

2-й этап разработки курсовой работы — разработка иерархии и интерфейса классов.

Результат этого этапа используется во втором разделе основной части КР, включающем разбиение на модули, классы и их интерфейсы, функции и глобальные переменные, иерархию классов. В отчете по курсовой работе этот документ дополняется и приводится в соответствие с разработанным кодом.

Объекты, найденные в результате анализа предметной области, разделяются на классы. **Класс** – это множество объектов, имеющих схожую структуру и поведение. Если совпадение поведения неполное или структура объектов как схожие характеристики, так и различающиеся, то это означает, что необходимо построить иерархию классов на основе отношения наследования. Наличие составных объектов говорит о необходимости отношения включения. В остальных случаях классы между собой могут связаны отношением использования. Подробнее об отношениях между классами и их обозначениях в диаграмме написано на странице <https://ipc.susu.ru/14616-1.html>

Для описания интерфейса классов при наличии отношения наследования нужно изучить <https://ipc.susu.ru/8506.html>. Если в задании необходима перегрузка операций, то нужно изучить <https://ipc.susu.ru/8505.html>

В некоторых вариантах задания могут быть приведены рекомендации по интерфейсу классов и/или реализации, которыми можно воспользоваться.

Результаты данного этапа являются предварительными и могут измениться после реализации кода.

Рассмотрим пример с графическим редактором.

Все выявленные объекты занимают некоторую прямоугольную область на экране и должны перерисовываться при изменении состояния. Некоторые объекты - кнопки управления и поле для рисования - должны реагировать на нажатие кнопок мыши на них. Также кнопки управления по поведению были разделены на 3 группы, различающие по поведению.

В результате выстраивается следующая иерархия классов:

класс для объектов на экране (хранит координаты)

1. подкласс для отображения текущих цветов рисования
2. подкласс для отображения текущих координат мыши и имени файла
3. подкласс для объектов, реагирующих на мышь
 - 3.1. подкласс для поля рисования
 - 3.2. подкласс для кнопок управления
 - 3.2.1. подкласс для кнопок выбора цвета
 - 3.2.2. подкласс для кнопок выбора инструмента рисования
 - 3.2.3. подкласс для кнопок-команд (сохранение или загрузка)

Программу можно либо не разбивать на модули (всё в paint.cpp), либо выделить 3 модуля: основная программа и вспомогательные функции (main.cpp), базовые классы и классы, содержащие по одному объекту

(interface.cpp, interface.hpp), классы для кнопок управления (button.cpp, button.hpp). Выбор варианта зависит от объема кода - если код большой, то его нужно разбить на несколько модулей. Модули должны содержать связанные между собой классы.

У классов 1, 2, 3.1 только один объект, для них можно использовать паттерн проектирования «одиночка» <https://ipc.susu.ru/8497-9.html>

Результат разработки иерархии и интерфейса классов для графического редактора см. в примере 1.

Рассмотрим **пример библиотеки классов для работы с форматом JSON**.

Поведение объектов должно меняться в зависимости от вида значения.

Можно применить паттерн проектирования «Состояние». Плюсами использования этого паттерна является уменьшение количества явных проверок текущего типа объекта, легкость добавления нового типа, упрощение хранения состояния объекта. Пример реализации можно найти в <https://github.com/dropbox/json11>. Но есть минусы — необходимо определить дополнительные классы для каждого типа значения. Кроме того, добавление новых типов в данной задаче не планируется, а большинство операций предназначено только для одного типа значений, поэтому в операциях будет обычно одна проверка на текущий тип объекта. Поэтому паттерн «Состояние» использовать не будем.

Для хранения строк будем использовать стандартный класс string, для массивов — класс vector, для ассоциативных массивов — класс map. Для классов vector и map сделаем собственные псевдонимы array и object, что позволит воспользоваться конструкторами классов vector и map, но при этом указать более подходящее имя для типа значения.

Значения можно хранить в union или как указатель на значение, выполняя преобразование к нужному типу с помощью reinterpret_cast, но такой вариант легко может привести к ошибкам. Типы string, map, vector уже являются динамическими типами, поэтому не занимают много памяти. Чтобы избежать сложных, потенциально опасных действий и переложить большую часть работы по управлению памятью на компилятор, мы все виды значений представим в виде набора из пяти полей (для хранения null поля не требуется, только флажок в value_type)

Для класса нужно определить операции преобразования, конструкторы-преобразователи, операции ввода и вывода, операции индексации. В перегрузке конструктора копий и присваивания нет необходимости, но лучше перегрузить для повышения эффективности (чтобы копировать только используемое поле и очистить память от ненужных уже данных).

