READYDEV

GitHub | LinkedIn | SiteWeb
Par Gérard KESSE

Plateforme de Développement Continu

Comprendre la Théorie pour mieux Pratiquer
Sciences de l'Ingénieur
Cours - Tutoriels

Patrons de conception

Apprendre les patrons de conception avec le C++

J'aime, Je partage
Montez en Compétences

READYDEV

GitHub | LinkedIn | SiteWeb
Par Gérard KESSE

Auteur

Je suis **Gérard KESSE**,
Ingénieur en Développement Informatique C/C++/Qt,
Avec à la fois des compétences en Systèmes Embarqués et en Robotique.

Formé à Polytech'Montpellier, Je suis un professionnel de conception de projets logiciel applicatif ou embarqué dans les secteurs de l'Aéronautique, de la Robotique, des Drones et de la Vision par Ordinateur. Aussi, Je reste ouvert à d'autres types de secteurs tels que l'Énergie et les Finances.

Les Sciences de l'Ingénieur sont au cœur du métier d'ingénieur. Sur le site **ReadyDev**, la Plateforme de Développement Continu, dont j'en suis le concepteur, vous trouverez des cours et des tutoriels adaptés aux sciences de l'ingénieur.

J'aime, Je partage.

Gérard KESSE

GitHub | LinkedIn | SiteWeb

Sommaire

Introduction

Le **C++** est un langage de programmation orienté objet. Les **patrons de conception** sont des modèles de conception basés sur la programmation orientée objet. Le but de ce tutoriel est de vous apprendre les **patrons de conception** en **C++**.
Produit par **Gérard KESSE**.

Installation sous Windows

Le but de cette section est de vous présenter les différents outils nécessaires pour apprendre les **Patrons de Conception** avec le **C++**, sous Windows.
Produit par **Gérard KESSE**.

Dans ce tutoriel, nous utiliserons, sous Windows :
- MinGW, comme compilateur.
- Notepad++, comme éditeur de texte.

Télécharger MinGW
http://www.mingw.org/

Télécharger Notepad++
https://notepad-plus-plus.org/fr/

Patrons de création

Patron Singleton (Singleton Pattern)

Le but de cette section est de vous apprendre à implémenter le **patron singleton** avec le C++.
Produit par **Gérard KESSE**.

Le Patron Singleton permet de restreindre l'instanciation d'une classe à un seul objet.

Programme principal

Programme principal (main.cpp)

```
//=============================================
#include "GSingleton.h"
//=============================================
int main(int argc, char** argv) {
    cout << "\n### Patron Singleton\n\n";
    string lData;
    GSingleton::Instance()->showData();
    lData = GSingleton::Instance()->getData();
    cout << "Lecture: " << lData << "\n";
    GSingleton::Instance()->setData("Bonjour tout le monde");
    GSingleton::Instance()->showData();
    lData = GSingleton::Instance()->getData();
    cout << "Lecture: " << lData << "\n";
    return 0;
}
//=============================================
```

Singleton

Déclarer le singleton (GSingleton.h)

```
//================================================
#ifndef _GSingleton_
#define _GSingleton_
//================================================
#include <iostream>
#include <string>
//================================================
using namespace std;
//================================================
class GSingleton {
private:
    GSingleton();

public:
    ~GSingleton();

public:
    static GSingleton* Instance();
    void setData(const string& data);
    string getData();
    void showData();

private:
    static GSingleton* m_instance;
    string m_data;
};
//================================================
#endif
//================================================
```

Définir le singleton (GSingleton.cpp)

```
//=============================================
#include "GSingleton.h"
//=============================================
GSingleton* GSingleton::m_instance = 0;
//=============================================
GSingleton::GSingleton() {
    m_data = "_NONE_";
}
//=============================================
GSingleton::~GSingleton() {

}
//=============================================
GSingleton* GSingleton::Instance() {
    if(m_instance == 0) {
        m_instance = new GSingleton;
    }
    return m_instance;
}
//=============================================
void GSingleton::setData(const string& data) {
    m_data = data;
}
//=============================================
string GSingleton::getData() {
    return m_data;
}
//=============================================
void GSingleton::showData() {
    cout << "Donnee: " << m_data << "\n";
}
//=============================================
```

Résultat

Patrons de comportement

Patron Stratégie (Strategy Pattern)

Le but de cette section est de vous apprendre à implémenter le **patron stratégie** avec le C++.
Produit par **Gérard KESSE**.

Le Patron Stratégie permet de sélectionner dynamiquement des algorithmes.

