



# Object Relational Mapping



# Review

We have developed our UAMS using 3 Tier Model.  
i.e., separate

1. Business Logic (BL)
2. Data Layer (DL)
3. User Interface (UI)

# Review

We made 3 classes in the business logic.

```
class Student
{
    public string name;
    public int age;
    public double fscMarks;
    public double ecatMarks;
    public double merit;
    public List<DegreeProgram> preferences;
    public List<Subject> regSubject;
    public DegreeProgram regDegree;
}
```

```
class Subject
{
    public string code;
    public string type;
    public int creditHours;
    public int subjectFees;
}
```

```
class DegreeProgram
{
    public string degreeName;
    public float degreeDuration;
    public List<Subject> subjects;
    public int seats;
}
```

# Review

Now, the question is how to **Store** the data related to objects in the Files.

```
class Student
{
    public string name;
    public int age;
    public double fscMarks;
    public double ecatMarks;
    public double merit;
    public List<DegreeProgram> preferences;
    public List<Subject> regSubject;
    public DegreeProgram regDegree;
}
```

```
class Subject
{
    public string code;
    public string type;
    public int creditHours;
    public int subjectFees;
}
```

```
class DegreeProgram
{
    public string degreeName;
    public float degreeDuration;
    public List<Subject> subjects;
    public int seats;
}
```

# Review

DegreeProgram objects contain the list of subjects which are also objects.

```
class Student
{
    public string name;
    public int age;
    public double fscMarks;
    public double ecatMarks;
    public double merit;
    public List<DegreeProgram> preferences;
    public List<Subject> regSubject;
    public DegreeProgram regDegree;
}
```

```
class Subject
{
    public string code;
    public string type;
    public int creditHours;
    public int subjectFees;
}
```

```
class DegreeProgram
{
    public string degreeName;
    public float degreeDuration;
    public List<Subject> subjects;
    public int seats;
}
```

# Review

Student objects contain the list of preferences which are also objects.

```
class Student
{
    public string name;
    public int age;
    public double fscMarks;
    public double ecatMarks;
    public double merit;
    public List<DegreeProgram> preferences;
    public List<Subject> regSubject;
    public DegreeProgram regDegree;
}
```

```
class Subject
{
    public string code;
    public string type;
    public int creditHours;
    public int subjectFees;
}
```

```
class DegreeProgram
{
    public string degreeName;
    public float degreeDuration;
    public List<Subject> subjects;
    public int seats;
}
```

# Object-Relational Mapping

The technique for converting data of objects into permanent storage using object-oriented programming languages is Called Object-Relational Mapping (**ORM**)

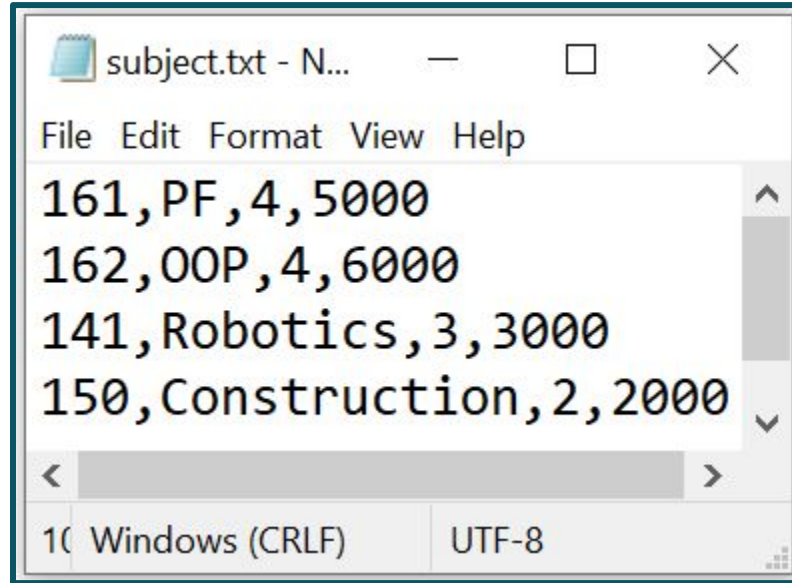
# | Data in Files

Lets see how we will store the data into files.



