un objet est une structure de données encapsulées qui répond à un **ensemble de messages**. Cette **structure de données (ses attributs) définit son état** tandis que l'**ensemble des messages (ses méthodes) décrit son comportement**.

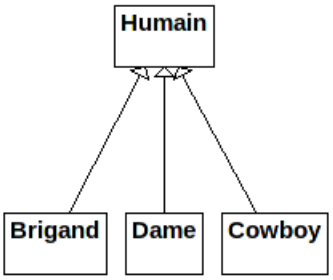
L'ensemble des messages forme ce que l'on appelle l'**interface de l'objet**. Les objets interagissent entre eux en **s'échangeant des messages**.



La réponse à la réception d'un message par un objet est appelée **une méthode**. Une **méthode est donc la mise en oeuvre du message** : elle décrit la réponse qui doit être donnée au message.

Exemple détaillé : des brigands kidnappent des dames et des cowboy les libèrent

*Rappel : Les **brigands**, les **dames** et les **cowboys** sont tous des **humains**.*



Une **dame** peut **se faire kidnapper** (auquel cas elle **hurle**), **se faire libérer** par un **cowboy** (elle **remercie** alors le **héros** qui l'a libérée).

La nouvelle classe `Dame` sera alors la suivante :

Dame
- string couleurRobe - string etat
+ Dame(const string nom = "", const string boissonFavorite = "lait", const string couleurRobe = "blanche") + string getNom() + string getEtat() + void sePresente() + void seFaitKidnapper() + void seFaitLiberer(Cowboy & cowboy) + void changeDeRobe(const string couleurRobe) - void hurle() - void remercie(const Cowboy & heros)

Un **brigand** peut **kidnapper** une **dame** (auquel cas, il s'exclame « Ah ah ! (nom de la dame), tu es mienne désormais ! ». Il peut également **se faire emprisonner** par un **cowboy** (il s'écrit alors « Damned, je suis fait ! (nom du cowboy), tu m'as eu ! ». On dispose également d'une fonction pour connaître la récompense obtenue en cas de capture.

La nouvelle classe `Brigand` sera alors la suivante :

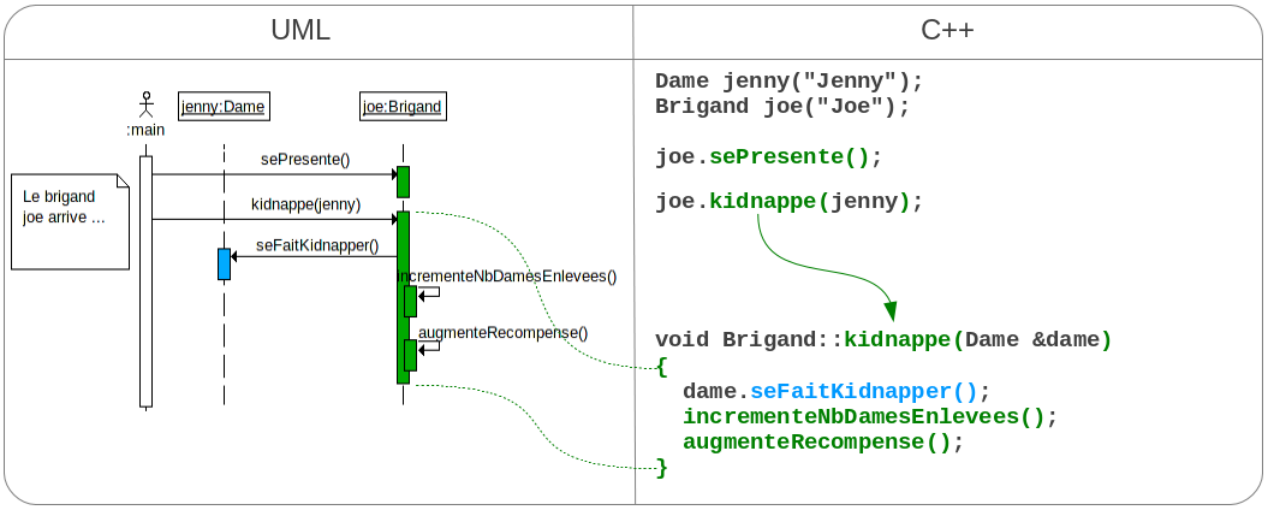
Brigand
- string comportement - int nbDamesEnlevees - int recompense - bool enPrison
+ Brigand(const string nom = "", const string boissonFavorite = "tord-boyaux", const string comportement = "méchant") + string getComportement() + int getNbDamesEnlevees() + int getRecompense() + void sePresente() + void kidnappe(Dame & dame) + void seFaitEmprisonne(Cowboy & cowboy) + void augmenteRecompense(const int prix = 100) + void diminueRecompense(const int prix = 100) + bool estEnPrison()

Un **cowboy** peut **tirer** sur un **brigand**. Un commentaire indique alors « Le (adjectif) (nom) tire sur (nom du méchant). PAN ! » et le cowboy s'exclame « Prend ça, rascal ! ». Il peut également **libérer** une **dame**.

