Object Oriented Programming Lecture

Persistence using Files: Random Access Files

School of Computing and Information Technology
Faculty of Engineering and Computing
University of Technology, Jamaica

Email: dwwhite@utech.edu.jm, taedwards@utech.edu.jm

Object Oriented Programming Concepts

Expected Outcome

At the end of this lecture the student should be able to:

- Define and understand object persistence
- Implement object persistence using files and streams
- Understand the difference between sequential and random access files and be able to use both in a program to achieve persistence

Object Oriented Programming Concepts

Topics to be covered in this lecture:

- Characteristics of Random Access Files
- Comparison Sequential and Random Access Files
- Fixed Record Size Requirement
- Working with Random Access Files
  - Create/Initialize File
  - Open File
  - Store Record
  - Retrieve Record
  - Delete Record
  - Update Record

Characteristics: Random Access files

- Requires an initialization phase
- Records are stored in a random manner
- Records can be added anywhere in the file
- The required record can be accessed immediately without reading other records
- Record deletion requires marking a record as deleted
- Can be very fast even for files with large amounts of records
- File size might be much larger than sequential access file
- Well-suited for applications where records may change often and records may need to be retrieved in a random order, e.g. a parts inventory system which allows a user to query the price and description of any item in the inventory

Summary of Features of Sequential Access vs. Random Access Files

Characteristics	**Sequential Access**	**Random Access**
Requires Initialization	No	Yes
Storing new records	Done at end of file	Done any where in the file
Access	Accessed in order from beginning of file until required record is reached	Direct access to required record without having to start from beginning of file
Speed of access	Slow	Fast
Record deletion	Requires second file	Deleted records marked as deleted but not actually removed
File size	smaller	larger
Applications best suited for	Where records are usually accessed together and which do not change often	Where records are required to be accessed instantly and/or data may change often

Fixed Record Size

- All records must use a fixed amount of memory, variable length records should not be used
- Determining Memory requirements
- Each data type requires a specific amount of memory
  - int 4 bytes
  - float 4 bytes
  - double 8 bytes
  - char 1 byte - C++
  - char 2 bytes – Java
- User defined data types require memory based on the composition of the data members.

Fixed Record Size

- For example a user class Pet with attributes:
  - int id;
  - String type;
  - char gender;
  - double cost;

- C++: id(4) + type(20) + gender(1) + cost(8) = 33 bytes
- Java: id(4) + type(20*2) + gender(1*2) + cost(8) = 54 bytes

Fixed Record Size

- Each record stored in a random access file has a starting byte and an ending byte.
- Formulae exists to calculate the byte values;
- Starting Byte:
- SB = (KFV – OFFSET) * Memory Used by Object
- KFV – Key Field Value, this is the unique value used to represent a record, normally an Identifier like customer number or employee id
- OFFSET – Should be the first valid KFV that can be assigned to a record; e.g. Employee id range 500 – 599, the OFFSET will be 500.
- EB = SB + (Memory Used by Object - 1)

Fixed Record Size

- Determine the starting byte and ending byte for the following:
- C++
- Pet p(4, “Dog”, ‘M’, 24500.00)
- SB = 4 – 1 * 33 = 3 * 33 = 99
- EB = 99 + (33 – 1) = 131
- Java
- Pet p = new Pet(4, “Dog”, ‘M’, 24500.00);
- SB = 4 – 1 * 54 = 3 * 54 = 162
- EB = 162 + (54 – 1) = 215

Fixed Record Size

- Given that records are of a fixed size then the size of a initialized random access file can easily be calculated.
- Determine the size of the initialized file based on the following:
- A file to store 500 pet records
- File Size = Memory Used by Object * Number of records
- C++
- File Size = 33 * 500 = 16,500 bytes
- Java
- File Size = 54 * 500 = 27,000 bytes

Fixed Record Size

- Determine the number of records stored in a file based on file size.
- Number of records = File Size / Memory Used by Object
- C++
- Number of records = 16,500 / 33 = 500
- Java
- Number of records = 27,500 / 54 = 500

Create/Initialize data in a random access file

C++

```
.int max = 100;
.Employee emp;//object created using default constructor
.try{
.ofstream raFile("employee.dat",ios::out|ios::binary);
.if(raFile.fail()){
.throw runtime_error("cannot create database");
.}
```

Create/Initialize data in a random access file

.C++ con't

```
.for(int idx = 1; idx <= max; idx++){

.raFile.seekp((idx – 1) * sizeof(Employee));

.raFile.write(reinterpret_cast <const char *> &emp,
sizeof(emp));

.}

.raFile.close();

.}catch(runtime_error &e){

.cerr << e.what() << endl

.}
```

Create/Initialize data in a random access file

- **Java**
- `int max = 100;`
- `Employee emp = new Employee();//object created using default constructor`
- `RandomAccessFile raFile = null;`
- `try{`
- file = new RandomAccessFile(“employee.dat”, “rw”);
- for(int idx = 1; idx <= max; idx++){
- raFile.seek((idx – 1) * (4 + (25*2) + (25*2));

Create/Initialize data in a random access file

- **Java con't**
- raFile.writeInt(emp.getId ());
- raFile.writeUTF(emp.getFirstName());
- raFile.writeUTF(emp.getLastName());
- }}//end try
- }catch(IOException e){e.printStackTrace();
- }finally{
- try{
- raFile.close();
- }catch(IOException e){
- }
- }

Opening a random access file

C++

- ifstream raFile(”employee.dat",ios::in|ios::binary);
- ofstream raFile(”employee.dat",ios::out|ios::binary);
- fstream raFile(”employee.dat",ios::in|ios::out|ios::binary);

- **Java**
- RandomAccessFile raFile = new RandomAccessFile(“RAF”, "rw");
- File file = new File(”employee.dat");
- RandomAccessFile raFile = new RandomAccessFile(file, "r");
- RandomAccessFile raFile = new RandomAccessFile(file, "rw");

Storing data in a random access file

C++

- raFile.seekp((emp.getId() – 1) * sizeof(emp));
- raFile.write(reinterpret_cast <const char *> &emp, sizeof(emp));

- **Java**
- raFile.seek((emp.getId() – 1) * (4 + (25*2) + (25*2));
- raFile.writeInt(emp.getId ());
- raFile.writeUTF(emp.getFirstName());
- raFile.writeUTF(emp.getLastName());

Retrieving data from a random access file

C++

- raFile.seekg((searchId – 1) * sizeof(emp));
- raFile.read(reinterpret_cast <char *> &emp, sizeof(emp));

- **Java**
- raFile.seek((searchId() – 1) * (4 + (25*2) + (25*2));
- int id = raFile.readInt();
- String firstName = raFile.readUTF();
- String lastName = raFile.readUTF();

Deleting data from a random access file

C++

- Employee emp;//create object using default constructor
- raFile.seekp((searchId – 1) * sizeof(emp));
- raFile.write(reinterpret_cast <const char *> &emp, sizeof(emp));

- **Java**
- Employee emp = new Employee();//create object using default constructor
- raFile.seek((searchId – 1) * (4 + (25*2) + (25*2));
- raFile.writeInt(emp.getId());
- raFile.writeUTF(emp.getFirstName());
- raFile.writeUTF(emp.getLastName());

Updating data from a random access file

C++

- int searchId = 34;
- Employee emp = retrieveRecord(searchId);
- emp.edit();
- storeRecord(emp);

- **Java**

- int searchId = 34;
- Employee emp = retrieveRecord(searchId);
- emp.edit();
- storeRecord(emp);