# Subject Data

Fields represent the following information.  
Subject Code, Subject Type, Credit Hours, Fees

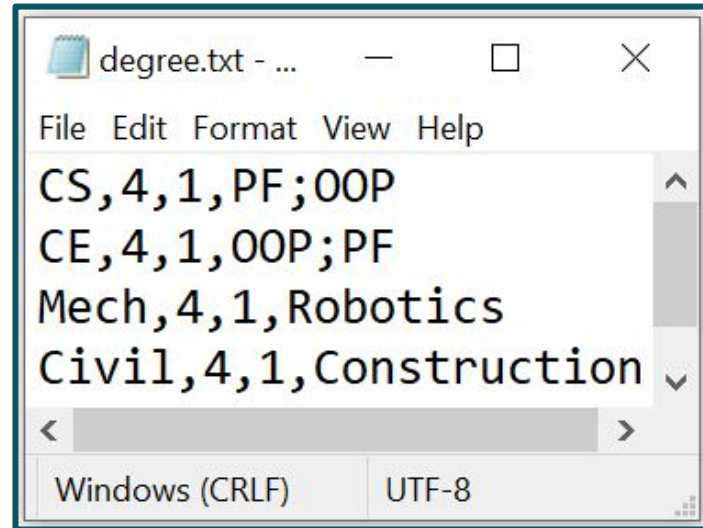


A screenshot of a text editor window titled "subject.txt - N...". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The text area contains four lines of data, each representing a subject entry: "161,PF,4,5000", "162,OOP,4,6000", "141,Robotics,3,3000", and "150,Construction,2,2000". The text is left-aligned and separated by commas. The window includes a vertical scrollbar on the right and a horizontal scrollbar at the bottom. The status bar at the bottom shows "10 Windows (CRLF)" and "UTF-8".

| Subject Code | Subject Type | Credit Hours | Fees |
|--------------|--------------|--------------|------|
| 161          | PF           | 4            | 5000 |
| 162          | OOP          | 4            | 6000 |
| 141          | Robotics     | 3            | 3000 |
| 150          | Construction | 2            | 2000 |

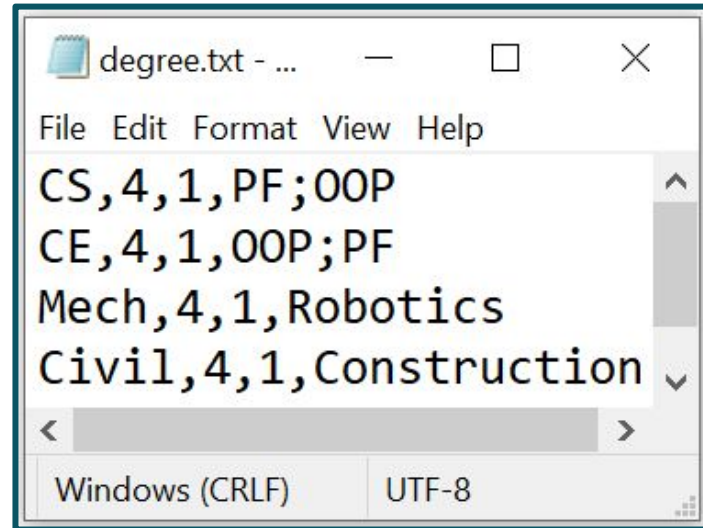
# DegreeProgram Data

Fields represent the following information.  
Degree Name, Duration, Seats, Subjects type



# DegreeProgram Data

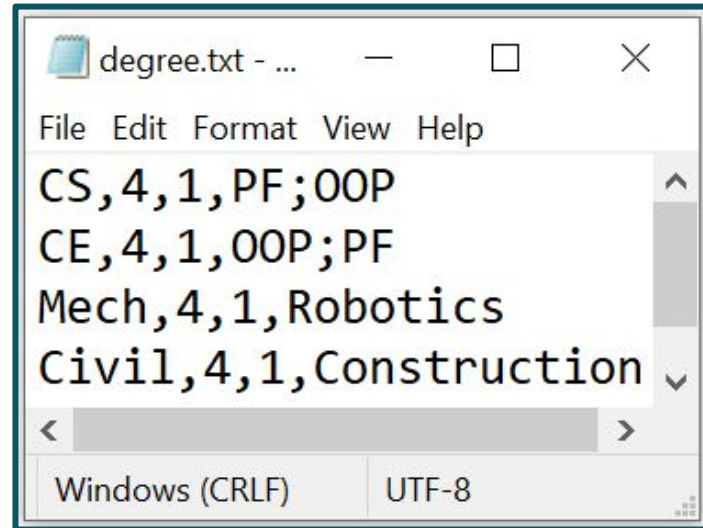
Important thing to note here is that we are just storing the **Subject type information** related to each degree.



```
degree.txt - ...
File Edit Format View Help
CS,4,1,PF;OOP
CE,4,1,OOP;PF
Mech,4,1,Robotics
Civil,4,1,Construction
Windows (CRLF) UTF-8
```

