Тема: Виды иерархий

- Наследование поведения и реализации
- Общая реализация
- Общее поведение
- Роль абстрактных классов

Наследование используется для группировки общих черт классов

- Общие поведение и реализация
- Общая только реализация
- Общее только поведение

- Общее поведение
  - производные классы разделяют публичные операции (методы) базового класса
  - позволяют писать полиморфный код, основанный на функциях базового класса (возможно, перегружаемых в производном классе)
- Общая реализация
  - производные классы разделяют данные (свойства) базового класса
  - операции (свойства) производного класса могут вызывать операции (свойства) базового класса

Общая реализация

Классы ***Stack*** и ***Set***

- могут быть реализованы в виде связного списка (на основе класса ***List***)
- у каждого свои методы
- не должны использовать операции класса ***List***

Класс ***List***

```
class List {
  public:
    int find(const element& ); // если не найдено, то возврат -1
    element  lookup( int index );
    void set( int index, const  element& );
    void insert( int index, const  element& );
    void remove( int index );
    int size( );
  private:
     // …
};
```

Для общей реализации можно использовать закрытое наследование

Классы ***Stack*** и ***Set*** : (закрытое наследование)

```
class Stack : private List {
  public:
    void push( const  element& );
    element  pop( );
    int size( );
  private:
};
```

```
class Set : private List {
  public:
    void add( const  element& );
    int in(const  element&);
    int size( );
  private:
};
```

Классы ***Stack*** и ***Set*** : (реализация методов)

```
void Stack :: push( const  element&  e ){
        insert( size( ), e ) ;
}

void Set :: add( const  element&  e ){
   if( find(e) == -1 ) insert( 0, e ) ;
}
```

Для общей реализации можно использовать закрытые свойства

Классы ***Stack*** и ***Set*** : (закрытые свойства)

```
class Stack {
  public:
    void push( const  element& );
    element  pop( );
    int size( );
  private:
    List s;
};
```

```
class Set {
  public:
    void add( const  element& );
    int in(const  element&);
    int size( );
  private:
    List s;
};
```

Классы ***Stack*** и ***Set*** : (реализация методов)

```
void Stack :: push( const  element&  e ){
        s.insert( s.size( ), e ) ;
}

void Set :: add( const  element&  e ){
   if( s.find(e) == -1 ) s.insert( 0, e ) ;
}
```

Классы ***List, Stack*** и ***Set*** : (использование)

```
void main( int, char*[] ){
    Stack  st ;
    Set  s ;
    List  sl ;
   sl.insert( 0, 256 ) ;
  sl.insert( 1, 128 ) ;
  sl.insert( 0, 128 ) ;

  st.push( 1 ) ;
  st.push( 2 ) ;
```

```
    st.push( 3 ) ;
    st.insert( 1, 4 ) ; // ошибка

    s.add( 2 ) ;
    s.add( 3 ) ;
    s.add( 5 ) ;
    s.insert( 1, 5 ) ; // ошибка
}
```

Общее поведение

Классы ***Screen*** и ***Printer***

- имеют общий интерфейс (методы)
- имеют общий контекст использования
- имеют различную реализацию (свойства)

Класс ***Screen***

```
class Screen  {
  public:
    void clear( );
    void add( Display_char  ch );
    void add( const  String&  str );
    Point size( ) const;
    Point cursor( ) const;
    int move_cursor(const  Point&   p );
  private:
    // реализация класса Screen
};
```

Класс ***Printer***

```
class Printer {
  public:
    void clear( );
    void add( Display_char ch );
    void add( const String& str );
    Point size( ) const;
    Point cursor( ) const;
    int move_cursor(const Point& p );
    void print( );
  private:
    // реализация класса Printer
};
```

Для общего поведения используем ***абстрактный базовый класс***

- чистые виртуальные методы
- свойств либо мало, либо нет вообще

Абстрактный базовый класс

```
class Display_medium {
  public:
    virtual Point size( ) const = 0;
    virtual Point cursor( ) const = 0;
    virtual int move_cursor(const Point& p ) = 0;
    virtual Display_char character( ) const = 0;
    virtual String line( ) const = 0;
    virtual void add( Display_char ch ) = 0;
    virtual void add( const String& s ) = 0;
    virtual void clear( );
  private:
};
```

Метод ***clear( )***

```
void  Display_medium :: clear( void ) {
    int  x , y ;
    for ( x = 0; x < size().x() ; x++ )
     for ( y = 0; y < size().y() ; y++ ) {
          move_cursor( Point(x,y) );
          add( ‘  ’ );
     }
}
```

Производный класс ***Screen***

```
class Screen : public Display_medium {
  public:
    Screen( );
    ~Screen( );
    Point size( ) const;
    Point cursor( ) const;
    int move_cursor(const  Point& p );
    Display_char  character( ) const;
    String  line( ) const;
    void add( Display_char  ch );
    void add( const  String& s );
    void clear( );
  private:
    // реализация класса Screen
};
```

Производный класс ***Printer***

```
class Printer : public Display_medium {
  public:
    Printer( );
    ~Printer( );
    Point size( ) const;
    Point cursor( ) const;
    int move_cursor(const  Point& p );
    Display_char  character( ) const;
    String  line( ) const;
    void add( Display_char  ch );
    void add( const  String& s );
    void clear( );
    virtual void print( );
  private:
    // реализация класса Printer
};
```

Использование абстрактного класса

```
void  say_hello( Display_medium& m ) {
      m.add( “Hello, world \n” );
}

extern  Screen display;
extern  Printer line_printer;

void main( void ) {
    say_hello( display ) ;
    say_hello( line_printer ) ;
    line_printer.print( );
}
```

Абстрактный класс

- если класс имеет хотя бы один чистый виртуальный метод, то он абстрактный
- создать объект абстрактного типа нельзя
- можно использовать такой тип для
  - формальных параметров
  - указателей
- имеет смысл только как базовый класс
- не сокращает объем кодирования потомков
- улучшает структурированность программы (поддержка полиморфизма)

РЕЗЮМЕ

- Классы имеют общее поведение и реализацию
  - производные классы наследуют методы и свойства базового класса
  - полиморфные функции должны использовать методы базового класса
  - унаследованная реализация упрощает программирование производных классов
- Только реализацию
  - используются закрытые свойства или закрытое наследование
  - упрощено программирование производных классов
- Только поведение
  - абстрактные базовые классы задают поведение (имена методов)
  - полиморфные функции используют методы абстрактного базового класса
  - не упрощает программирование производных классов