Object Oriented Programming Lecture

Persistence using Files: Sequential 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

Expected Outcome

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

- explain object persistence
- summarize the use of files and streams in object persistence
- explain the use of sequential access file in object persistence
- implement object persistence using sequential access files in an object oriented programming language

Object Oriented Programming

Topics to be covered in this lecture:

- Persistence
- Streams
- File Streams
- File Organization
- File Access Methods
- Characteristics of Sequential Access Files
- Working with Sequential Access Files

Persistence

- When an object of a class is instantiated (created), it resides in main memory (RAM) of the computer the program that created it is running on.
- Normally, as soon as the program terminates, the objects created by the program are removed from RAM.

Persistence

- Persistence is the ability of an object to continue existence after the program that created the object terminates, and is removed from RAM.
- Persistence is achieved by:
  - Storing data from an object in a file
  - Storing data from an object in a database

Persistence

- Some object-oriented database management systems (OODBMS) allow entire objects to be stored and retrieved in a database through use of object oriented programming languages like C++ and Java.
- In this course we will use files to achieve object persistence

Streams

- A stream is a flow of bytes to or from a device, a network connection or a file.
- A stream may be one-way such as a stream associated with an output device such as a display screen or a stream associated with an input device such as a keyboard.

Streams

- A stream may also be multi-directional such as a stream associated with a file or a network connection.
- Some OOP languages provide stream classes which programmers can use to create stream objects.
- These stream objects can be used to open and manipulate data in files.

Streams

- Predefined stream objects are available in both C++ and Java
- These predefined stream objects are:
  - C++: cout and cin
  - Java: System.out and System.in
- These stream objects are associated with the default output (Monitor) and input (Keyboard) devices.

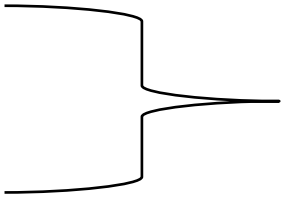
Streams

- There is also a special predefined stream object available in both C++ and Java, dedicated for use when logging program errors.
- This predefined stream object is:
  - C++: cerr
  - Java: System.err
- This stream object is associated with the default output device.

Streams

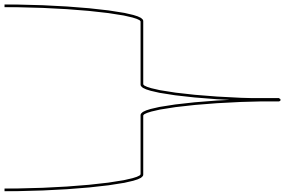
cout — Standard output stream object in C++

System.out — Standard output stream object in Java



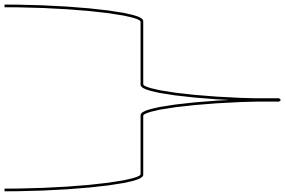
cin — Standard input stream object in C++

System.in — Standard input stream object in Java



cerr — Standard error stream object in C++

System.err — Standard error stream object in Java



Stream Input

C++

```
#include <string>
#include <iostream>
using namespace std;

int a;
cin >> a;


int a, b, c;
cin >> a >> b >> c;

string address;

getline(cin, address);
```

Java

```
import java.util.Scanner;

int a;
Scanner in = new
Scanner(System.in);
a = in.nextInt();

double d;
Scanner inp = new
Scanner(System.in);
d = inp.nextDouble();

Scanner X = new Scanner(System.in);

String address;
address = X.nextLine();

String word;
word = X.next();
```

Stream Input

Java

```
import java.io.BufferedReader;

InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
String s = br.readLine();


int Value = 0;
String FullLine = null;
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
FullLine = br.readLine();
Value = Integer.parseInt(FullLine);
```

Stream Output

C++

```
#include <iostream>
using namespace std;

int a = 9;
cout << a;


int a = 3;
int b = 4;
cout << “a is ” << a << “, b is ” << b
        << endl;


cout << “Welcome to our world!” << endl;

cerr << “An error occurred.”;
```

Java

```
int a = 9;
System.out.print(“ ” + a);


int a = 3;
int b = 4;
System.out.println(“a is ” + a + “, b is
” + b);


System.out.println(
        “Welcome to our world!”);

System.err.println(
        “An error occurred.”);
```

File Streams

- A file stream is a flow of bytes to or from a file.
- A file stream may be one-way or two-way depending on the mode used to open the stream
- Read-only: one-way, data is read from the file and used as input to a program

File Streams

- Write-only: one-way, data output from a program is used as input to a file.
- Read-Write: two-way, data from the file is used as input for the program, and data output from the program is used as input to the file.