# DegreeProgram Data

These Subject types information are separated by semicolons as commas are separating the fields.

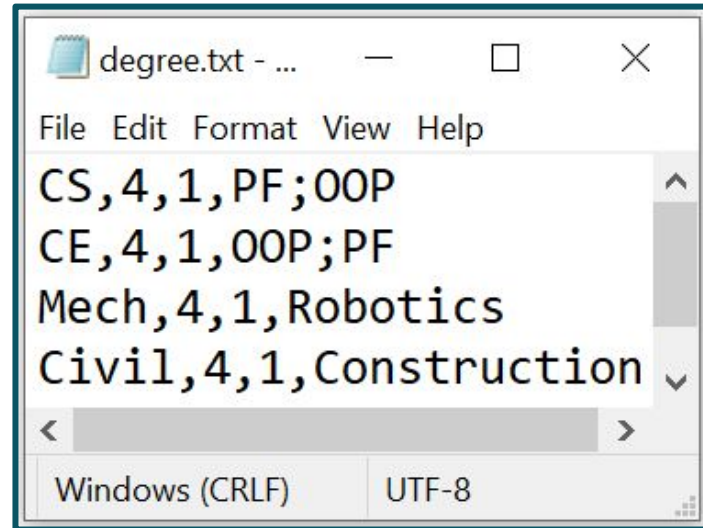


A screenshot of a text editor window titled "degree.txt - ...". The window contains a menu bar with "File", "Edit", "Format", "View", and "Help". The text area displays four lines of data, each representing a degree program: "CS,4,1,PF;OOP", "CE,4,1,OOP;PF", "Mech,4,1,Robotics", and "Civil,4,1,Construction". The data is formatted with commas separating the fields and semicolons separating the subject types. A horizontal scrollbar is visible below the text area. At the bottom of the window, there are two tabs: "Windows (CRLF)" and "UTF-8".

```
CS,4,1,PF;OOP
CE,4,1,OOP;PF
Mech,4,1,Robotics
Civil,4,1,Construction
```

# DegreeProgram Data

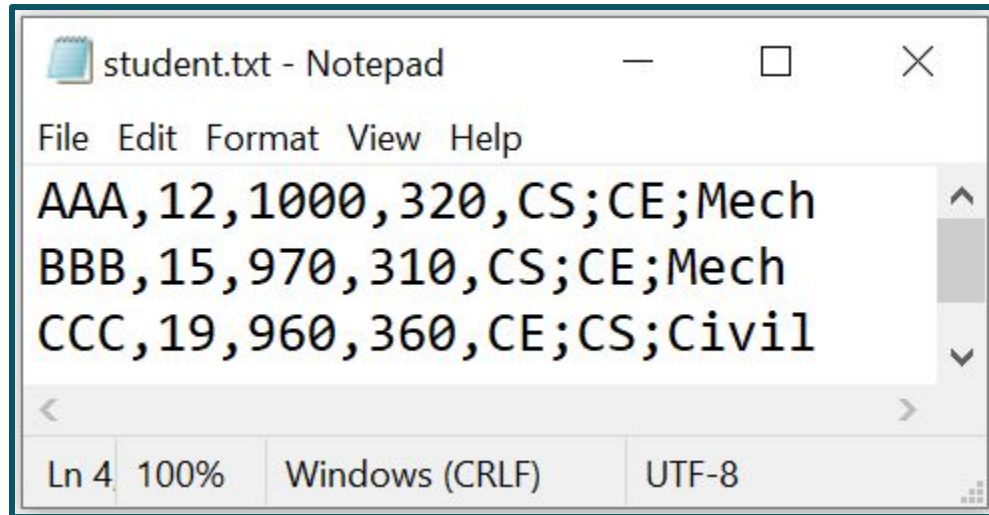
After reading the data into the objects in the code we will assign the related objects of subjects by comparing the Subject Type.



```
degree.txt - ...
File Edit Format View Help
CS,4,1,PF;OOP
CE,4,1,OOP;PF
Mech,4,1,Robotics
Civil,4,1,Construction
Windows (CRLF) UTF-8
```

