# Паттерн Flyweight (приспособленец)

## Назначение паттерна Flyweight

* Паттерн Flyweight использует разделение для эффективной поддержки большого числа мелких объектов.
* Является стратегией Motif GUI для замены "тяжеловесных" виджетов "легковесными" гаджетами.

Motif–библиотека для разработки приложений c графическим интерфейсом под X Window System. Появилась в конце 1980-х и на данный момент считается устаревшей.

В Motif элементы графического интерфейса (кнопки, полосы прокрутки, меню и т.д.) строятся на основе виджетов. Каждый виджет имеет свое окно. Исторически окна считаются "тяжеловесными" объектами. Если приложение с графическим интерфейсом использует множество виджетов, то производительность системы может упасть. Для решения этой проблемы в Motif предусмотрены гаджеты, являющиеся безоконными аналогами виджетов. Гаджеты управляются менеджерами виджетов, использующих схему "parent-children".

## Решаемая проблема

Проектирование системы из объектов самого низкого уровня обеспечивает оптимальную гибкость, но может быть неприемлемо "дорогим" решением с точки зрения производительности и расхода памяти.

## Обсуждение паттерна Flyweight

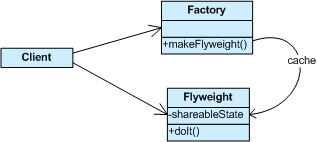
Паттерн Flyweight описывает, как совместно разделять очень мелкие объекты без чрезмерно высоких издержек. Каждый объект-приспособленец имеет две части: внутреннее и внешнее состояния. Внутреннее состояние хранится (разделяется) в приспособленце и состоит из информации, не зависящей от его контекста. Внешнее состояние хранится или вычисляется объектами-клиентами и передается приспособленцу при вызове его методов.

Замена Motif-виджетов "легковесными" гаджетами иллюстрирует этот подход. Если виджет являются достаточно самостоятельным элементом, то гаджет находится в зависимости от своего родительского менеджера компоновки виджетов. Каждый менеджер предоставляет своим гаджетам контекстно-зависимую информацию по обработке событий, ресурсам. Гаджет хранит в себе только контекстно-независимые данные.

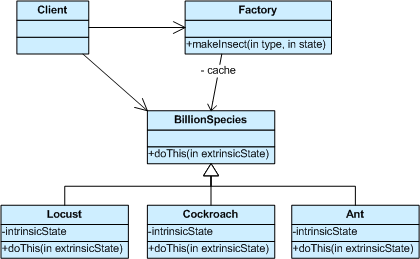
## Структура паттерна Flyweight

Клиенты не создают приспособленцев напрямую, а запрашивают их у фабрики. Любые атрибуты (члены данных класса), которые не могут разделяться, являются внешним состоянием. Внешнее состояние передается приспособленцу при вызове его методов. При этом наибольшая экономия памяти достигается в том случае, если внешнее состояние не хранится, а вычисляется при вызове.

### UML-диаграмма классов паттерна Flyweight

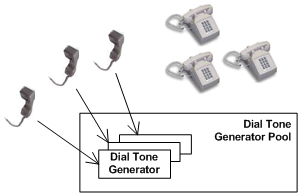


Классы, описывающие различных насекомых Ant, Locust и Cockroach могут быть "легковесными", потому что специфичная для экземпляров информация может быть вынесена наружу и затем, передаваться клиентом в запросе.



## Пример паттерна Flyweight

Паттерн Flyweight использует разделение для эффективной поддержки большого числа мелких объектов. Телефонная сеть общего пользования ТФОП является примером Flyweight. Такие ресурсы как генераторы тональных сигналов (Занято, КПВ и т.д.), приемники цифр номера абонента, набираемого в тоновом наборе, являются общими для всех абонентов. Когда абонент поднимает трубку, чтобы позвонить, ему предоставляется доступ ко всем нужным разделяемым ресурсам.



## Использование паттерна Flyweight

* Убедитесь, что существует проблема повышенных накладных расходов.
* Разделите состояние целевого класса на разделяемое (внутреннее) и неразделяемое (внешнее).
* Удалите из атрибутов (членов данных) класса неразделяемое состояние и добавьте его в список аргументов, передаваемых методам.
* Создайте фабрику, которая может кэшировать и повторно использовать существующие экземпляры класса.
* Для создания новых объектов клиент использует эту фабрику вместо оператора new.
* Клиент (или третья сторона) должен находить или вычислять неразделяемое состояние и передавать его методам класса.