File Streams

- There classes available in both C++ and Java to support file processing.
- A few of these classes are:
  - C++: ofstream, ifstream and fstream
  - Java: FileWriter, FileReader, RandomAccessFile and Scanner
- NB: C++ classes use the “fstream” library, while in Java use the packages “java.io” and “java.util”

File Streams

- A file can be considered as a sequence of bytes
- To manipulate files in OOP:
  - A file stream object is associated with the file when the file is opened
  - Data from the file may be used as input for the program
  - Data from the program may be sent as output to the file

File Organization

Bytes

- A byte is a group of 8 binary digits (bits)

Fields

- A field is a group of bytes

Records

- A record is a collection of related fields

Files

- A file is a collection of related records

Database

- A database is a collection of related files.

File Organization

- **For example, in a student information system:**
  - **Bytes**
    - Series of bytes such as “01000100” may make up each field
  - **Fields**
    - First Name, Last Name, Id number, would each be fields. e.g. First Name = “David”

File Organization

- **Records**
  - Each student would have a separate record, which would represent the group of fields for that student
  - e.g. “David”, “White”, 123456789 would be one record
  - e.g. “Louise”, “Palmer”, 230471986 would be another

File Organization

Files

- The individual records for David, Louise and other students might be stored in a student master file. A grades file might store records contain the grade of each student, and still another file might contain each students progress report.

File Organization

Database

- The student information database would contain the various files that are a part of the system, such as the student master file, the grades file, and the progress report file.

File Access Methods

- Files are often classified by how there are accessed.
- Two common modes of access are:
  - Sequential Access
  - Random Access
- Files belonging to each of the above classification types are processed differently

Sequential Access File: Characteristics

- Records are stored sequentially
- Records are added (appended) at the end
- Records are always accessed beginning with the first record, then processed sequentially until the required record is found or all the records are processed

Sequential Access File: Characteristics

- Record deletion requires a temporary file
- Might be slow for files with large amounts of records
- File size might be smaller than random access file

Sequential Access File: Characteristics

- Well-suited for applications involving records that are always processed sequentially and the records do not change often, e.g. a payroll system when all the salaried employees are paid each month

Sequential Access File: Creation

C++

Requires the fstream library

ofstream sOutFile("emp-db.sq2",ios::out);

Java

Requires the java.io package library

FileWriter sOutFile = new FileWriter("emp-db.sq2",false);

.Creates a new text file to accept output from the program

NB: In both scenarios, if a file already exists with the same name, then the file will be truncated (i.e. over-riden).

Sequential Access File: Data Output

C++

```
Employee emp(1011, “John”, “Wayne”);

sOutFile << emp.getId() << “\t” << emp.getFirstName() << “\t” <<
emp.getLastName() << endl;

sOutFile.close();
```

Java

```
Employee emp = new Employee(1011, “John”, “Wayne”);

String record = emp.getId() + “\t” + emp.getFirstName() + “\t” +
emp.getLastName() + “\n”;

sOutFile.write(record);

sOutFile.close();
```

Sequential Access File: Data Retrieval – First Record

C++

```
ifstream sInFile("emp-db.sq2",ios::in);
int empId;
string firstName, lastName;
sInFile >> empId >> firstName >> lastName;
Employee rec(empId, firstName, lastName);
rec.show();
sInFile.close();
```