# Student Data

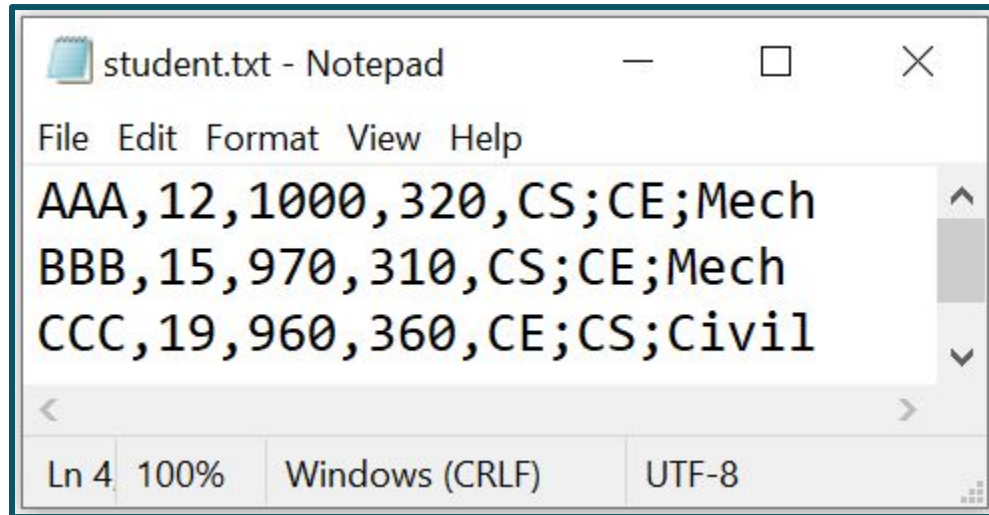
Fields represent the following information.  
Student Name, Age, Fsc Marks, Ecat Marks, Preferences



```
student.txt - Notepad
File Edit Format View Help
AAA,12,1000,320,CS;CE;Mech
BBB,15,970,310,CS;CE;Mech
CCC,19,960,360,CE;CS;Civil
Ln 4 100% Windows (CRLF) UTF-8
```

# Student Data

Preferences only contain the information of the **degree names**.



```
student.txt - Notepad
File Edit Format View Help
AAA,12,1000,320,CS;CE;Mech
BBB,15,970,310,CS;CE;Mech
CCC,19,960,360,CE;CS;Civil
Ln 4 100% Windows (CRLF) UTF-8
```

# | Load Data

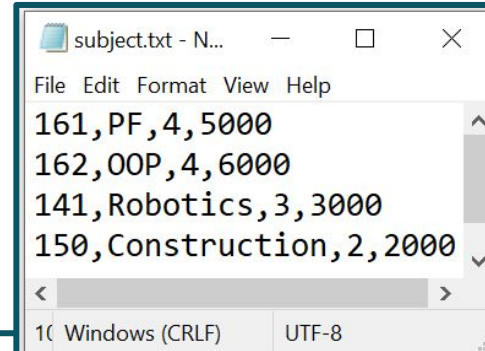
Lets see how to load this data and created related objects.



# LoadData for Subject

Previously, we had made a separate function for `parseData`. C# strings also provide a function `Split` that will split the records into string arrays after splitting on a specific character.

