Design pattern Commande en C++

Hyacinthe MENIET

3 août 2019

Le design pattern Commande (**Command**) permet de gérer des appels de méthodes en tant qu'objets. Qu'est-ce que cela signifie ? Cela signifie simplement que le design pattern Commande offre la possibilité de paramétrer un objet en fonction de l'action à effectuer. Ce type de paramétrage est déjà possible dans la majorité des langages procéduraux, grâce aux fonctions de callback notamment. Le design pattern Commande est, entre autres, la manière orientée objet d'adresser ce genre de problématiques.
Note : J'utilise sciemment l'appellation anglaise *Design pattern* au lieu des françaises *Patron de conception* et *Motif de conception*. Ceci car c'est généralement sous ce nom que ces techniques de programmation apparaissent dans la littérature technique, y compris francophone.

1. Pré-requis

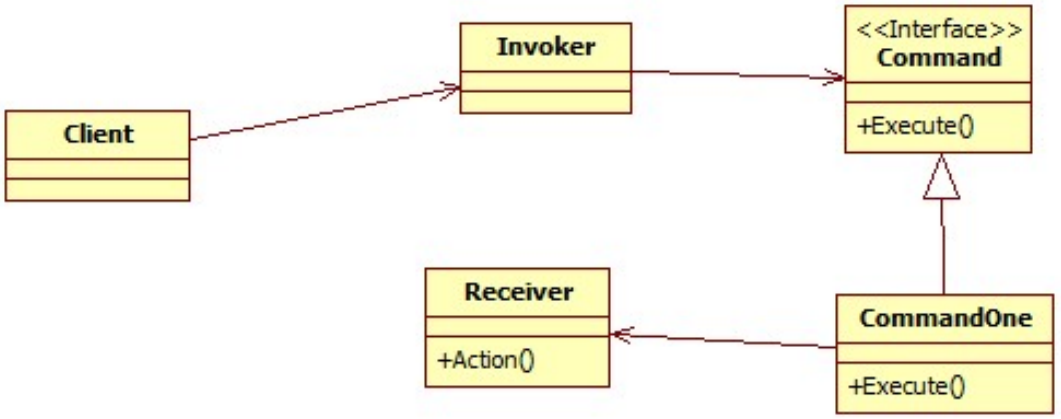
- Vous êtes familier de la programmation orientée objet
- Vous savez ce qu'est un design pattern
- Vous maîtrisez la syntaxe C++
- Vous êtes familier de la compilation séparée
- Vous êtes familier des bibliothèques C++ (ex : STL)
- Vous disposez d'un environnement de développement pour C++, à minima 1 éditeur de texte (ex : vim, emacs, notepad ...) et un compilateur (ex : g++, cygwin, mingw ...).

Note : Pour limiter les adhérences techniques j'ai développé le code ci-dessous à l'aide de vim et g++ sous Linux. Cependant, si vous êtes plus à l'aise avec, sachez que Microsoft Visual C++ Express (Windows) ou Code : :Blocks (Windows, Linux et OSX) feront également l'affaire.

