Detailed Design Document

Ausbesserungsbeispiel (ExA)

Softwareentwicklung Praktikum

Einleitung

Lernziel des Ausbesserungsbeispiels ist es, eine objektorientierte Version eines Programms zur Generierung von Vektorgrafiken zu erlernen.

Schreiben Sie ein Programm, mit dem Sie auf der Konsole rudimentäre SVG-Dateien http://de.wikipedia.org/wiki/SVG erstellen können. Sie können diese Dateien später mit einem Webbrowser anzeigen.

Aufgabenstellung

Das Programm erwartet einen Kommandozeilenparameter. Dieser beginnt mit ["-c="] und gibt den Namen eines Konfigurationsfiles an.

Das Programm gibt danach in einer Endlosschleife einen Standardprompt (z.B.: ["sep> "] näheres siehe Konfiguration) aus und wartet auf die Eingabe eines der definierten Befehle. Die Eingabe der Befehle erfolgt case sensitive. Nach Eingabe eines Befehls erfragt das Programm gegebenenfalls die Parameter durch ein eigenes Parameterprompt (jeweils eine Zeile pro Parameter, siehe Beschreibung der Befehle).

Beschreibung des Designs

Das Programm besteht aus 23 Klassen. Die Zusammenhänge der
Klassen und deren Members kann dem Klassendiagramm am Ende des Dokuments entnommen werden.

UserInterface

Attribute:

- `std::vector<Command*> commands_;`
- `bool run_`
- `std::string prompt_;`
- `Database *db_`
- `SVGDocument *svgdoc_;`
- `SVGHandler *svgh_;`
- `Error *error_;`

Methoden:

- `UserInterface(Database *db, SVGDocument *svgdoc_, SVGHandler *svgh, std::string prompt);`
  - Konstruktor
- `virtual ~UserInterface();`
  - Destruktor
- `void run();`
  - Exekutiert einen Befehl und gibt true zurück wenn erfolgreich, andernfalls false.
- `std::string getString();`
  - Liest die Eingabe des Users und gibt diese als String zurück.
- `bool checkCommand(std::string& name);`
  - Überprüft ob ein Befehl eingegeben wurde und führt diesen aus wenn es ihn gibt. Falls nicht wird false zurückgegeben.
- `bool stringToUnsignedInt(std::string& string_number, unsigned int& value);`
  - Überprüft ob ein eingelesener Wert ein Integer ist.
- `void setRun(bool run);`
  - Setzt den Parameter run_. Damit kann das Programm beendet werden.
- `bool getParam(std::string prompt, bool polygon, bool mod_command, signed int &int_value, std::string &str_value, bool conv_int)`
  - Checkt ob ein String leer ist und gibt den String zurück
- `bool checkID(std::string id);`
  - Checkt ob eine id correkt eingegeben wurde
- `bool checkIfIdExists(std::string id);`
  - Checkt ob eine Id bereits existiert. Gibt true zurück falls die id bereits existiert. Andernfalls false.

SVGHandler

Attribute:

- `Error *error_;`
- `UserInterface *ui_;`
- `Database *db_;`
- `SVGDocument *svgdoc_;`
- `signed int errors_;`
- `std::string prompt_;`

Methoden:

- `SVGHandler(unsigned int arg_count, char **parameters);`
  - Konstruktor
- `virtual ~SVGHandler();`
  - Destruktor
- `signed int getErrors();`
  - Gibt den Fehler zurück.
- `void setErrors(signed int errorcode);`
  - Setzt den Fehler.
- `bool checkParameter(std::string parameter);`
  - Checkt die Commandline Parameter und gibt true oder false zurück.
- `Database* getDB();`
  - Gibt ein File handle auf die Datenbank zurück.
- `void freeMemory();`
  - Löscht den angelegten Speicher.
- `void setPrompt(std::string prompt);`
  - Setzt den Prompt der aus den Config file ausglesen wird bei der Commandline als Ausgabeprompt.

Basisklasse:

Command

Attribute:

- `UserInterface* ui_;`
- `Database *db_;`
- `std::string name_;`

Methoden:

- `Command(UserInterface *ui, Database *db);`
  - Konstruktor
- `virtual ~Command();`
  - Destruktor
- `virtual bool execute();`
  - Exekutiert einen Befehl und gibt true zurück wenn erfolgreich, andernfalls false.
- `virtual std::string getName();`
  - Gibt den Namen zurück.

