File Conversion Program

Using Common File Control and Object Class

You have the following data in an Excel worksheet that needs to be exported and converted.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Data Type** | **Number Decimals** | **Length** | **Notes** |
| Student ID Number | Integer | 0 | 7 |  |
| Name | String | N/A | 30 |  |
| Major | String | 0 | 30 |  |
| Grade Point Average | Float | 2 | 6 |  |
| Number Credits Taken | Float | 2 | 6 |  |
| Number Credits Earned | Float | 2 | 6 |  |
| College Year | Character | N/A | 1 | 0 = Admitted 1= Freshman  2=Sophomore  3=Junior  4=Senior |
| Is On Financial AId | Boolean | N/A |  | true – on financial aid  false – not on financial aid |

The file will be exported as a comma separated values text file and converted to a binary file. The text file (CSV) file name is student.csv and the binary file name is student.dat. Display the student's id number, name, major, and gpa on the monitor.

Planning Document – Student Class

**Actions (2 points):**

Create instance fields

Create Assessor/Mutator Methods

Create Display method

Create Parse method

**Data Items (3 points):** < This is a list of fields (variables, constants, and objects you will need >

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Item** | **Access (1)** | **Type (2)** | **Identifier** | **Notes** |
| Student ID Number | private | integer | studentID |  |
| Name | private | string | name |  |
| Major | private | string | major |  |
| Grade Point Average | private | float | gpa |  |
| Number Credits Taken | private | float | creditsTaken |  |
| Number Credits Earned | private | float | creditsEarned |  |
| College Year | private | character | collegeYear |  |
| Is On Financial Aid | private | boolean | isFinancialAid |  |
| Input Line | local | string | inputLine | parse method |
| succeeded | local | boolean | succeeded | parse method |

1. Access: public, private, protected

2. Type: string, char, byte, short, integer, long, double, float, etc.

**Sample Output (5 Points):**

Student ID: 999999999

Name: X----------------------------X

Major: X----------------------------X

Grade Point Average: 99.99

Number Credits Taken: 999.99

Number Credits Earned: 999.99

College Year: X

Is On Financial AId: X----X

**Test Data (5 points):** < How will you prove your program works >

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier** | Value 1 | Value 2 | Value 3 | Value 4 |
| studentID | 12345 |  |  |  |
| name | Gary Smith |  |  |  |
| major | CIS |  |  |  |
| gpa | 3.85 |  |  |  |
| creditsTaken | 52 |  |  |  |
| creditsEarned | 51 |  |  |  |
| collegeYear | 2 |  |  |  |
| isFinancialAid | false |  |  |  |

Note: You made more or fewer test cases depending on your application.

**Process Design (10 points):** < What is the solution (pseudocode, IPO diagram, flowchart) >

NOTE: Please see the pseudocode syntax summary document for guidelines. *[ and ] symbols indicate an optional value and is not included as part of the code. Items in yellow must be specified. Items in cyan, are program statement(s).*

class Student

private integer studentID

private string name

private string major

private float gpa

private float creditsTaken

private float creditsEarned

private character collegeYear

private boolean isFinancialAid

public Student() // Default constructor

return

public Student(string inputLine) // Constructor for reading and parsing an input line

parse(inputLine)

return

// Accessor/Mutator metods

public integer getStudentID()

return studentID

public void setStudentID(integer value)

studentID = value

return

public string getName()

return name

public void setName(string value)

name = value

return

public string getMajor()

return major

public void setMajor(string value)

major = value

return

public float getGPA()

return gpa

public void setGPA(float value)

gpa = value

return

public float getCreditsTaken()

return creditsTaken

public void setCreditsTaken(float value)

craditsTaken = value

retirm

public float getCreditsEarned()

return craditsEarned

public void setCreditsEarned(float value)

creditsEarned = value

return

public charcter getCollegeYear()

return collegeYear

public void setCollegeYear(charcter value)

collegeYear = value

return

public boolean getIsFinancialAid()

return isFinancialAid

public void setIsFinancialAid(boolean value)

isFinancialAid = value

return

public void displayData()

display "Student ID: " + getStudentID()

