COMP2404
Zach Renaud
Alex Cameron
David McGregor
Feb. 14th 2013

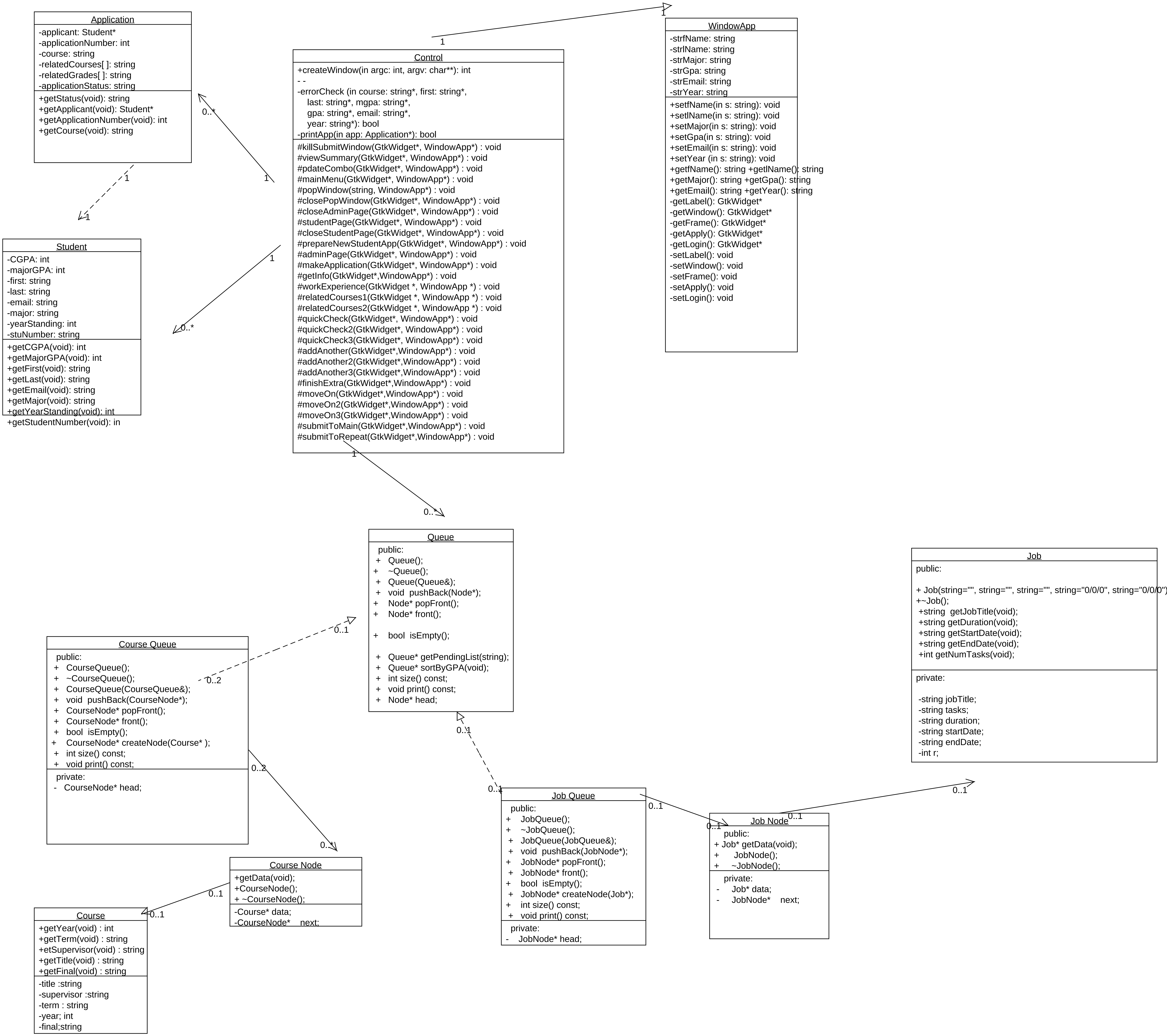
Persistent Storage Implementation

The design pattern that we chose to implement for assignment 2 calls for Objects to be saved and loaded from a file. In particular the central data structure, AppQueue, being a Queue of many Application Objects requires a method by which it can obtain the history of previously entered student applications, upon start-up of the program.

The implementation that we chose saves each of the basic information data members of an application object into a file, written line by line in a specific order. Separated by headers in the document is the list of related courses, related TA positions held and related job experience. As each of these additional related experience fields is maintained by its own class, either Course or Job Object, we decided to take advantage of the Queue framework that we built and store them in their own Queues. Therefore the persistent storage process involved first saving all basic information, such as the information pertaining to the student applicant, followed by headers and header symbols to separate information from each node of the relevant Queue.

The reason we chose to implement the storage in this manner is to promote facility and efficiency of read access to the file, as the information regarding the application queue is centralized and locally managed by the control class alone. The information is organized in such a way that following the basic information of the application we are able to iterate through the fields pertaining to a related course or job until reaching a specified header and moving on to the next queue data member of application. Thus, we are able to rebuild each related experience queue one at a time and add it to the application before finally pushing the application to the application queue. Throughout the life of the program, extending to the load an application function, AppQueue is encapsulated within the control portion of the program.

In conclusion, the implementation chosen for the storage of applications and their related data members, of basic information and related experience, supports read and write access from only one class in the system. This promotes encapsulation and is relatively secure; as each data member is stored as nothing but its raw value. Therefore without knowledge about the data types, the significance of the separator symbols, or the construction process this information is relatively un-usable in terms of replicating system objects.