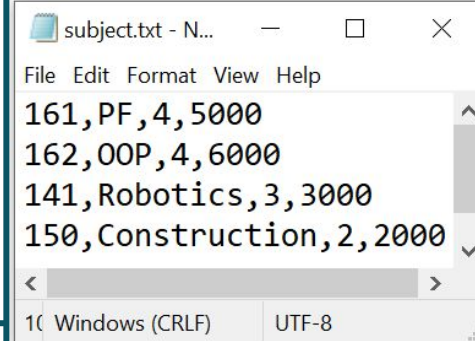
```
public static bool readFromFile(string path)
{
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path))
    {
        while ((record = f.ReadLine()) != null)
        {
            string[] splittedRecord = record.Split(',');
            string code = splittedRecord[0];
            string type = splittedRecord[1];
            int creditHours = int.Parse(splittedRecord[2]);
            int subjectFees = int.Parse(splittedRecord[3]);
            Subject s = new Subject(code, type, creditHours, subjectFees);
            addSubjectIntoList(s);
        }
        f.Close();
        return true;
    }
    else
    {
        return false;
    }
}
```



# LoadData for Subject

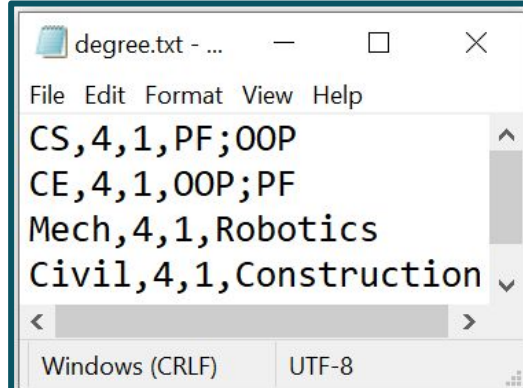
After successfully loading the data, we are creating the object and adding into the list

```
public static bool readFromFile(string path)
{
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path))
    {
        while ((record = f.ReadLine()) != null)
        {
            string[] splittedRecord = record.Split(',');
            string code = splittedRecord[0];
            string type = splittedRecord[1];
            int creditHours = int.Parse(splittedRecord[2]);
            int subjectFees = int.Parse(splittedRecord[3]);
            Subject s = new Subject(code, type, creditHours, subjectFees);
            addSubjectIntoList(s);
        }
        f.Close();
        return true;
    }
    else
    {
        return false;
    }
}
```



# LoadData for DegreeProgram

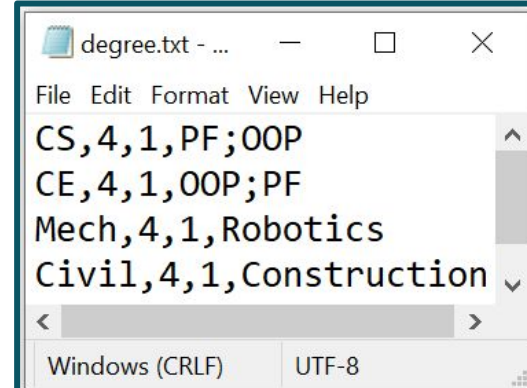
```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string degreeName = splittedRecord[0];
            float degreeDuration = float.Parse(splittedRecord[1]);
            int seats = int.Parse(splittedRecord[2]);
            string[] splittedRecordForSubject = splittedRecord[3].Split(';');
            DegreeProgram d = new DegreeProgram(degreeName, degreeDuration, seats);
            for (int x = 0; x < splittedRecordForSubject.Length; x++){
                Subject s = SubjectDL.isSubjectExists(splittedRecordForSubject[x]);
                if (s != null){
                    d.AddSubject(s);
                }
            }
            addIntoDegreeList(d);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



# LoadData for DegreeProgram

We  
splitted  
the  
records  
by  
comma.

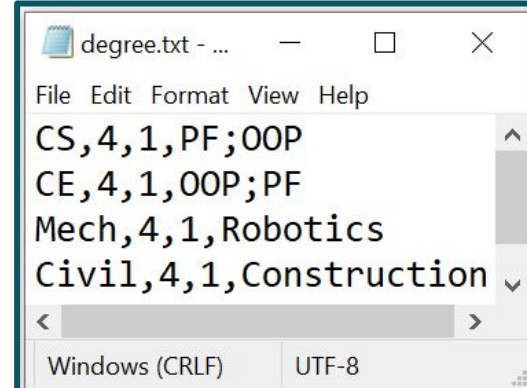
```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string degreeName = splittedRecord[0];
            float degreeDuration = float.Parse(splittedRecord[1]);
            int seats = int.Parse(splittedRecord[2]);
            string[] splittedRecordForSubject = splittedRecord[3].Split(';');
            DegreeProgram d = new DegreeProgram(degreeName, degreeDuration, seats);
            for (int x = 0; x < splittedRecordForSubject.Length; x++){
                Subject s = SubjectDL.isSubjectExists(splittedRecordForSubject[x]);
                if (s != null){
                    d.AddSubject(s);
                }
            }
            addIntoDegreeList(d);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



# LoadData for DegreeProgram

Last  
record is  
again  
splitted  
by  
semicolon.