Davon abgeleitete Klassen:

LineCommand:

Attribute:

- `SVGHandler *svgh_;`
- `Error *error_;`
- `SVGCoordinates *coordinates_;`

Methoden:

- `LineCommand(UserInterface *ui, Database *db, SVGHandler *svgh);`
  - Konstruktor
- `virtual ~LineCommand();`
  - Destruktor
- `virtual bool execute();`
  - Erstellt ein neues Objekt vom Type SVGLine.
- `signed int readParam(std::string prompt);`
  - Liest die Parameter von dem UserInterface ein.

CircleCommand:

Attribute:

- `SVGHandler *svgh_;`
- `Error *error_;`
- `SVGCoordinates *coordinates_;`

Methoden:

- `CircleCommand(UserInterface *ui, Database *db, SVGHandler *svgh);`
  - Konstruktor
- `virtual ~CircleCommand();`
  - Destruktor
- `virtual bool execute();`
  - Erstellt ein neues Objekt vom Type SVGCircle.
- `signed int readParam(std::string prompt);`
  - Liest die Parameter von dem UserInterface ein.

RectCommand:

Attribute:

- `SVGHandler *svgh_;`
- `Error *error_;`
- `SVGCoordinates *coordinates_;`

Methoden:

- `RectCommand(UserInterface *ui, Database *db, SVGHandler *svgh);`
  - Konstruktor
- `virtual ~RectCommand();`
  - Destruktor
- `virtual bool execute();`
  - Erstellt ein neues Objekt vom Type SVGRect.
- `signed int readParam(std::string prompt);`
  - Liest die Parameter von dem UserInterface ein.

PolygonCommand:

Atrribute:

- `SVGHandler *svgh_;`
- `Error *error_;`
- `SVGCoordinates *coordinates_;`

Methoden:

- `PolygonCommand(UserInterface *ui, Database *db, SVGHandler *svgh);`
  - Konstruktor
- `virtual ~PolygonCommand();`
  - Destruktor
- `virtual bool execute();`
  - Erstellt ein neues Objekt vom Type SVGPolygon.
- `void readCoordinates();`
  - Liest die Koordinaten eines SVGPolygons ein.

MoveCommand:

Atrribute:

- `Error *error_;`

Methoden:

- `MoveCommand(UserInterface *ui, Database *db);`
  - Konstruktor
- `virtual ~MoveCommand();`
  - Destruktor
- `virtual bool execute();`
  - Ruft die move Methode des jeweiligen SVGObjektes auf.

ResizeCommand:

Atrribute:

- `Error *error_;`

Methoden:

- `ResizeCommand(UserInterface *ui, Database *db);`
  - Konstruktor
- `virtual ~ResizeCommand();`
  - Destruktor
- `virtual bool execute();`
  - Ruft die resize Methode des jeweiligen SVGObjektes auf.

RotateCommand:

Atrribute:

- `Error *error_;`

Methoden:

- `RotateCommand(UserInterface *ui, Database *db);`
  - Konstruktor
- `virtual ~RotateCommand();`
  - Destruktor
- `virtual bool execute();`
  - Ruft die rotate Methode des jeweiligen SVGObjektes auf.

ListCommand:

Methoden:

- `ListCommand(UserInterface *ui, Database *db);`
  - Konstruktor
- `virtual ~ListCommand();`
  - Destruktor
- `virtual bool execute();`
  - Gibt eine Liste aller angelegten Objekte aus. Die Liste ist primär nach Gruppennummer und sekundär nach ID in der Gruppe sortiert.

QuitCommand:

Atrribute:

- `SVGDocument *svgdoc_;`

Methoden:

- `ResizeCommand(UserInterface *ui, Database *db, SVGDocument *svgdoc);`
  - Konstruktor
- `virtual ~ResizeCommand();`
  - Destruktor
- `virtual bool execute();`
  - Ruft die Methode setRun() aus dem UserInterface auf und beendet damit die Ausführung des Programmes.

Basisklasse:

SVGObject:

Attribute:

- `UserInterface *ui_;`
- `Database *db_;`
- `SVGHandler *svgh_;`
- `signed int id_;`
- `signed int group_id_;`
- `std::vector<Coordinates> coordinates_;`
- `std::stringstream tag_;`

Methoden:

- `SVGObject (UserInterface* ui, Database *db, SVGHandler *svgh, signed int id, signed int group_id, sdt::vector<Coordinates> coordinates);`
  - Konstruktor
- `virtual ~ SVGObject ();`
  - Destruktor
- `virtual bool move(signed int x, signed int y);`
  - Bewegt das Objekt um die angegebenen x und y Wert.
- `virtual bool resize();`
  - Vergrößert/Verkleinert das Objekt.
- `virtual bool rotate(std::string direction);`
  - Dreht das angegebene Objekt um 90° in die mit angegebene Richtung.
- `virtual std::string getSVGTag();`
  - Gibt den SVG Tag des Objektes zurück .
- `signed int getID();`
  - Gibt die id des Objektes zurück.
- `signed int getGrpID();`
  - Gibt die group des Objektes zurück.
- `std::vector<Coordinates>& getCoordinates();`
  - Gibt die Coordinaten des Objektes zurück.

