Template Method 패턴

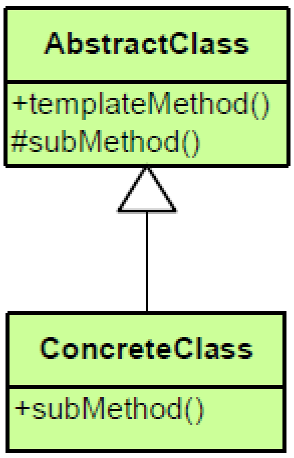
타입

Behavioral Pattern

문제

해결

클래스 다이어그램



예제

template_method.cpp

```
#include <ctime>
#include <assert.h>
#include <iostream>

namespace wikibooks_design_patterns
{
    /**
    * An abstract class that is common to several games in
    * which players play against the others, but only one is
    * playing at a given time.
    */

    class Game {
```

```
    public:
        Game(): playersCount(0), movesCount(0), playerWon(-1)
        {
            srand( (unsigned)time( NULL));
        }

        /* A template method : */
        void playOneGame(const int playersCount = 0) {

            if (playersCount)
                this->playersCount = playersCount;

            InitializeGame();
            assert(this->playersCount);

            int j = 0;
            while (!endOfGame()) {
                makePlay(j);
                j = (j + 1) % this->playersCount;
                if (!j) {
                    ++movesCount;
                }
            }
            printWinner();
        }

    protected:
        virtual void initializeGame() = 0;
        virtual void makePlay(int player) = 0;
        virtual bool endOfGame() = 0;
        virtual void printWinner() = 0;

    private:
        void InitializeGame()
        {
            movesCount = 0;
            playerWon = -1;

            initializeGame();
        }

    protected:
        int playersCount;
        int movesCount;
        int playerWon;
    };

    //Now we can extend this class in order
    //to implement actual games:

    class Monopoly: public Game {

        /* Implementation of necessary concrete methods */
        void initializeGame() {
            // Initialize players
            playersCount = rand() * 7 / RAND_MAX + 2;
            // Initialize money
        }
        void makePlay(int player) {
            // Process one turn of player

            //  Decide winner
            if (movesCount < 20)
                return;
            const int chances = (movesCount > 199) ? 199 : movesCount;
            const int random = MOVES_WIN_CORRECTION * rand() * 200 /
RAND_MAX;
            if (random < chances)
                playerWon = player;
        }
        bool endOfGame() {
            // Return true if game is over
            // according to Monopoly rules
            return (-1 != playerWon);
        }
        void printWinner() {
            assert(playerWon >= 0);
            assert(playerWon < playersCount);

            // Display who won
            std::cout<<"Monopoly, player "<<playerWon<<" won in
"<<movesCount<<" moves."<<std::endl;
        }

    private:
        enum
        {
            MOVES_WIN_CORRECTION = 20,
        };
    };

    class Chess: public Game {

        /* Implementation of necessary concrete methods */
        void initializeGame() {
            // Initialize players
            playersCount = 2;
            // Put the pieces on the board
        }
        void makePlay(int player) {
            assert(player < playersCount);

            // Process a turn for the player

            //  decide winner
            if (movesCount < 2)
                return;
            const int chances = (movesCount > 99) ? 99 : movesCount;
            const int random = MOVES_WIN_CORRECTION * rand() * 100 /
RAND_MAX;
            //std::cout<<random<<" : "<<chances<<std::endl;
            if (random < chances)
                playerWon = player;
        }
        bool endOfGame() {
            // Return true if in Checkmate or
            // Stalemate has been reached
            return (-1 != playerWon);
        }
        void printWinner() {
            assert(playerWon >= 0);
            assert(playerWon < playersCount);

            // Display the winning player
            std::cout<<"Player "<<playerWon<<" won in "<<movesCount<<"
moves."<<std::endl;
        }

    private:
        enum
        {
            MOVES_WIN_CORRECTION = 7,
        };
    };

}

int main()
{
    using namespace wikibooks_design_patterns;

    Game* game = NULL;

    Chess chess;
    game = &chess;
    for (unsigned i = 0; i < 100; ++i)
        game->playOneGame();
    Monopoly monopoly;
    game = &monopoly;
    for (unsigned i = 0; i < 100; ++i)
        game->playOneGame();

    return 0;
}
```

참고

http://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns#Template_Method