Sequential Access File: Data Retrieval – First Record

Java

```
Scanner sInFile = new Scanner(new File("emp-db.sq2"));
int empId;
String firstName, lastName;
empId = sInFile.nextInt();
firstName = sInFile.next();
lastName = sInFile.next();
Employee rec = new Employee(empId, firstName, lastName);
System.out.println(rec);
```

Sequential Access File: Data Retrieval – All Records

C++

```
ifstream sInFile("emp-db.sq2",ios::in);
int empId;
string firstName, lastName;
while(!sInFile){
        sInFile >> empId >> firstName >> lastName;
        Employee rec(empId, firstName, lastName);
        rec.show();
}
sInFile.close();
```

Sequential Access File: Data Retrieval – All Records

Java

```
Scanner sInFile = new Scanner(new File("emp-db.sq2"));
int empId;
String firstName, lastName;
while(sInFile.hasNext()){
        empId = sInFile.nextInt();
        firstName = sInFile.next();
        lastName = sInFile.next();
        Employee rec = new Employee(empId, firstName, lastName);
        System.out.println(rec);
}
sInFile.close();
```

Sequential Access File: Data Search

C++

```
ifstream sInFile("emp-db.sq2",ios::in);
int empId;
string firstName, lastName;
bool found = false;
while(!sInFile){
        sInFile >> empId >> firstName >> lastName;
        if(empId == searchId){
                found = true;
                break;
        }
}
sInFile.close();
```

Sequential Access File: Data Search

Java

```
Scanner sInFile = new Scanner(new File(“emp-db.sq2”));
int empId;
String firstName, lastName;
boolean found = false;
while(sInFile.hasNext()){
        empId = sInFile.nextInt();
        firstName = sInFile.next();
        lastName = sInFile.next();
        if(empId == searchId){
                found = true;
                break;
        }
}
sInFile.close();
```

Sequential Access File: Data Deletion

C++

```
ifstream sInFile("emp-db.sq2",ios::in);
ofstream sOutFile("tempEmp.sq2",ios::out);
int empId;
string firstName, lastName;
while(!sInFile){
        sInFile >> empId >> firstName >> lastName;
        if(empId != deleteId){
                sOutFile << empId << "\t" << firstName << "\t" << lastName
<< endl;
        }
}
sInFile.close();
sOutFile.close();
```

Sequential Access File: Data Deletion

Java

```
Scanner sInFile = new Scanner(new File(“emp-db.sq2”));
FileWriter sOutFile = new FileWriter(“tempEmp.sq2”,false);
int empId;
String firstName, lastName;
while(sInFile.hasNext()){
        empId = sInFile.nextInt();
        firstName = sInFile.next();
        lastName = sInFile.next();
        if(empId != deleteId){
                sOutFile.write(empId + “\t” + firstName + “\t” + lastName + “\n”);
        }
}
sInFile.close();
sOutFile.close();
```

Sequential Access File: Example

C++

```
void saveRecord(Employee data){
   try{
      ofstream sOutFile("emp-db.sq2", ios::app);
      if(sOutFile.fail()){
         throw runtime_error("Error access database!");
      }
      sOutFile << data.getId() << "\t" << data.getFirstName() << "\t" <<
data.getLastName() << endl;
      sOutFile.close();
   }catch(runtime_error &err){
      cerr << err.what() << endl;
   }
}
```

Sequential Access File: Example

Java

```
public void saveRecord(Employee data){
   FileWriter sOutFile = null;
   try{
      sOutFile = new FileWriter("emp-db.sq2",true);
      String rec = data.getId() + "\t" + data.getFirstName() + "\t" + data.getLastName() + "\n";
      sOutFile.write(rec);
   }catch(IOException ioex){
      System.err.println(ioex.getMessage());
   }finally{
      try{
         sOutFile.close();
      }catch(IOException ioex){
         //Necessary to prevent resource leak;
      }
   }
}
```