display "Name: " + getName()

display "Major: + getMajor()

display "Grade Point Average: " + getGPA()

display "Number Credits Taken: " + getCreditsTaken()

display "Number Credits Earned: " + getCreditsEarned()

display "College Year: " + getCollegeYear()

display "Is Financial AId: " + getIsFinancialAid()

return

public boolean parse(string inputLine)

boolean succeeded = false

string[] splitFields

splitFields = inputLine.split(",")

try

studentID = Convert.toInteger(splitFIelds[0])

name = splitFIelds[1]

major = splitFIelds[2]

gpa = Convert.toFloat(splitFIelds[3])

creditsTaken = Convert.toFloat(splitFIelds[4])

creditsEarned = Convert.toFloat(splitFIelds[5])

collegeYear = Convert.toCharacter(splitFields[6])

isFinancialAid = Convert.toBoolean(splitFIelds[7])

succeeded = true

endTry

catch (Exception err)

err.getMessage()

display "Error parsing input record." + inputLine

succeeded = false

throw new DataFormatException()

endCatch

return succeeded

endClass

File Control Class for Student Comma Separated Values File

**Actions (2 points):**

Create methods to handle file operations:

openRead() – opens the file for reading

openAppend() – opens the file for adding records

openOutput() – opens the file for output (overwrite)

close() – closes the file

readRecord()- reads a record from the file

write() – writes a record to the file

**Data Items (3 points):** < This is a list of fields (variables, constants, and objects you will need >

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Item** | **Access (1)** | **Type (2)** | **Identifier** | **Notes** |
| Student class | public | Student | data |  |
| Stream Reader | private | StreamReader | reader |  |
| Stream Writer | private | SteamWriter | writer |  |
| File class | private | File | file |  |
| File Name | private | string | filename |  |
| File is Opened | public | boolean | isOpen |  |
| End of File | public | boolean | isEOF |  |
| Input Line | local | string | inputLine |  |
| Success Flag | local | boolean | succeeded |  |

1. Access: public, private, protected

2. Type: string, char, byte, short, integer, long, double, float, etc.

**Sample Output (5 Points):**

Only error messages will be displayed.

**Test Data (5 points):** < How will you prove your program works >

Test each method to insure it performs the correct function.

**Process Design (10 points):** < What is the solution (pseudocode, IPO diagram, flowchart) >

public class FileStudentCSV

// Data Object

public Student data = new Student()

// File handling fields

private StreamReader reader

private StreamWriter writer

private constant string filename = "..\\..\\data\\students.csv"

private File = new File(filename)

public boolean isOpen = false

public boolean isEOF = false

// Opens the Items file for reading.

public void openRead()

// Check to see if file exists. If so open the file.

if (File.exists(filename))

try

// Open the file. This allows the file to be

// opened for shared read access.

reader = new StreamReader(filename)

isOpen = true

endTry

catch (IOException err)

MessageBox.Show(err.Message, "File system error",

MessageBoxButtons.OK, MessageBoxIcon.Error)

isOpen = false

endCatch

else

// Display message and terminate.

throw new FileNotFoundException("File not found")

endIf

return

// Opens the Items file for adding records.

public void openAppend()

// Check to see if file exists. If so open the file.

if (File.exists(filename))

try

// Open the file.

writer = new StreamWriter(filename, true)

isOpen = true

endTry

catch (IOException err)

MessageBox.Show(err.Message, "File system error",

MessageBoxButtons.OK, MessageBoxIcon.Error)

isOpen = false

endCatch

else

// Display message and terminate.

throw new FileNotFoundException("File not found")

endIf

return

// Opens the Items file for writing records.

public void openOutput()

// Check to see if file exists. If so open the file.

if (File.exists(filename))

try

// Open the file.

writer = new StreamWriter(filename, false)

isOpen = true

endTry

catch (IOException err)

MessageBox.Show(err.Message, "File system error",

MessageBoxButtons.OK, MessageBoxIcon.Error)

isOpen = false

endCatch

else

// Display message and terminate.

