Discussion of Homework 1

2005/10/15

Meng-Che Chen, Zheng-Wen Shen
System Architecture

User → Command → Input

Display

Output

request

Text

TextEditorApp
Object-Based design

Diagram showing the class relationships and methods of the TextEditorApp, Display, Input, Text, Tokener, Location, TextLocationMap, and TextLocationMapBuilder classes.
Object-Oriented Design Principles

- Open-Closed (OCP)
- Single Responsibility (SRP)
- Liskov Substitution (LSP)
- Dependency Inversion (DIP)
- Interface Segregation (ISP)
- Composition/Aggregation Reuse (CARP)
- Least Knowledge (LKP)
Open-Closed Principle (OCP)

Software entities (Classes, Modules, Functions, etc.) should be \textit{open for extension}, but \textit{closed for modification}. 
Abstraction is the key

In hw2? hw3? or hw7??

closed for modification

open for extension
Single Responsibility principle (SRP)

- There should never be more than one reason for a class to change.
Design of TextLocationMap

1. Create

- **Data encapsulation**
- **Data display**

Not good!!!

Client

- **Text**
  - `+ doService() : void`

Product

- **TextLocationMap**
  - `+ add() : void`
  - `+ getLocationSet() : void`
  - `+ dump() : void`

Builder

1. Create

1. Data encapsulation
2. Data display

Not good!!!
bool Text::readFile(string filename)
{
    //...
    TextLocationMapBuilder builder;
    m_pLocationMap = builder.build( m_pTexts );
    //...
}

TextLocationMap*
TextLocationMapBuilder::
build(vector<string>* lines_of_text)
{
    vector<string> words;
    vector<Location> locs;

    // detail of product
    separate_words(lines_of_text, &words, &locs);
    filter_text( &words );
    suffix_text( &words );
    strip_cap( &words );
    sort_text( &words, &locs );

    // build the entire product
    return build_location_map( &words, &locs );
}
TextLocationMap::dump()

Communication, sensitive Display object

Context (Text Location Map) -> Display -> Context3

Context2
class OutputToDisplay {
public:
    OutputToDisplay(Display *display): _display(display) {}
    ~OutputToDisplay() {}
    void operator()(string line) { _display->drawText( line ); }
private:
    OutputToDisplay() {}
    Display * _display;
};
template <class Output>

void TextLocationMap::dump(Output out) {
  string line = ""
  for(unsigned int i=0; i<_words->size(); i++) {
    line = (*_words)[i] + " (";
    LocationSet *ls = &(*_locs)[i];
    for(unsigned int j=0; j<ls->size(); j++) {
      if (j < ls->size()-1) {
        line += (*ls)[j].toString() + " ", ";
      } else {
        line += (*ls)[j].toString();
      }
    }
    line += ")";
    out(line);
  }
}
Test the \textit{Input}
class Input
{
public:
  Input();
  virtual ~Input();
  void processInput();
  void setText(Text *t);
  bool dispatchCommand(string cmd, Tokener arg);
private:
  Text *m_pText;
  string last_cmd;
  Tokener last_arg;
};
Test the Text

CppUnit

MockDisplay

Display

Text

Input

Command

request

Output

TextEditorApp

2005 Fall OOP
class Text
{
public:
    bool readFile(string filename);
    bool saveFile(string filename);
    bool search(string arg);
    void display(string mode = "");

    bool previousPage();
    bool nextPage();

    bool previousLine();
    bool nextLine();

    bool moveTo(int x, int y);

    bool home();
    bool end();
    void map();

    void setDisplay(Display *v);
};