2. Vue d'ensemble

2.1 Vue d'ensemble du design pattern Commande

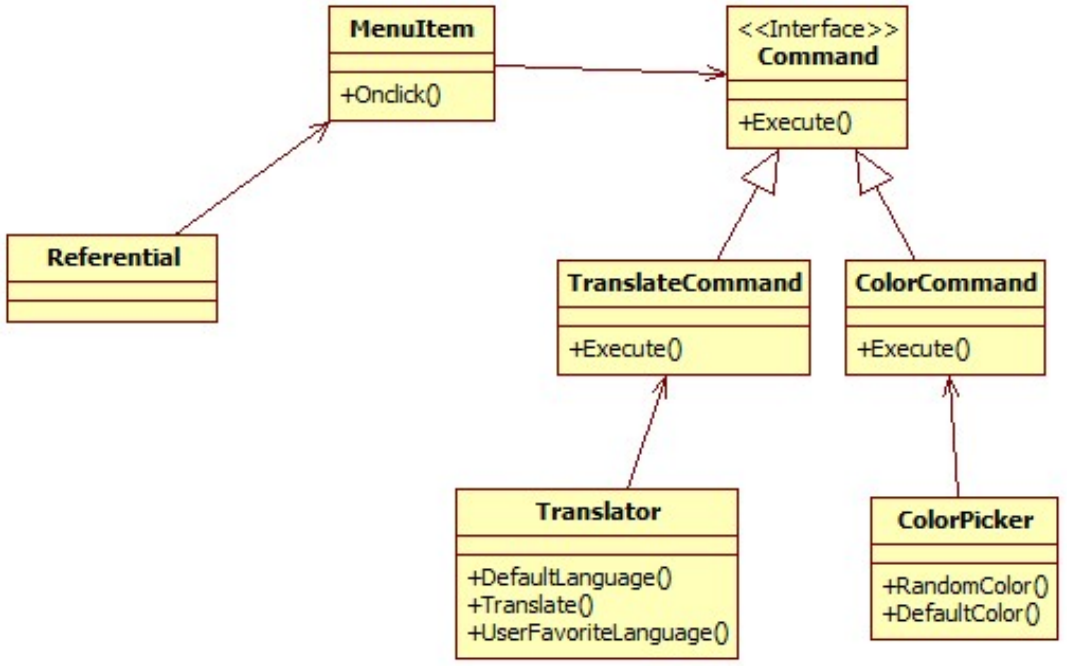
Comme vous l'avez compris, le design pattern Commande découple le programme appelant de l'objet qui réalise l'action demandée. Ce design pattern est particulièrement indiqué lorsque vous avez besoin d'adresser des requêtes à des objets sans rien connaître, ni de la méthode à appeler, ni de la nature de l'objet qui contient cette méthode. Par exemple, dans le cas d'une application graphique, un menu est un composant générique qui permet d'associer un clic utilisateur à une action. Dans ce cas, le Framework graphique (ex : QT, Gtk, Swing ...), n'a aucune connaissance du métier des actions à effectuer, il se contente d'appeler la bonne méthode suite au clic de l'utilisateur. C'est le principe du design pattern Commande.
Le design pattern Commande est construit à partir de plusieurs composants :



— **Command :** L'abstraction des commandes à exécuter.
— **CommandOne :** Une implémentation de Command, elle encapsule le code métier.
— **Receiver :** Il contient le code métier à exécuter.
— **Invoker :** Le composant générique qui reçoit la requête de l'utilisateur.
— **Client :** Le programme principal.

2.2 Vue d'ensemble de l'article

Pour illustrer le design pattern Commande je vais montrer comment résoudre simplement ce problème un peu simpliste mais qui vous servira de base pour résoudre des problèmes plus pointus : Nous devons réaliser un référentiel de services. C'est à dire un annuaire de services qui exécute un service (ex : traduction, gestion des couleurs) lorsqu'on appelle une de ses pages. Le référentiel n'a aucune connaissance métier du service à exécuter, il se contente d'appeler celui qui est paramétré. La solution proposée correspond au schéma UML suivant :



Vous pouvez également télécharger le source C++ du tutoriel sur le design pattern Commande.

3. Les commandes

3.1 L'abstraction

Créez l'interface Command avec la méthode virtuelle pure Execute. Dans mon cas cela correspond aux en-têtes Command.h :

```
/*
* Command.h
*
*       Author: Hyacinthe MENIET
*/
#ifndef COMMAND_H_
#define COMMAND_H_
namespace dotmyself {
/*
* The Command interface.
*/
class Command {
public:
virtual ~Command();
virtual void Execute() = 0;
protected:
Command();
```

```
};
}
#endif /* COMMAND_H_ */
```

Et au source Command.cpp :

```
/*
* Command.cpp
*
*       Author: Hyacinthe MENIET
*/
#include "Command.h"
namespace dotmyself {
Command::~Command() {}
Command::Command() {}
}
```

3.2 Les implémentations

Implémentez la classe Command pour un service de traduction. Dans mon cas cela correspond aux en-têtes TranslateCommand.h :

```
/*
* TranslateCommand.h
*
*       Author: Hyacinthe MENIET
*/
#ifndef TRANSLATECOMMAND_H_
#define TRANSLATECOMMAND_H_
#include "Command.h"
#include "Translator.h"
namespace dotmyself {
/*
* Implementation of the Command interface.
* It encapsulates a request to the Translator.
*/
class TranslateCommand: public dotmyself::Command {
public:
typedef void (Translator::*Action)();
TranslateCommand(Translator *, Action);
virtual ~TranslateCommand();
void Execute();
private:
Translator *receiver;
Action method;
};
}
#endif /* TRANSLATECOMMAND_H_ */
```