Application
-applicant: Student*
-applicationNumber: int
-course: string
-relatedCourses[ ]: string
-relatedGrades[ ]: string
-applicationStatus: string
+getStatus(void): string
+getApplicant(void): Student*
+getApplicationNumber(void): int
+getCourse(void): string
Control
+createWindow(in argc: int, argv: char**): int
- -
-errorCheck (in course: string*, first: string*,
last: string*, mgpa: string*,
gpa: string*, email: string*,
year: string*): bool
-printApp(in app: Application*): bool
#killSubmitWindow(GtkWidget*, WindowApp*) : void
#viewSummary(GtkWidget*, WindowApp*) : void
#pdateCombo(GtkWidget*, WindowApp*) : void
#mainMenu(GtkWidget*, WindowApp*) : void
#popWindow(string, WindowApp*) : void
#closePopWindow(GtkWidget*, WindowApp*) : void
#closeAdminPage(GtkWidget*, WindowApp*) : void
#studentPage(GtkWidget*, WindowApp*) : void
#closeStudentPage(GtkWidget*, WindowApp*) : void
#prepareNewStudentApp(GtkWidget*, WindowApp*) : void
#adminPage(GtkWidget*, WindowApp*) : void
#makeApplication(GtkWidget*, WindowApp*) : void
#getInfo(GtkWidget*,WindowApp*) : void
#workExperience(GtkWidget *, WindowApp *) : void
#relatedCourses1(GtkWidget *, WindowApp *) : void
#relatedCourses2(GtkWidget *, WindowApp *) : void
#quickCheck(GtkWidget*, WindowApp*) : void
#quickCheck2(GtkWidget*, WindowApp*) : void
#quickCheck3(GtkWidget*, WindowApp*) : void
#addAnother(GtkWidget*,WindowApp*) : void
#addAnother2(GtkWidget*,WindowApp*) : void
#addAnother3(GtkWidget*,WindowApp*) : void
#finishExtra(GtkWidget*,WindowApp*) : void
#moveOn(GtkWidget*,WindowApp*) : void
#moveOn2(GtkWidget*,WindowApp*) : void
#moveOn3(GtkWidget*,WindowApp*) : void
#submitToMain(GtkWidget*,WindowApp*) : void
#submitToRepeat(GtkWidget*,WindowApp*) : void
WindowApp
-strfName: string
-strlName: string
-strMajor: string
-strGpa: string
-strEmail: string
-strYear: string
+setfName(in s: string): void
+setlName(in s: string): void
+setMajor(in s: string): void
+setGpa(in s: string): void
+setEmail(in s: string): void
+setYear (in s: string): void
+getfName(): string +getlName(): string
+getMajor(): string +getGpa(): string
+getEmail(): string +getYear(): string
-getLabel(): GtkWidget*
-getWindow(): GtkWidget*
-getFrame(): GtkWidget*
-getApply(): GtkWidget*
-getLogin(): GtkWidget*
-setLabel(): void
-setWindow(): void
-setFrame(): void
-setApply(): void
-setLogin(): void
1
1
0..*
1
1
1
1
0..*
Student
-CGPA: int
-majorGPA: int
-first: string
-last: string
-email: string
-major: string
-yearStanding: int
-stuNumber: string
+getCGPA(void): int
+getMajorGPA(void): int
+getFirst(void): string
+getLast(void): string
+getEmail(void): string
+getMajor(void): string
+getYearStanding(void): int
+getStudentNumber(void): in
1
0..*
Queue
public:
+ Queue();
+ ~Queue();
+ Queue(Queue&);
+ void pushBack(Node*);
+ Node* popFront();
+ Node* front();
+ bool isEmpty();
+ Queue* getPendingList(string);
+ Queue* sortByGPA(void);
+ int size() const;
+ void print() const;
+ Node* head;
Job
public:
+ Job(string="", string="", string="", string="0/0/0", string="0/0/0");
+~Job();
+string getJobTitle(void);
+string getDuration(void);
+string getStartDate(void);
+string getEndDate(void);
+int getNumTasks(void);
private:
-string jobTitle;
-string tasks;
-string duration;
-string startDate;
-string endDate;
-int r;
0..1
0..2
Course Queue
public:
+ CourseQueue();
+ ~CourseQueue();
+ CourseQueue(CourseQueue&);
+ void pushBack(CourseNode*);
+ CourseNode* popFront();
+ CourseNode* front();
+ bool isEmpty();
+ CourseNode* createNode(Course*);
+ int size() const;
+ void print() const;
private:
- CourseNode* head;
0..2
0..*
0..1
0..1
Job Queue
public:
+ JobQueue();
+ ~JobQueue();
+ JobQueue(JobQueue&);
+ void pushBack(JobNode*);
+ JobNode* popFront();
+ JobNode* front();
+ bool isEmpty();
+ JobNode* createNode(Job*);
+ int size() const;
+ void print() const;
private:
- JobNode* head;
0..1
0..1
0..1
0..1
Job Node
public:
+ Job* getData(void);
+ JobNode();
+ ~JobNode();
private:
- Job* data;
- JobNode* next;
Course Node
+getData(void);
+CourseNode();
+ ~CourseNode();
-Course* data;
-CourseNode* next;
0..1
0..1
Course
+getYear(void) : int
+getTerm(void) : string
+etSupervisor(void) : string
+getTitle(void) : string
+getFinal(void) : string
-title :string
-supervisor :string
-term : string
-year; int
-final;string