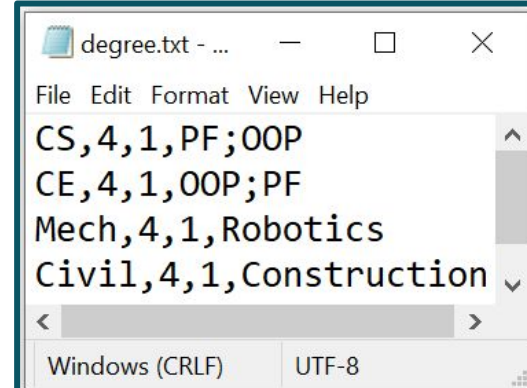
```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string degreeName = splittedRecord[0];
            float degreeDuration = float.Parse(splittedRecord[1]);
            int seats = int.Parse(splittedRecord[2]);
            string[] splittedRecordForSubject = splittedRecord[3].Split(';');
            DegreeProgram d = new DegreeProgram(degreeName, degreeDuration, seats);
            for (int x = 0; x < splittedRecordForSubject.Length; x++){
                Subject s = SubjectDL.isSubjectExists(splittedRecordForSubject[x]);
                if (s != null){
                    d.AddSubject(s);
                }
            }
            addIntoDegreeList(d);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



# LoadData for DegreeProgram

Then we create the DegreeProgram objects. But we are not adding subjects right now.

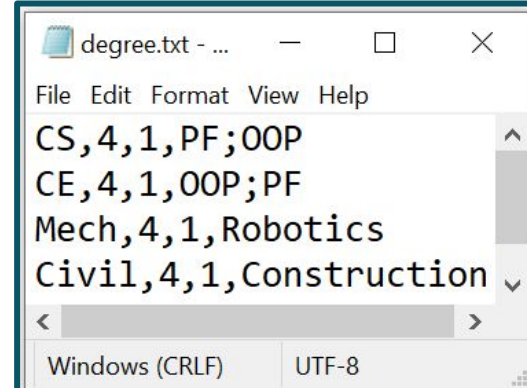
```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string degreeName = splittedRecord[0];
            float degreeDuration = float.Parse(splittedRecord[1]);
            int seats = int.Parse(splittedRecord[2]);
            string[] splittedRecordForSubject = splittedRecord[3].Split(';');
            DegreeProgram d = new DegreeProgram(degreeName, degreeDuration, seats);
            for (int x = 0; x < splittedRecordForSubject.Length; x++){
                Subject s = SubjectDL.isSubjectExists(splittedRecordForSubject[x]);
                if (s != null){
                    d.AddSubject(s);
                }
            }
            addIntoDegreeList(d);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



# LoadData for DegreeProgram

Here we are returning the reference of subject if it exists in the subjectList.

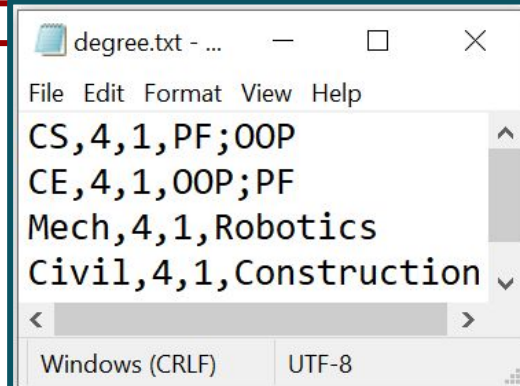
```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string degreeName = splittedRecord[0];
            float degreeDuration = float.Parse(splittedRecord[1]);
            int seats = int.Parse(splittedRecord[2]);
            string[] splittedRecordForSubject = splittedRecord[3].Split(';');
            DegreeProgram d = new DegreeProgram(degreeName, degreeDuration, seats);
            for (int x = 0; x < splittedRecordForSubject.Length; x++){
                Subject s = SubjectDL.isSubjectExists(splittedRecordForSubject[x]);
                if (s != null){
                    d.AddSubject(s);
                }
            }
            addIntoDegreeList(d);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



# LoadData for DegreeProgram

Then,  
adding  
the  
subject  
into the  
subject  
list of  
DegreeProgram.