Et au source TranslateCommand.cpp :

```
/*
* TranslateCommand.cpp
*
*       Author: Hyacinthe MENIET
*/
#include "TranslateCommand.h"
namespace dotmyself {
TranslateCommand::TranslateCommand(Translator *t, Action a) {
this->method = a;
this->receiver = t;
}
TranslateCommand::~TranslateCommand() {}
void TranslateCommand::Execute() {
(receiver->*method )();
}
}
```

Implémentez également la classe Command pour un service de gestion des couleurs. Dans mon cas cela correspond aux en-têtes ColorCommand.h :

```
/*
* ColorCommand.h
*
*       Author: Hyacinthe MENIET
*/
#ifndef COLORCOMMAND_H_
#define COLORCOMMAND_H_
#include "Command.h"
#include "ColorPicker.h"
namespace dotmyself {
/*
* Implementation of the Command interface.
* It encapsulates a request to the ColorPicker.
*/
class ColorCommand: public dotmyself::Command {
public:
typedef void (ColorPicker:: *Action)();
ColorCommand(ColorPicker *, Action);
virtual ~ColorCommand();
void Execute();
private:
ColorPicker *receiver;
Action method;
};
}
#endif /* COLORCOMMAND_H_ */
```

Et au source ColorCommand.cpp :

```
/*
* ColorCommand.cpp
*
*       Author: Hyacinthe MENIET
*/
#include "ColorCommand.h"
namespace dotmyself {
ColorCommand::ColorCommand(ColorPicker *c, Action a) {
this->method = a;
this->receiver = c;
}
ColorCommand::~ColorCommand() {}
void ColorCommand::Execute() {
(receiver->*method )();
}
}
```

4. Les récepteurs

Créez un récepteur dédié au service de traduction. Dans mon cas cela correspond aux en-têtes Translator.h :

```
/*
* Translator.h
*
*       Author: Hyacinthe MENIET
*/
#ifndef TRANSLATOR_H_
#define TRANSLATOR_H_
namespace dotmyself {
/*
* Exposes a Translator features.
* It is acted upon by each request to TranslateCommand.
*/
class Translator {
public:
Translator();
virtual ~Translator();
void Translate();
void DefaultLanguage();
void UserFavoriteLanguage();
};
}
#endif /* TRANSLATOR_H_ */
```

Et au source Translator.cpp :

```
/*
* Translator.cpp
*
*      Author: Hyacinthe MENIET
*/
#include <iostream>
#include "Translator.h"
using namespace std;
namespace dotmyself {
Translator::Translator() {}
Translator::~Translator() {}
void Translator::Translate() {
cout << "[Translator] Called Translate" << endl;
}
void Translator::DefaultLanguage() {
cout << "[Translator] Called DefaultLanguage" << endl;
}
void Translator::UserFavoriteLanguage() {
cout << "[Translator] Called UserFavoriteLanguage" << endl;
}
}
```

Créez également un récepteur dédié au service de gestion des couleurs. Dans mon cas cela correspond aux en-têtes ColorPicker.h :

```
/*
* ColorPicker.h
*
*      Author: Hyacinthe MENIET
*/
#ifndef COLORPICKER_H_
#define COLORPICKER_H_
namespace dotmyself {
/*
* Exposes a Color Picker features.
* It is acted upon by each request to ColorCommand.
*/
class ColorPicker {
public:
ColorPicker();
virtual ~ColorPicker();
void RandomColor();
void DefaultColor();
};
}
#endif /* COLORPICKER_H_ */
```

Et au source ColorPicker.cpp :

```
/*
* ColorPicker.cpp
*
*       Author: Hyacinthe MENIET
*/
#include <iostream>
#include "ColorPicker.h"
using namespace std;
namespace dotmyself {
ColorPicker::ColorPicker() {}
ColorPicker::~ColorPicker() {}
void ColorPicker::RandomColor() {
cout << "[ColorPicker] Called RandomColor" << endl;
}
void ColorPicker::DefaultColor() {
cout << "[ColorPicker] Called DefaultColor" << endl;
}
}
```

5. L'invocateur

J'ai choisis comme Invocateur, le composant générique MenuItem qui associe un clic utilisateur sur un item de menu à une Commande. Cela correspond aux en-têtes MenuItem.h :

```
/*
* MenuItem.h
*
*       Author: Hyacinthe MENIET
*/
#ifndef MENUITEM_H_
#define MENUITEM_H_
#include <string>
#include "Command.h"
using namespace std;
namespace dotmyself {
/*
* An item of Menu.
* It contains a reference to the Command to perform at each click
    .
*/
class MenuItem {
private:
string name;
Command *cmd;
public:
```

```
MenuItem(string ,Command *);
virtual ~MenuItem();
void Onclick();
};
}
#endif /* MENUITEM_H_ */
```