## Особенности паттерна Flyweight

* Если Flyweight показывает, как сделать множество небольших объектов, то [Facade](http://cpp-reference.ru/patterns/structural-patterns/facade/) показывает, как представить целую подсистему одним объектом.
* Flyweight часто используется совместно с [Composite](http://cpp-reference.ru/patterns/structural-patterns/composite/) для реализации иерархической структуры в виде графа с разделяемыми листовыми вершинами.
* Терминальные символы абстрактного синтаксического дерева [Interpreter](http://cpp-reference.ru/patterns/behavioral-patterns/interpreter/) могут разделяться при помощи Flyweight.
* Flyweight объясняет, когда и как могут разделяться объекты [State](http://cpp-reference.ru/patterns/behavioral-patterns/state/).

## Реализация паттерна Flyweight

### Паттерн Flyweight: до и после

При использовании объектов на очень низких уровнях детализации накладные расходы могут быть непомерно большими. Использование паттерна Flyweight предполагает удаление неразделяемого состояния из класса и его передачу клиентом в вызываемые методы. И хотя это накладывает большую ответственность на клиента, но теперь создается значительно меньшее число объектов-приспособленцев. Совместное использование этих экземпляров облегчается с помощью класса-фабрики, поддерживающей "кэш" существующих приспособленцев.

В этом примере, "X" состояние рассматривается как разделяемое, а "Y" состояние выносится наружу (передается клиентом при вызове метода report()).

#### До

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | class Gazillion  {    public:      Gazillion()      {          m\_value\_one = s\_num / Y;          m\_value\_two = s\_num % Y;          ++s\_num;      }      void report()      {          cout << m\_value\_one << m\_value\_two << ' ';      }      static int X, Y;    private:      int m\_value\_one;      int m\_value\_two;      static int s\_num;  };    int Gazillion::X = 6, Gazillion::Y = 10, Gazillion::s\_num = 0;    int main()  {    Gazillion matrix[Gazillion::X][Gazillion::Y];    for (int i = 0; i < Gazillion::X; ++i)    {      for (int j = 0; j < Gazillion::Y; ++j)        matrix[i][j].report();      cout << '\n';    }  } |

Вывод программы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 00 01 02 03 04 05 06 07 08 09  10 11 12 13 14 15 16 17 18 19  20 21 22 23 24 25 26 27 28 29  30 31 32 33 34 35 36 37 38 39  40 41 42 43 44 45 46 47 48 49  50 51 52 53 54 55 56 57 58 59 |

#### После

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | class Gazillion  {    public:      Gazillion(int value\_one)      {          m\_value\_one = value\_one;          cout << "ctor: " << m\_value\_one << '\n';      }      ~Gazillion()      {          cout << m\_value\_one << ' ';      }      void report(int value\_two)      {          cout << m\_value\_one << value\_two << ' ';      }    private:      int m\_value\_one;  };    class Factory  {    public:      static Gazillion \*get\_fly(int in)      {          if (!s\_pool[in])            s\_pool[in] = new Gazillion(in);          return s\_pool[in];      }      static void clean\_up()      {          cout << "dtors: ";          for (int i = 0; i < X; ++i)            if (s\_pool[i])              delete s\_pool[i];          cout << '\n';      }      static int X, Y;    private:      static Gazillion \*s\_pool[];  };    int Factory::X = 6, Factory::Y = 10;  Gazillion \*Factory::s\_pool[] =  {    0, 0, 0, 0, 0, 0  };    int main()  {    for (int i = 0; i < Factory::X; ++i)    {      for (int j = 0; j < Factory::Y; ++j)        Factory::get\_fly(i)->report(j);      cout << '\n';    }    Factory::clean\_up();  } |

Вывод программы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | ctor: 0  00 01 02 03 04 05 06 07 08 09  ctor: 1  10 11 12 13 14 15 16 17 18 19  ctor: 2  20 21 22 23 24 25 26 27 28 29  ctor: 3  30 31 32 33 34 35 36 37 38 39  ctor: 4  40 41 42 43 44 45 46 47 48 49  ctor: 5  50 51 52 53 54 55 56 57 58 59  dtors: 0 1 2 3 4 5 |

### Паттерн Flyweight: разделение иконок.

Паттерн Flyweight показывает, как эффективно разделять множество мелких объектов. Ключевая концепция - различие между внутренним и внешним состояниями.

Внутреннее состояние состоит из информации, которая не зависит от контекста и может разделяться (например, имя иконки, ее ширина и высота). Оно хранится в приспособленце (то есть в классе Icon).