Davon abgeleitete Klassen:

SVGLine:

Methoden:

- `SVGLine (UserInterface* ui, Database *db, SVGHandler *svgh, signed int id, signed int group_id, std::vector<Coordinates>coordinates);`
  - Konstruktor
- `virtual ~SVGLine ();`
  - Destruktor
- `virtual bool move(signed int x, signed int y);`
  - Bewegt das Objekt um die angegebenen x und y Wert.
- `virtual bool resize();`
  - Vergrößert/Verkleinert das Objekt.
- `virtual std::string getSVGTag();`
  - Gibt den SVG Tag des Objektes zurück .

SVGCircle:

Attribute:

- `signed int r_;`
- `std::string fill_;`
- `std::string stroke_;`
- `unsigned int stroke_width_;`

Methoden:

- `SVGCircle (UserInterface* ui, Database *db, SVGHandler *svgh, signed int id, signed int group_id, std::vector <Coordinates>coordinates, signed int r, std::string fill, std::string stroke, unsigned int stroke_width);`
  - Konstruktor
- `virtual ~ SVGCircle ();`
  - Destruktor
- `virtual bool resize();`
  - Vergrößert/Verkleinert das Objekt.
- `virtual bool rotate();`
  - Gibt true zurück.

- `virtual std::string getSVGTag();`
  - Gibt den SVG Tag des Objektes zurück .

SVGRect:

Attribute:

- `signed int width_;`
- `signed int height_;`
- `std::string fill_;`
- `std::string stroke_;`
- `signed int stroke_width_;`

Methoden:

- `SVGRect (UserInterface* ui, Database *db, SVGHandler *svgh, signed int id, signed int group_id,std::vector <Coordinates> coordinates, signed int width, signed int height, std::string fill, std::string stroke, unsigned int stroke_width);`
  - Konstruktor
- `virtual ~ SVGRect ();`
  - Destruktor
- `virtual bool resize();`
  - Vergrößert/Verkleinert das Objekt.
- `virtual bool rotate(std::string direction);`
  - Rotiert das Objekt um 90° in die mitgegebene Richtung.
- `virtual std::string getSVGTag();`
  - Gibt den SVG Tag des Objektes zurück .

SVGPolygon:

Attribute:

- `std::string fill_;`
- `Error* error;`

Methoden:

- `SVGPolygon (UserInterface* ui, Database *db, SVGHandler *svgh, signed int id, signed int group_id, std::vector <Coordinates> coordinates, std::string fill);`
  - Konstruktor
- `virtual ~ SVGPolygon ();`
  - Destruktor
- `virtual bool move(signed int x, signed int y);`
  - Bewegt das Objekt um die angegebenen x und y Wert.
- `virtual bool resize();`
  - Vergrößert/Verkleinert das Objekt.
- `virtual std::string getSVGTag();`
  - Gibt den SVG Tag des Objektes zurück .

SVGDocument:

Attribute:

- `Error *error_;`
- `Database *db_;`
- `SVGHandler *svgh_;`
- `UserInterface *ui_;`

Methoden:

- `SVGDocument (Database *db, SVGHandler *svgh);`
  - Konstruktor
- `virtual ~SVGDocument();`
  - Destruktor
- `std::string getSVGFooter();`
  - Gibt den Footer tag des SVG Files zurück.
- `std::string getSVGHeader();`
  - Gibt den Header tag des SVG Files zurück.
- `bool writeSVG ();`
  - Schreibt die einzelnen SVG Objekte in das SVGDocument.
- `void setUI(UserInterface *ui);`
  - setzt das handle für das UserInterface.
- `bool checkOpen(std::string filename);`
  - Checkt ob das SVG File bereits offen ist.