Programme principal

Programme principal (main.cpp)

```
//================================================
#include "GDatabase.h"
#include "GConfig.h"
//================================================
int main(int argc, char** argv) {
    cout << "\n### Patron Strategie\n\n";
    GConfig::Instance()->setData("DATABASE", "SQLITE");
    GDatabase::Instance()->open();
    GConfig::Instance()->setData("DATABASE", "MYSQL");
    GDatabase::Instance()->open();
    return 0;
}
//================================================
```

Gestionnaire de configuration

Déclarer le gestionnaire de configuration (GConfig.h)

```
//=============================================
#ifndef _GConfig_
#define _GConfig_
//=============================================
#include <iostream>
#include <string>
#include <map>
//=============================================
using namespace std;
//=============================================
class GConfig {
private:
    GConfig();

public:
    ~GConfig();

public:
    static GConfig* Instance();
    void setData(const string& key, const string& value);
    string getData(const string& key);

private:
    static GConfig* m_instance;
    map<string, string> m_dataMap;
};
//=============================================
#endif
//=============================================
```

Définir le gestionnaire de configuration (GConfig.cpp)

```
//=============================================
#include "GConfig.h"
//=============================================
GConfig* GConfig::m_instance = 0;
//=============================================
GConfig::GConfig() {

}
//=============================================
GConfig::~GConfig() {

}
//=============================================
GConfig* GConfig::Instance() {
    if(m_instance == 0) {
        m_instance = new GConfig;
    }
    return m_instance;
}
//=============================================
void GConfig::setData(const string& key, const string& value) {
    m_dataMap[key] = value;
}
//=============================================
string GConfig::getData(const string& key) {
    return m_dataMap[key];
}
//=============================================
```

Gestionnaire de base de données

Déclarer le gestionnaire de base de données (GDatabase.h)

```
//=============================================
#ifndef _GDatabase_
#define _GDatabase_
//=============================================
#include <iostream>
#include <string>
//=============================================
using namespace std;
//=============================================
class GDatabase {
public:
    GDatabase();
    virtual ~GDatabase();

public:
    static GDatabase* Instance();
    virtual void open() = 0;

private:
    static GDatabase* m_instance;
};
//=============================================
#endif
//=============================================
```

Définir le gestionnaire de base de données (GDatabase.cpp)

```
//=============================================
#include "GDatabase.h"
#include "GDatabaseSQLite.h"
#include "GDatabaseMySQL.h"
#include "GConfig.h"
//=============================================
GDatabase* GDatabase::m_instance = 0;
//=============================================
GDatabase::GDatabase() {

}
//=============================================
GDatabase::~GDatabase() {

}
//=============================================
GDatabase* GDatabase::Instance() {
    string lDatabase = GConfig::Instance()->getData("DATABASE");
    if(lDatabase == "SQLITE") return GDatabaseSQLite::Instance();
    if(lDatabase == "MYSQL") return GDatabaseMySQL::Instance();
    return GDatabaseSQLite::Instance();
}
//=============================================
```

Gestionnaire de base de données SQLite

Déclarer le gestionnaire de base de données SQLite (GDatabaseSQLite.h)

```
//=============================================
#ifndef _GDatabaseSQLite_
#define _GDatabaseSQLite_
//=============================================
#include "GDatabase.h"
//=============================================
class GDatabaseSQLite : public GDatabase {
public:
    GDatabaseSQLite();
    ~GDatabaseSQLite();

public:
    static GDatabaseSQLite* Instance();
    void open();

private:
    static GDatabaseSQLite* m_instance;
};
//=============================================
#endif
//=============================================
```

Définir le gestionnaire de base de données SQLite (GDatabaseSQLite.cpp)

```
//================================================
#include "GDatabaseSQLite.h"
//================================================
GDatabaseSQLite* GDatabaseSQLite::m_instance = 0;
//================================================
GDatabaseSQLite::GDatabaseSQLite() {

}
//================================================
GDatabaseSQLite::~GDatabaseSQLite() {

}
//================================================
GDatabaseSQLite* GDatabaseSQLite::Instance() {
    if(m_instance == 0) {
        m_instance = new GDatabaseSQLite;
    }
    return m_instance;
}
//================================================
void GDatabaseSQLite::open() {
    cout << "Ouvrir la base de donnee SQLite\n";
}
//================================================
```

Gestionnaire de base de données MySQL

Déclarer le gestionnaire de base de données MySQL (GDatabaseMySQL.h)

```
//================================================
#ifndef _GDatabaseMySQL_
#define _GDatabaseMySQL_
//================================================
#include "GDatabase.h"
//================================================
class GDatabaseMySQL : public GDatabase {
public:
    GDatabaseMySQL();
    ~GDatabaseMySQL();

public:
    static GDatabaseMySQL* Instance();
    void open();

private:
    static GDatabaseMySQL* m_instance;
};
//================================================
#endif
//================================================
```

Définir le gestionnaire de base de données MySQL (GDatabaseMySQL.cpp)

```
//================================================
#include "GDatabaseMySQL.h"
//================================================
GDatabaseMySQL* GDatabaseMySQL::m_instance = 0;
//================================================
GDatabaseMySQL::GDatabaseMySQL() {

}
//================================================
GDatabaseMySQL::~GDatabaseMySQL() {

}
//================================================
GDatabaseMySQL* GDatabaseMySQL::Instance() {
    if(m_instance == 0) {
        m_instance = new GDatabaseMySQL;
    }
    return m_instance;
}
//================================================
void GDatabaseMySQL::open() {
    cout << "Ouvrir la base de donnee MySQL\n";
}
//================================================
```

[Résultat]