Внешнее состояние не может разделяться, оно зависит от контекста и изменяется вместе с ним (например, координаты верхнего левого угла для каждого экземпляра иконки). Внешнее состояние хранится или вычисляется клиентом и передается приспособленцу при вызове операций. Клиенты не должны создавать экземпляры приспособленцев напрямую, а получать их исключительно из объекта FlyweightFactory для правильного разделения.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134 | #include <iostream.h>  #include <string.h>    class Icon  {    public:      Icon(char \*fileName)      {          strcpy(\_name, fileName);          if (!strcmp(fileName, "go"))          {              \_width = 20;              \_height = 20;          }          if (!strcmp(fileName, "stop"))          {              \_width = 40;              \_height = 40;          }          if (!strcmp(fileName, "select"))          {              \_width = 60;              \_height = 60;          }          if (!strcmp(fileName, "undo"))          {              \_width = 30;              \_height = 30;          }      }      const char \*getName()      {          return \_name;      }      draw(int x, int y)      {          cout << "   drawing " << \_name               << ": upper left (" << x << "," << y               << ") - lower right (" << x + \_width << ","               << y + \_height << ")" << endl;      }    private:      char \_name[20];      int \_width;      int \_height;  };    class FlyweightFactory  {    public:      static Icon \*getIcon(char \*name)      {          for (int i = 0; i < \_numIcons; i++)            if (!strcmp(name, \_icons[i]->getName()))              return \_icons[i];          \_icons[\_numIcons] = new Icon(name);          return \_icons[\_numIcons++];      }      static void reportTheIcons()      {          cout << "Active Flyweights: ";          for (int i = 0; i < \_numIcons; i++)            cout << \_icons[i]->getName() << " ";          cout << endl;      }    private:      enum      {          MAX\_ICONS = 5      };      static int \_numIcons;      static Icon \*\_icons[MAX\_ICONS];  };    int FlyweightFactory::\_numIcons = 0;  Icon \*FlyweightFactory::\_icons[];    class DialogBox  {    public:      DialogBox(int x, int y, int incr):         \_iconsOriginX(x), \_iconsOriginY(y), \_iconsXIncrement(incr){}      virtual void draw() = 0;    protected:      Icon \*\_icons[3];      int \_iconsOriginX;      int \_iconsOriginY;      int \_iconsXIncrement;  };    class FileSelection: public DialogBox  {    public:      FileSelection(Icon \*first, Icon \*second, Icon \*third):          DialogBox(100, 100, 100)      {          \_icons[0] = first;          \_icons[1] = second;          \_icons[2] = third;      }      void draw()      {          cout << "drawing FileSelection:" << endl;          for (int i = 0; i < 3; i++)            \_icons[i]->draw(\_iconsOriginX +                            (i \*\_iconsXIncrement), \_iconsOriginY);      }  };    class CommitTransaction: public DialogBox  {    public:      CommitTransaction(Icon \*first, Icon \*second, Icon \*third):          DialogBox(150, 150, 150)      {          \_icons[0] = first;          \_icons[1] = second;          \_icons[2] = third;      }      void draw()      {          cout << "drawing CommitTransaction:" << endl;          for (int i = 0; i < 3; i++)            \_icons[i]->draw(\_iconsOriginX +                            (i \*\_iconsXIncrement), \_iconsOriginY);      }  };    int main()  {    DialogBox \*dialogs[2];    dialogs[0] = new FileSelection(                        FlyweightFactory::getIcon("go"),                        FlyweightFactory::getIcon("stop"),                        FlyweightFactory::getIcon("select"));    dialogs[1] = new CommitTransaction(                        FlyweightFactory::getIcon("select"),                        FlyweightFactory::getIcon("stop"),                        FlyweightFactory::getIcon("undo"));      for (int i = 0; i < 2; i++)      dialogs[i]->draw();      FlyweightFactory::reportTheIcons();  } |

Вывод программы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | drawing FileSelection:     drawing go: upper left (100,100) - lower right (120,120)     drawing stop: upper left (200,100) - lower right (240,140)     drawing select: upper left (300,100) - lower right (360,160)  drawing CommitTransaction:     drawing select: upper left (150,150) - lower right (210,210)     drawing stop: upper left (300,150) - lower right (340,190)     drawing undo: upper left (450,150) - lower right (480,180)  Active Flyweights: go stop select undo |

**http://cpp-reference.ru/patterns/structural-patterns/flyweight/**