Ревью паттерна "Прототип"

* структура паттерна:



* Участники
  + Prototype () - прототип:

объявляет интерфейс для клонирования самого себя;

* + ConcretePrototype () - конкретный прототип:

реализует операцию клонирования себя;

* + Client () - клиент:

создает новый объект, обращаясь к прототипу с запросом клонировать

себя.

1. Prototype ()

class Confectionery {

public:

Confectionery() { }

// virtual Confectionery\* clone()=0;

// virtual void initialize(string, string)=0;

};

Если является классом-прототипом, то нет объявления функции clone()

Предлагаемое исправление: оставить объявление виртуальной функции

class Confectionery {

public:

Confectionery() { }

virtual Confectionery\* *clone*()=0;

};

2. ConcretePrototype ()

class CandyBar: public Confectionery {

string glaze;

string filling;

public:

CandyBar() {}

CandyBar(const CandyBar& obj) {

}

~CandyBar() {}

CandyBar clone() {

return \*this;

}

...

};

Операция клонирования реализована, но после исправления в п.1 следует немного ее дописать:

CandyBar \**clone*() {

return this;

}

Аналогично со вторым конкретным прототипом HardCandy

3. Client ()

int main()

{

...

vector <CandyBar> CandyBarBox(30);

vector <HardCandy> HardCandyBox(30);

int i = 0;

for(; i<10; i++) {

CandyBarBox[i]=mngr.BarPrototype["StrawberryBar"].clone();

HardCandyBox[i]=mngr.CandyPrototype["CoffeeCandy"].clone();

}

for(; i<20; i++) {

CandyBarBox[i]=mngr.BarPrototype["GreenTeaBar"].clone();

HardCandyBox[i]=mngr.CandyPrototype["CherryCandy"].clone();

}

for(; i<30; i++) {

CandyBarBox[i]=mngr.BarPrototype["TruffleBar"].clone();

HardCandyBox[i]=mngr.CandyPrototype["MintCandy"].clone();

...

}

Обращение к прототипу выполняется, после исправлений в п.2 переопределяем векторы:

vector <CandyBar\*> CandyBarBox(30);

vector <HardCandy\*> HardCandyBox(30);

**Вывод: паттерн реализован верно, за исключением отсутствия объявления абстрактного прототипа.**