Ревью паттерна "Строитель"

Структура:



Участники:

* **Builder** () - строитель:

задает абстрактный интерфейс для создания частей объекта Product;

* **ConcreteBuilder**()-конкретный строитель:

конструирует и собирает вместе части продукта посредством реализации интерфейса Builder;

определяет создаваемое представление и следит за ним;

предоставляет интерфейс для доступа к продукту;

* **Director** () - распорядитель:

конструирует объект, пользуясь интерфейсом Builder;

* **Product** () - продукт:

представляет сложный конструируемый объект. ConcreteBuilder

строит внутреннее представление продукта и определяет процесс его сборки;

включает классы, которые определяют составные части, в том числе интерфейсы для сборки конечного результата из частей.

1. **Builder()**

class SetBuilder

{

protected:

Set \*s;

public:

virtual ~*SetBuilder*() {}

virtual void *createSet*() {}

virtual void *buildRed*() {}

virtual void *buildGreen*() {}

virtual void *buildBlue*() {}

virtual void *buildYellow*() {}

virtual void *buildPurple*() {}

virtual void *buildBlack*() {}

virtual Set\* *getSet*() { return s; }

};

Нареканий нет.

2. **ConcreteBuilder**()

class RGB: public SetBuilder

{

public:

void *createSet*(){ s = new Set;}

void *buildRed*() {s->vr.push\_back(Red());

s->vr.push\_back(Red());}

void *buildGreen*() {s->vg.push\_back(Green());}

void *buildBlue*() {s->vb.push\_back(Blue());}

};

Нареканий нет.

3. **Director** ()

class Director

{

public:

Set\* createSet( SetBuilder &builder )

{

builder.*createSet*();

builder.*buildRed*();

builder.*buildGreen*();

builder.*buildBlue*();

builder.*buildYellow*();

builder.*buildPurple*();

builder.*buildBlack*();

return( builder.*getSet*());

}

};

Нареканий нет.

4. **Product** ()

int main()

{

Director dir;

RGB rgb\_builder;

Set \* rgb = dir.createSet(rgb\_builder);

return 0;

}

Вывод: паттерн реализован верно.