```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string degreeName = splittedRecord[0];
            float degreeDuration = float.Parse(splittedRecord[1]);
            int seats = int.Parse(splittedRecord[2]);
            string[] splittedRecordForSubject = splittedRecord[3].Split(';');
            DegreeProgram d = new DegreeProgram(degreeName, degreeDuration, seats);
            for (int x = 0; x < splittedRecordForSubject.Length; x++){
                Subject s = SubjectDL.isSubjectExists(splittedRecordForSubject[x]);
                if (s != null){
                    d.AddSubject(s);
                }
            }
            addIntoDegreeList(d);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```

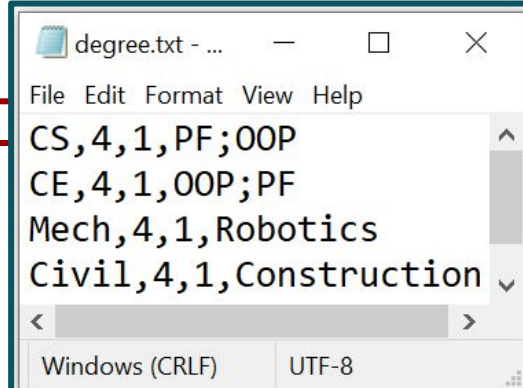




# LoadData for DegreeProgram

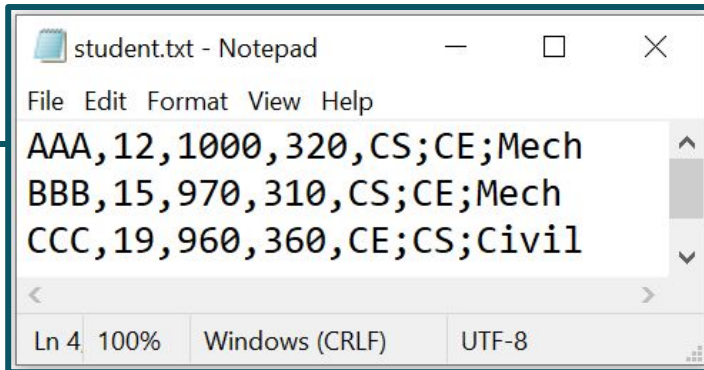
Then,  
adding  
the  
degree  
into the  
DegreePro  
gram list.

```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string degreeName = splittedRecord[0];
            float degreeDuration = float.Parse(splittedRecord[1]);
            int seats = int.Parse(splittedRecord[2]);
            string[] splittedRecordForSubject = splittedRecord[3].Split(';');
            DegreeProgram d = new DegreeProgram(degreeName, degreeDuration, seats);
            for (int x = 0; x < splittedRecordForSubject.Length; x++){
                Subject s = SubjectDL.isSubjectExists(splittedRecordForSubject[x]);
                if (s != null){
                    d.AddSubject(s);
                }
            }
            addIntoDegreeList(d);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



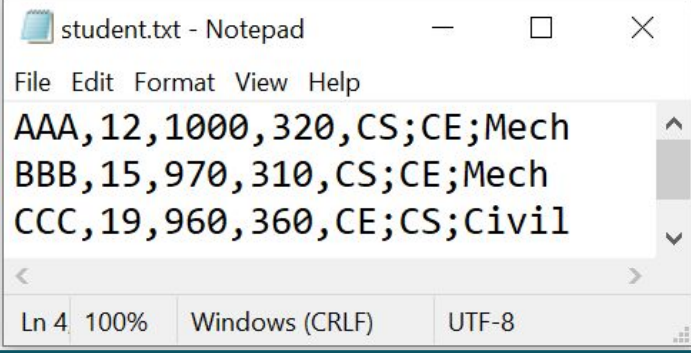
# LoadData for Student