throw new FileNotFoundException("File not found")

endIf

return

// Close the file

public void close()

if (Not(reader == null)) then

reader.close()

endIf

if (Not(writer == null)) then

writer.close()

endIf

return

// Read a record from the file and populate the class.

public boolean readRecord()

boolean succeeded = false

// Read the next line.

string inputLine = reader.readLine()

// Null items denotes end of file.

if (inputLine == null)

isEOF = true

succeeded = true

else

try

// Create an instance of the class and populate the class from the input line.

data = new Student(inputLine)

endTry

catch (Exception err)

succeeded = false

endCatch

endIf

return succeeded

// Writes a student record in text format.

public boolean writeRecord()

boolean succeeded = false

try

writer.write(

data.getStudentID() + "," +

data.getName() + "," +

data.getMajor() + "," +

data.getGPA() + "," +

data.getCreditsTaken()+ "," +

data.getCreditsEarned()+ "," +

data.getCollegeYear()+ "," +

data.getIsFinancialAid()

)

writer.writeLine();

succeeded = true

endTry

catch (Exception err)

MessageBox.Show(err.Message + "\n" + err.StackTrace, "Failure to write to file",

MessageBoxButtons.OK, MessageBoxIcon.Error)

endCatch

return succeeded

endClass

File Control Class for Student Binary File

**Actions (2 points):**

Create methods to handle file operations:

openRead() – opens the file for reading

openAppend() – opens the file for adding records

openOutput() – opens the file for output (overwrite)

close() – closes the file

readRecord()- reads a record from the file

write() – writes a record to the file

**Data Items (3 points):** < This is a list of fields (variables, constants, and objects you will need >

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Item** | **Access (1)** | **Type (2)** | **Identifier** | **Notes** |
| Student class | public | Student | data |  |
| Binary Reader | private | BinaryReader | reader |  |
| Binary Writer | private | BinaryWriter | writer |  |
| File class | private | File | file |  |
| File Name | private | string | filename |  |
| File is Opened | public | boolean | isOpen |  |
| End of File | public | boolean | isEOF |  |
| Success Flag | local | boolean | succeeded |  |

1. Access: public, private, protected

2. Type: string, char, byte, short, integer, long, double, float, etc.

**Sample Output (5 Points):**

Only error messages will be displayed.

**Test Data (5 points):** < How will you prove your program works >

Test each method to insure it performs the correct function.

**Process Design (10 points):** < What is the solution (pseudocode, IPO diagram, flowchart) >

public class FileStudentBIN

// Data Object

public Student data = new Student()

// File handling fields

private BinaryReader reader

private BinaryWriter writer

private constant string filename = "..\\..\\data\\students.dat"

private File file = new File(filename)

public boolean isOpen = false

public boolean isEOF = false

// Opens the Items file for reading.

public void openRead()

// Check to see if file exists. If so open the file.

if (File.exists(filename))

try

// Open the file.

reader = new BinaryReader(filename)

isOpen = true

endTry

catch (IOException err)

MessageBox.Show(err.Message, "File system error", MessageBoxButtons.OK,

MessageBoxIcon.Error)

isOpen = false

endCatch

else

// Display message and terminate.

throw new FileNotFoundException("File not found")

endIf

return

// Opens the Items file for adding records.

public void openAppend()

// Check to see if file exists. If so open the file.

if (File.exists(filename))

try

// Open the file.

writer = new BinaryWriter(filename, true)

isOpen = true

endTry

catch (IOException err)

MessageBox.Show(err.Message, "File system error", MessageBoxButtons.OK,

MessageBoxIcon.Error)

isOpen = false

endCatch

else

// Display message and terminate.

throw new FileNotFoundException("File not found")

endIf

return

// Opens the Items file for writing records.

public void openOutput()

// Check to see if file exists. If so open the file.

if (File.exists(filename))

try

// Open the file.

writer = new BinaryWriter(filename, false)

isOpen = true

endTry

catch (IOException err)

MessageBox.Show(err.Message, "File system error", MessageBoxButtons.OK,

MessageBoxIcon.Error)

isOpen = false

endCatch

else

// Display message and terminate.