Пример 1. Растровый графический редактор

Разработка иерархии и интерфейса классов.

1. Модуль main (файл main.cpp) содержит следующие функции:

```
int main(); // основная функция
```

```
void init_controls(); // инициализация объектов программы
```

```
void draw_controls(); // инициализация объектов программы
```

и глобальные переменные:

```
Button* buttons[22]; // кнопки управления
```

2. Модуль interface (интерфейсная часть в файле interface.hpp, реализация в файле interface.cpp) содержит следующие классы:

```
// класс для объектов на экране
```

```
class DrawObject {
```

```
protected:
```

```
    int x1,y1,x2,y2; // координаты углов прямоугольника, в котором находится объект
```

```
public:
```

```
    // конструктор получает координаты левого верхнего угла x1,y1
```

```
    // и координаты правого нижнего угла x2,y2
```

```
    DrawObject(int x1, int y1, int x2, int y2);
```

```
    virtual ~DrawObject() {} // деструктор
```

```
    bool in(int x, int y); // координаты x, y внутри объекта
```

```
    virtual void draw()=0; // нарисовать объект
```

```
};
```

```
// класс для отображения текущих цветов рисования
```

```
class SelectedColors: public DrawObject {
```

```
    int fc, // цвет рисования
```

```
    bc; // цвет закрашки
```

```
    // конструктор получает координаты углов x1,y1,x2,y2
```

```
    SelectedColors(int x1, int y1, int x2, int y2);
```

```
public:
```

```
    static SelectedColors &instance(); // объект для текущих цветов
```

```
    void draw(); // нарисовать объект
```

```
    void set_fc(int c); // установить цвет рисования
```

```
    int get_fc(); // получить цвет рисования
```

```
    void set_bc(int c); // установить цвет закрашки
```

```
    int get_bc(); // получить цвет закрашки
```

```
};
```

```
// класс для отображения текущих координат мыши и имени файла
```

```
class StatusLine: public DrawObject {
```

```
    int x,y; // координаты мыши
```

```
    string name; // имя файла
```

```
    // конструктор получает координаты углов x1,y1,x2,y2
```

```
    StatusLine(int x1, int y1, int x2, int y2);
```

```
public:
```

```
    static StatusLine &instance(); // строка состояния
```

```
    void draw(); // нарисовать объект
```

```

void set_xy(int x, int y); // изменить текущие координаты
void set_name(const string& name); // установить имя файла
string get_name(); // получить имя файла
void edit_name(); // ввести имя файла
};
// класс для объектов, реагирующих на мышь
class ControlObject: public DrawObject {
public:
    // конструктор получает координаты углов x1,y1,x2,y2
    ControlObject(int x1, int y1, int x2, int y2);
    virtual void press()=0; // реакция на нажатие
};
// вспомогательный тип - указатель на функцию
typedef void (*fun_ptr)();
// класс для поля рисования
class PaintArea: public ControlObject {
    fun_ptr tool; // текущий инструмент
    // конструктор получает координаты углов x1,y1,x2,y2
    PaintArea(int x1, int y1, int x2, int y2);
public:
    static PaintArea &instance(); // поле для рисования
    void draw(); // нарисовать объект
    void press(); // реакция на нажатие
    void set_tool(fun_ptr t) // установить инструмент рисования
    fun_ptr get_tool() // получить текущий инструмент рисования
    void save(const string& name); // сохранить изображение
    void load(const string& name); // загрузить изображение
};

```

3. Модуль button (интерфейсная часть в файле button.hpp, реализация в файле button.cpp) содержит следующие классы:

```

// Базовый класс для кнопок управления
class Button: public ControlObject {
public:
    // конструктор получает координаты левого верхнего угла x,y
    Button(int x, int y);
};
// Класс для кнопок выбора цвета
class ColorButton: public Button {
    int c; // цвет кнопки
public:
    // конструктор получает координаты x,y и номер цвета c
    ColorButton(int x, int y, int c):Button(x,y),c(c) {}
    void draw(); // нарисовать объект
    void press(); // реакция на нажатие
};