Et au source MenuItem.cpp :

```
/*
* Invoker.cpp
*
*       Author: Hyacinthe MENIET
*/
#include <iostream>
#include <string>
#include "MenuItem.h"
using namespace std;
namespace dotmyself {
MenuItem::MenuItem(string n, Command *c) {
this->name = n;
this->cmd = c;
}
MenuItem::~MenuItem() {}
void MenuItem::Onclick() {
cout << "The user has clicked on the menu " << this->name << " :"
    << endl;
cmd->Execute();
}
}
```

6. Le référentiel

Terminez par le référentiel qui est le programme principal. Dans mon cas cela correspond au source Referential.cpp :

```
//============================================================================

// Name        : Referential.cpp
// Author      : Hyacinthe MENIET
// Description : The registry of services that expose services
   using menus
//============================================================================

#include <iostream>
#include "dotmyself/TranslateCommand.h"
#include "dotmyself/Translator.h"
#include "dotmyself/ColorCommand.h"
```

```
#include "dotmyself/ColorPicker.h"
#include "dotmyself/MenuItem.h"
using namespace std;
using namespace dotmyself;
int main() {
TranslateCommand tt(new Translator, &Translator::Translate);
ColorCommand cr(new ColorPicker, &ColorPicker::RandomColor);
TranslateCommand td(new Translator, &Translator::DefaultLanguage)
    ;
TranslateCommand tu(new Translator, &Translator::
    UserFavoriteLanguage);
ColorCommand cd(new ColorPicker, &ColorPicker::DefaultColor);
MenuItem *RegisteredUsersMenu[] =
{
new MenuItem("Translate",&tt),
new MenuItem("Random color",&cr),
new MenuItem("Default language",&td),
new MenuItem("Favorite Language",&tt),
new MenuItem("Default color",&tt),
};
MenuItem *BasicMenu[] =
{
new MenuItem("Translate",&tt),
new MenuItem("Random color",&cr)
};
// Demo
cout << "Simulating users clicks on menus ..." << endl << endl;
RegisteredUsersMenu[3]->Onclick();
BasicMenu[1]->Onclick();
return 0;
}
```

7. Compilation et exécution

Pour ceux qui ne disposent pas d'environnement de développement intégré et ont téléchargé le source ci-dessus, les lignes suivantes vous permettront de compiler le projet. Ces commandes sont à entrer dans un terminal Linux et supposent que vous ayez le compilateur g++ installé sur votre distribution.
Compilation des classes :

```
$ mkdir -p target/classes/dotmyself target/executable
$ g++ -Wall -c -o"target/classes/dotmyself/ColorCommand.o" "src/
    main/cpp/dotmyself/ColorCommand.cpp"
$ g++ -Wall -c -o"target/classes/dotmyself/ColorPicker.o" "src/
    main/cpp/dotmyself/ColorPicker.cpp"
$ g++ -Wall -c -o"target/classes/dotmyself/Command.o" "src/main/
    cpp/dotmyself/Command.cpp"
```

```
$ g++ -Wall -c -o"target/classes/dotmyself/MenuItem.o" "src/main/
    cpp/dotmyself/MenuItem.cpp"
$ g++ -Wall -c -o"target/classes/dotmyself/TranslateCommand.o" "
    src/main/cpp/dotmyself/TranslateCommand.cpp"
$ g++ -Wall -c -o"target/classes/dotmyself/Translator.o" "src/
    main/cpp/dotmyself/Translator.cpp"
$ g++ -Wall -c -o"target/classes/Referential.o" "src/main/cpp/
    Referential.cpp"
```

Quelques explications :

- `src/main/cpp` est le dossier qui contient les sources
- `target/classes` est le dossier qui contient les résultats des compilations
- `target/executable` est le dossier qui contiendra l'exécutable

Génération de l'exécutable :

```
$ g++ -o"target/executable/Referential" ./target/classes/
    dotmyself/ColorCommand.o ./target/classes/dotmyself/
    ColorPicker.o ./target/classes/dotmyself/Command.o ./target/
    classes/dotmyself/MenuItem.o ./target/classes/dotmyself/
    TranslateCommand.o ./target/classes/dotmyself/Translator.o ./
    target/classes/Referential.o
```