Error:

Methoden:

- `Error();`
  - Konstruktor
- `virtual ~Error();`
  - Destruktor
- `inline void usageConfigFile();`
  - Gibt folgende Fehlermeldung aus: "usage: exA -c=configfile\n"
- `inline void outOfMemory();`
  - Gibt folgende Fehlermeldung aus: "error: out of memory\n"
- `inline void cannotReadConfigFile(std::string filename);`
  - Gibt folgende Fehlermeldung aus: "error: cannot read configfile [filename]\n"
- `inline void configFileCorrupt(std::string filename);`
  - Gibt folgende Fehlermeldung aus: "error: configfile [filename] corrupt\n"
- `inline void filenameExists(std::string filename);`
  - Gibt folgende Fehlermeldung aus: "warning: file [filename] exists and will be replaced. Do you want to proceed? (y/n)\n"

- `inline void unknownCommand();`
  - Gibt folgende Fehlermeldung aus: "error: unknown command\n"
- `inline void invalidParameter();`
  - Gibt folgende Fehlermeldung aus: "error: invalid parameter - please enter an integer number\n"
- `inline void idDoesnotExist();`
  - Gibt folgende Fehlermeldung aus: "error: id does not exist\n"
- `inline void idAlreadyExist();`
  - Gibt folgende Fehlermeldung aus: "error: id does already exist\n"
- `inline void cannotWriteToFile(std::string filename);`
  - Gibt folgende Fehlermeldung aus: "error: cannot write to savefile [filename]. Quit anyway? (y/n)\n"

Database:

Attribute:

- `SVGHandler *svgh_;`
- `std::string filename_;`
- `std::string magic_number_;`
- `signed int width_;`
- `signed int height_;`
- `unsigned int svg_object_counter_;`
- `std::vector<SVGObject*> svg_objects_;`
- `std::vector<SVGObject*> group_;`

Methoden:

- `Database(std::string  parameter, SVGHandler* svgh);`
  - Konstruktor

- `virtual ~Database();`
  - Destruktor
- `std::string getFilename()`
  - Gibt den Filenamen der Configfiles zurück
- `signed int getWidth();`
  - Gibt die Weite des SVG Documentes zurück.
- `signed int getHeight();`
  - Gibt die Höhe des SVG Documents zurück.
- `bool checkConfigFile();`
  - Checkt ob das Configfile geöffnet werden kann.
- `bool readConfigFile();`
  - checkt ob das Configfile correct ist.
- `bool checkMagicNumber(char *content);`
  - Prüft ob die MagicNumber richtig ist.
- `bool checkPrompt(char *buffer);`
  - Checkt und setzt den Prompt.
- `bool setSVGDimensions(unsigned char *buffer);`
  - Liest die Dimensionen des SVGFiles und speichert diese als Integer Werte.
- `signed int charToSignedInt(unsigned char* content);`
  - Konvertiert einen Character in einen Integer.
- `bool readFile(unsigned char *buffer, unsigned int lenght, std::ifstream &config_file);`
  - Liest das Configfile aus und überprüft auf Korrektheit.
- `std::vector<SVGObject*>& getAllSVGObjects()`
  - Gibt alle erstellten SVGObjekte zurück.
- `std::vector<SVGObject*>& getSVGObjectByGrp(signed int grp_id);`
  - Gibt einen Vector mit SVGObjekten einer Gruppe zurück.
- `SVGObject* getSVGObjectsByID();`
  - Gibt ein SVGObject zurück.
- `void setSVGObjects(SVGObject* svg_object);`
  - Hängt ein Element an den Vector svg_objects_ an.
- `void sort ();`
  - Sortiert die SVG Objecte in dem Vector svg_objects_.

Coordinates:

Attribute:

- `signed int x_;`
- `signed int y_;`

Methoden:

- `Coordinates(signed int x, signed int y);`
  - Konstruktor
- `virtual ~Coordinates();`
  - Destruktor
- `void setX(signed int x);`
  - Setzt die x Coordinate.
- `void setY(signed int y);`
  - Setzt die y Coordinate.
- `signed int getX();`
  - Gibt die x Coordinate zurück.
- `signed int getY();`
  - Gibt die y Coordinate zurück.

BaseException:

Basisklasse für eine Exception.

ConfigFileException:

Klasse für eine ConfigFile Exception. Wird geworfen wenn es Probleme mit dem Konfigurationsfile gibt. (kein Config File, fehlerhafte Daten …)

Enums:

Constants:

```
A_LOWER = 65,
A_UPPER = 97,
Z_LOWER = 90,
Z_UPPER = 122,
MAX_FILE_SIZE = 8,
MAGIC_SIZE = 5,
PROMPT_SIZE = 4,
SHIFT_CONSTANT = 8
```

ErrorCodes:

```
SUCCESS = 0,
USAGE_CONFIG_FILE = -1,
OUT_OF_MEMORY = -2,
CANNOT_READ_CONFIG_FILE = -3,
CONFIG_FILE_CORRUPT = -4
```

Klassendiagramm