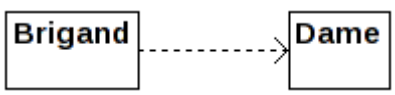
La nouvelle classe `Cowboy` sera alors la suivante :

Cowboy
- int popularite - string qualite
+ Cowboy(const string nom = "", const string boissonFavorite = "whisky", const string qualite = "vaillant") + int getPopularite() + void setPopularite(const int popularite) + string getQualite() + void setQualite(const string qualite) + void sePresente() + void incrementePopularite() + void decrementePopularite() + void tire(const Brigand & brigand) + void emprisonne(Brigand & brigand) + void libere(Dame & dame)

Prenons la situation où un **brigand kidnappe une dame** : les objets `joe` et `jenny` doivent interagir entre eux



Ici l'objet joe de type `Brigand` "utilise" les services de l'objet jenny de type `Dame` par l'intermédiaire de la méthode `seFaitKidnapper()`. Une relation d'utilisation est une dépendance entre classes. La plupart du temps, les dépendances servent à montrer qu'une classe utilise une autre comme argument dans la signature d'une méthode. On parle aussi de lien temporaire car il ne dure que le temps de l'exécution de la méthode. Cela peut être aussi le cas d'un objet local à une méthode.



Une dépendance s'illustre par une flèche en pointillée dans un diagramme de classes en UML.

Les dépendances entre classes

Ces dépendances vont avoir des répercussions sur le code des classes.

Par exemple, lorsqu'on va déclarer la méthode `kidnappe()` dans la classe `Brigand`, il va falloir écrire :

```
class Brigand : public Humain
{
   ...
   void kidnappe(Dame &dame);
   ...
};
```

brigand.h

On va obtenir une erreur à la compilation car le type `Dame` n'a pas été déclaré :

```
erreur: 'Dame' has not been declared
```

Pour corriger cette erreur, il suffit d'indiquer au compilateur que le type **`Dame` est une classe** puisque c'est ce qu'il veut savoir :

```
class Dame; // je ''déclare'' : Dame est une classe !

class Brigand : public Humain
{
   ...
   void kidnappe(Dame &dame);
   ...
};
```

brigand.h

La définition de la méthode `kidnappe()` entraîne de nouveaux problèmes :

```
...
void Brigand::kidnappe(Dame &dame) // <- problème : appel du constructeur Dame ?!
{
   dame.seFaitKidnapper(); // <- problème : appel de seFaitKidnapper() ?!
   nbDamesEnlevees++;
   augmenteRecompense();
   cout << "(" << nom << ") -- " << "Ah ah ! " << dame.getNom() << ", tu es mienne désormais
        !" << endl; // <- problème : appel de getNom() ?!
}
...
```

brigand.cpp

On va obtenir des erreurs à la compilation car celui-ci ne connaît pas "suffisamment" le type `Dame` :

```
erreur: invalid use of incomplete type 'struct Dame'
...
```

Pour corriger ces erreurs, il suffit d'**inclure la déclaration (complète) de la classe `Dame`** qui est contenue dans le **fichier d'en-tête** (*header*) `Dame.h` :

```
#include "brigand.h"
#include "dame.h" // accès à la déclaration complète de la classe Dame
...
void Brigand::kidnappe(Dame &dame)
```



```
{
   dame.seFaitKidnapper();
   nbDamesEnlevees++;
   augmenteRecompense();
   cout << "(" << nom << ") -- " << "Ah ah ! " << dame.getNom() << ", tu es mienne désormais
        !" << endl;
}
...
```

brigand.cpp

La classe Brigand

Maintenant, on déclare la classe `Brigand` :

```
#ifndef BRIGAND_H
#define BRIGAND_H

#include <iostream>
#include <string>
#include "humain.h"
using namespace std;

class Dame; // nécessaire pour indiquer que Dame est une classe (1)
class Cowboy; // nécessaire pour indiquer que Cowboy est une classe (2)

class Brigand : public Humain
{
   public:
      // Constructeurs et Destructeur
      Brigand(const string nom="", const string boissonFavorite="tord-boyaux", const string
          comportement="méchant");
      // Accesseurs
      string getComportement() const;
      int getNbDamesEnlevees() const;
      int getRecompense() const;
      // Services
      void sePresente() const;
      void kidnappe(Dame &dame); // Dame ? -> (1)
      void seFaitEmprisonne(Cowboy &cowboy); // Cowboy ? -> (2)
      void augmenteRecompense(const int prix=100);
      void diminueRecompense(const int prix=100);
      bool estEnPrison() const;

   private:
      string comportement;
      int nbDamesEnlevees;
      int recompense;
      bool enPrison;
};

#endif // BRIGAND_H
```