```

```

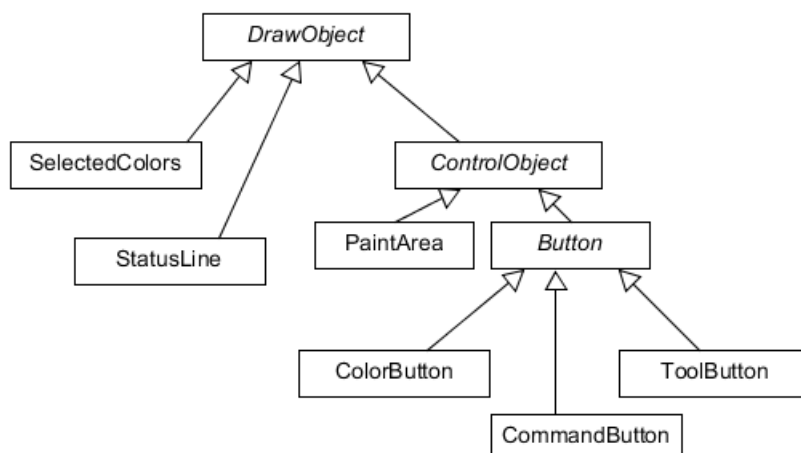
// Класс для кнопок выбора инструмента
class ToolButton: public Button {
    fun_ptr tool; // инструмент
    IMAGE image; // загруженная картинка для кнопки
public:
    // конструктор получает координаты x,y, имя файла с картинкой icon
    // и указатель на функцию-инструмент t
    ToolButton(int x, int y, string icon, fun_ptr t);
    ~ToolButton(); // деструктор
    void draw(); // нарисовать объект
    void press(); // реакция на нажатие
};

// Класс для кнопок-команд
class CommandButton: public Button {
    fun_ptr action; // действие
    IMAGE image; // загруженная картинка для кнопки
public:
    // конструктор получает координаты x,y, имя файла с картинкой icon
    // и указатель на действие a
    CommandButton(int x, int y, string icon, fun_ptr a);
    ~CommandButton(); // деструктор
    void draw(); // нарисовать объект
    void press(); // реакция на нажатие
};

и глобальные функции
void pencil_tool(); // инструмент карандаш
void line_tool(); // инструмент линия
void rect_tool(); // инструмент прямоугольник
void fillrect_tool(); // инструмент закрашенный прямоугольник
void load_command(); // загрузка рисунка
void save_command(); // сохранение рисунка

```

Иерархия классов



Пример 2. Библиотека классов для работы с форматом JSON

Разработка иерархии и интерфейса классов.

Модуль json (интерфейсная часть в файле json.hpp, реализация в файле json.cpp) содержит следующие классы:

```
class JSON {
public:
    // константы для типа элементов
    enum JSON_type {
        null_type=0,
        bool_type,
        number_type,
        string_type,
        array_type,
        object_type
    };
    // массив элементов
    typedef vector<JSON> array;
    // ассоциативный массив элементов
    typedef map<string, JSON> object;

    // конструкторы
    JSON();
    JSON(const JSON &);

    // конструкторы-преобразователи
    JSON(nullptr_t);
    JSON(bool);
    JSON(double);
    JSON(int);
    JSON(const string &);
    JSON(const char []);
    JSON(const array &);
    JSON(const object &);

    // операция присваивания
    JSON &operator=(const JSON &);
    // получить тип элемента
    JSON_type type() const;

    // добавить элемент e в массив
    void push(const JSON& e);
    // удалить последний элемент из массива
    void pop();
    // получить i-й элемент массива
    JSON &operator[](int i);
```

```

const JSON &operator[](int i) const;
// получить длину массива
int length() const;

// получить элемент ассоциативного массива по ключу key
JSON &operator[](const string & key);
const JSON &operator[](const string & key) const;
// проверить наличие элемента с ключом key
bool has(const string & key) const;
// удалить элемент с ключом key
void erase(const string &key);

// операции преобразования
operator bool() const;
operator double() const;
operator const char *() const;
operator string() const;

// операция вывода в поток
friend ostream &operator<<(ostream &o, const JSON &j);
// операция ввода из потока
friend istream &operator>>(istream &i, JSON &j);
private:
JSON_type value_type; // тип значения
bool bool_value;      // поле для хранения булевского значения
double number_value;  // поле для хранения числа
string string_value;  // поле для хранения строки
array array_value;    // поле для хранения массива
object object_value;  // поле для хранения ассоциативного массива
void clear();         // метод для очистки, значение=null
};

```