throw new FileNotFoundException("File not found")

endIf

return

// Close the file

public void close()

if (Not(reader == null)) then

reader.close()

endIf

if (Not(writer == null)) then

writer.close()

endIf

return

// Read a record from the file and populate the class.

public boolean readRecord()

boolean succeeded = false

// Create an instance of the class and populate the class from input data.

data = new Student()

try

data.setStudentID(reader.readInteger())

data.setName(reader.readString())

data.setMajor(reader.readString())

data.setGPA(reader.readFloat())

data.setCreditsTaken(reader.readFloat())

data.setCreditsEarned(reader.readFloat())

data.setCollegeYear(reader.readCharacter())

data.setIsFinancialAid(reader.readBoolean())

succeeded = true

endTry

catch (EndOfFileException err)

isEOF = true

succeeded = true

endCatch

catch (Exception err)

MessageBox.Show(err.Message + "\n" + err.StackTrace, "Error reading file",

MessageBoxButtons.OK, MessageBoxIcon.Error)

succeeded = false

throw new DataFormatException()

endCatch

return succeeded

// Writes student record in binary format.

public boolean writeRecord()

boolean succeeded = false

try

writer.writeInteger(data.getStudentID())

writer.writeString(data.getName())

writer.writeString(data.getMajor())

writer.writeFloat(data.getGPA())

writer.writeFloat(data.getCreditsTaken())

writer.writeFloat(data.getCreditsEarned())

writer.writeChacter(data.getCollegeYear())

writer.writeBoolean(data.getIsFinancialAid())

succeeded = true

endTry

catch (Exception err)

MessageBox.Show(err.Message + "\n" + err.StackTrace, "Failure to write to file",

MessageBoxButtons.OK, MessageBoxIcon.Error)

endCatch

return succeeded

endClass

Planning Document - Convert Student

**Actions (2 points):**

Open files

Read and parse input record

write binary record

display student id, name, major and gpa on monitor

close files

**Data Items (3 points):** < This is a list of fields (variables, constants, and objects you will need >

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Item** | **Access (1)** | **Type (2)** | **Identifier** | **Notes** |
| File Control Student CSV | private | FileStudentCSV | studentIn |  |
| File Control Student BIN | private | FileStudentBIN | studentOut |  |

1. Access: public, private, protected

2. Type: string, char, byte, short, integer, long, double, float, etc.

**Sample Output (5 Points):**

Student ID Name Major GPA

99999999 X-----------------------------------X X-----------------------------------X 9.99

**Test Data (5 points):** < How will you prove your program works >

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier** | Value 1 | Value 2 | Value 3 | Value 4 |
| studentID | 12345 |  |  |  |
| name | Gary Smith |  |  |  |
| major | CIS |  |  |  |
| gpa | 3.85 |  |  |  |
| creditsTaken | 52 |  |  |  |
| creditsEarned | 51 |  |  |  |
| collegeYear | 2 |  |  |  |
| isFinancialAid | false |  |  |  |

Note: You made more or fewer test cases depending on your application.

**Process Design (10 points):** < What is the solution (pseudocode, IPO diagram, flowchart) >

NOTE: Please see the pseudocode syntax summary document for guidelines. *[ and ] symbols indicate an optional value and is not included as part of the code. Items in yellow must be specified. Items in cyan, are program statement(s).*

class ConvertStudent

private FileStudentCSV studentIn = new FileStudentCSV()

private FileStudentBIN studentOut = new FileStudentBIN()

void main()

startup()

while Not studentIn.isEOF

processing()

end while

shutdown()

return

void startup()

studentIn.openRead()

studentOut.openOutput()

if (Not (studentIn.isOpen And studentOut.isOpen)

shutdown()

endIf

display displayHeading()

studentIn.readRecord()

return

void processing()

studentOut.data = studentIn.data // Points the output object to the input object.

studentOut.writeRecord()

displayRecord()

studentIn.readRecord()

return

void shutdown()

studentIn.close()

studentOut.close()

return

void displayHeading()

display " Student ID Name Major GPA"

return

void displayRecord()

display studentIn.data.getStudentID() + " " + studentIn.data.getName() + " " +

studentIn.data.getMajor() + " " + studentIn.data.getGPA()

return

endClass