brigand.h

Puis, on définit les méthodes de la classe `Brigand` :

```
#include "brigand.h"
#include "dame.h" // nécessaire pour accéder à la déclaration de la classe Dame
#include "cowboy.h" // nécessaire pour accéder à la déclaration de la classe Cowboy

// Constructeurs et Destructeur
Brigand::Brigand(const string nom/*=""*/, const string boissonFavorite/*="tord-boyaux"*/,
    const string comportement/*="méchant"*/) : Humain(nom, boissonFavorite), comportement(
    comportement), nbDamesEnlevees(0), recompense(0), enPrison(false)
{
}

// Accesseurs
string Brigand::getComportement() const
{
   return comportement;
}

int Brigand::getNbDamesEnlevees() const
{
   return nbDamesEnlevees;
}

int Brigand::getRecompense() const
{
   return recompense;
}

// Services
void Brigand::sePresente() const
{
   cout << "(" << nom << ") -- " << "Bonjour, je suis " << getNom() << " le " <<
       getComportement() << " et j'aime le " << getBoissonFavorite() << "." << endl;
   // TODO : présentation détaillée
}

void Brigand::kidnappe(Dame &dame)
{
   if(!estEnPrison())
   {
      dame.seFaitKidnapper();
      nbDamesEnlevees++;
      augmenteRecompense();
      cout << "(" << nom << ") -- " << "Ah ah ! " << dame.getNom() << ", tu es mienne
          désormais !" << endl;
   }
}

void Brigand::seFaitEmprisonne(Cowboy &cowboy)
{
   if(!estEnPrison())
   {
      enPrison = true;
```



```
      cout << "(" << nom << ") -- " << "Damned, je suis fait ! " << cowboy.getNom() << ", tu
          m'as eu !" << endl;
   }
}

void Brigand::augmenteRecompense(const int prix/*=100*/)
{
   if(prix > 0)
      this->recompense += prix;
}

void Brigand::diminueRecompense(const int prix/*=100*/)
{
   if(prix > 0 && prix <= recompense)
      this->recompense -= prix;
}

bool Brigand::estEnPrison() const
{
   return enPrison;
}
```

brigand.cpp

Travail demandé

Un **brigand** se présentera en parlant de son look, du nombre de dames qu'il a enlevé et de la récompense offerte pour sa capture. Par exemple le brigand Bob se se présentera par exemple de la manière suivante :

```
(Bob) -- Bonjour, je suis Bob le méchant et j'aime le tord-boyaux.
(Bob) -- J'ai l'air méchant et j'ai déjà kidnappé 5 dames !
(Bob) -- Ma tête est mise à prix 100 $ !
```

Question 1. Compléter les classes `Brigand`, `Cowboy` et `Dame` afin d'assurer l'exécution du programme de test ci-dessous.

```
#include <iostream>

using namespace std;

#include "cowboy.h"
#include "dame.h"
#include "brigand.h"

/* TP Western C++ n°3 : Brigands, cowboys et dames en détresse */

int main()
{
   Cowboy lucky("Lucky Luke", "coca-cola");
   Dame jenny("Jenny");
   Brigand joe("Joe");
```



```
    // 1. La rencontre ...
    lucky.sePresente();
    jenny.sePresente();

    // 2. Mais un brigand arrive ...
    joe.sePresente();
    joe.kidnappe(jenny);

    // 3. Heureusement le cowboy s'interpose ...
    lucky.sePresente();
    joe.sePresente();
    lucky.tire(joe);
    lucky.emprisonne(joe);
    lucky.libere(jenny);

    return 0;
}
```

histoire-3.cpp

```
$ ./histoire-3
(Lucky Luke) -- Bonjour, je suis le vaillant Lucky Luke et j'aime le coca-cola
(Jenny) -- Bonjour, je suis Miss Jenny et j'ai une jolie robe blanche
(Joe) -- Bonjour, je suis Joe le méchant et j'aime le tord-boyaux.
** Miss Jenny hurle
(Jenny) -- Au secours, je me fais kidnapper !
(Joe) -- Ah ah ! Miss Jenny, tu es mienne désormais !
(Lucky Luke) -- Bonjour, je suis le vaillant Lucky Luke et j'aime le coca-cola
(Joe) -- Bonjour, je suis Joe le méchant et j'aime le tord-boyaux.
(Joe) -- J'ai l'air méchant et j'ai déjà kidnappé 1 dames !
(Joe) -- Ma tête est mise à prix 100 $ !
** Le vaillant Lucky Luke tire sur Joe
(Lucky Luke) -- Prends ça, rascal !
(Joe) -- Damned, je suis fait ! Lucky Luke, tu m'as eu !
** Le vaillant Lucky Luke libère Miss Jenny
(Jenny) -- Merci Lucky Luke, je suis enfin libre !
$
```

