Фабричный метод

Назначение: система остается расширяемой при добавлении новых типов объектов, позволяет системе оставаться независимой как от процесса порождения, так и от типов объектов. Заранее известен момент создания объекта, но неизвестен его тип. Имеет две реализации: на основе обобщенного конструктора и фабрики.

1. Диаграмма ебат

   Пример:

```
enum Unit_Id{
    Warrior_Id = 0,
    Archer_Id,
    Lancer_Id
}
class Unit{
public:
    static Unit createUnit(Unit_Id Id){};
    virtual info() = 0;
    Unit(string info) : info_text(info){};
};

class Warior : public Unit{
    void info(){
        std::cout << "Воин " + getName();
    }
    Warrior(string s) : Unit(s){};
}
class Archer : public Unit{
    void info(){
        std::cout << "Лучник " + getName();
```

```
    }
    Archer (string s) : Unit(s){};
}
Unit Unit::createUnit(Unit_Id Id){
    Unit *p;
    case Id:
        Warrior_Id: p = new Warrior("Петя");break;
        Archer_Id: p = new Archer("Эмия Широ");break;
        default: p = NULL;
    return p;
}

int main(){
    std::vector <Unit *> v_unit;
    v_unit.push_back(Unit::createUnit(Warrior_Id));
    v_unit.push_back(Unit::createUnit(Archer_Id));
    for (int i = 0; i < v_unit.size(); i++){
        v_unit[i]->info();
    }
}
```

2. Еще диаграмма

```
class Unit{
public:
    virtual info() = 0;
    Unit(string info) : info_text(info){};
};

class Warior : public Unit{
    void info(){
        std::cout << "Воин " + getName();
    }
    Warrior(string s) : Unit(s){};
```

```
}

class Archer : public Unit{
    void info(){
        std::cout << "Лучник " + getName();
    }
    Archer (string s) : Unit(s){};
}

class Creator{
public:
    virtual Unit *createUnit = 0;
}
class CreatorWarrior : public Creator{
    Unit *createUnitUnit(){
        return new Warrior();
    }
}

class CreatorArcher : public Creator{
    Unit *createUnit(){
        return new Archer();
    }
}

int main(){
    Creator *createWarrior = new CreateWarrior();
    Creator *createArcher = new CreateArcher();
    std::vector <Unit *> v_unit;
    v_unit.push_bacc(craetorWarrior->createUnit());
    for (int i = 0; i<v_unit.size(); i++){
        v_unit[i]->info();
    }
}
```

Итератор:

Назначение: предоставление последовательного доступа к элементам контейнера (основного объекта).

```
class List{
    int items[n];
    int n;
    public:
    List();
    List(int el){
        items[n+1] = el;
    }
    void push(int el){
        items[n+1] = el;
    }
    int pop(){
        return items[--n];
    }
    Iterator *createIterator() const;
    friend classIterator;
}

IteratorList::createIterator(){
    return new IteratorList(this);
}

class Iterator{
    const List *l;
    int n;
    public:
    IteratorList(Collection *c);
    void begin(){ n = 0 };
    bool isDone{
        return n == l->n;
```

```
    }
    void next(){
        n++;
    }
    int get_item{
        return l->item[this->n];
    }
}
```