```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string name = splittedRecord[0];
            int age = int.Parse(splittedRecord[1]);
            double fscMarks = double.Parse(splittedRecord[2]);
            double ecatMarks = double.Parse(splittedRecord[3]);
            string[] splittedRecordForPreference = splittedRecord[4].Split(';');
            List<DegreeProgram> preferences = new List<DegreeProgram>();
            for (int x = 0; x < splittedRecordForPreference.Length; x++){
                DegreeProgram d = DegreeProgramDL.isDegreeExists(splittedRecordForPreference[x]);
                if (d != null){
                    if (!preferences.Contains(d))
                        preferences.Add(d);
                }
            }
            Student s = new Student(name, age, fscMarks, ecatMarks, preferences);
            studentList.Add(s);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



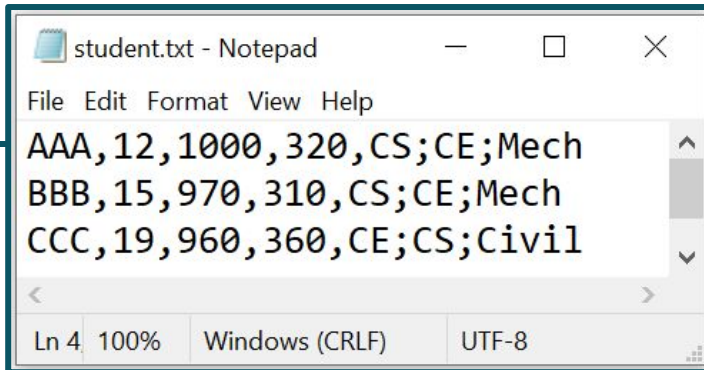
# LoadData for Student

```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string name = splittedRecord[0];
            int age = int.Parse(splittedRecord[1]);
            double fscMarks = double.Parse(splittedRecord[2]);
            double ecatMarks = double.Parse(splittedRecord[3]);
            string[] splittedRecordForPreference = splittedRecord[4].Split(';');
            List<DegreeProgram> preferences = new List<DegreeProgram>();
            for (int x = 0; x < splittedRecordForPreference.Length; x++){
                DegreeProgram d = DegreeProgramDL.isDegreeExists(splittedRecordForPreference[x]);
                if (d != null){
                    if (!preferences.Contains(d))
                        preferences.Add(d);
                }
            }
            Student s = new Student(name, age, fscMarks, ecatMarks, preferences);
            studentList.Add(s);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



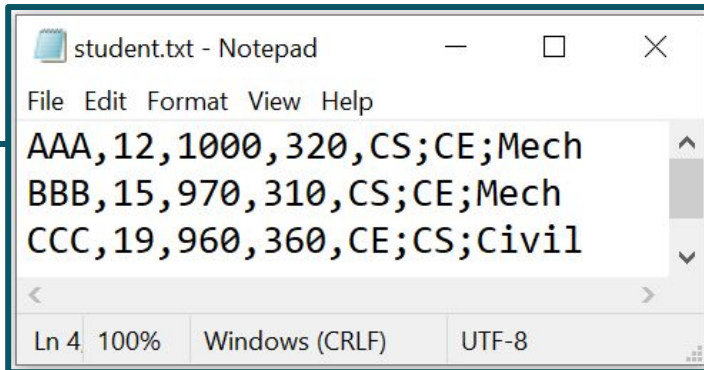
# LoadData for Student

```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string name = splittedRecord[0];
            int age = int.Parse(splittedRecord[1]);
            double fscMarks = double.Parse(splittedRecord[2]);
            double ecatMarks = double.Parse(splittedRecord[3]);
            string[] splittedRecordForPreference = splittedRecord[4].Split(';');
            List<DegreeProgram> preferences = new List<DegreeProgram>();
            for (int x = 0; x < splittedRecordForPreference.Length; x++){
                DegreeProgram d = DegreeProgramDL.isDegreeExists(splittedRecordForPreference[x]);
                if (d != null){
                    if (!preferences.Contains(d))
                        preferences.Add(d);
                }
            }
            Student s = new Student(name, age, fscMarks, ecatMarks, preferences);
            studentList.Add(s);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



# LoadData for Student

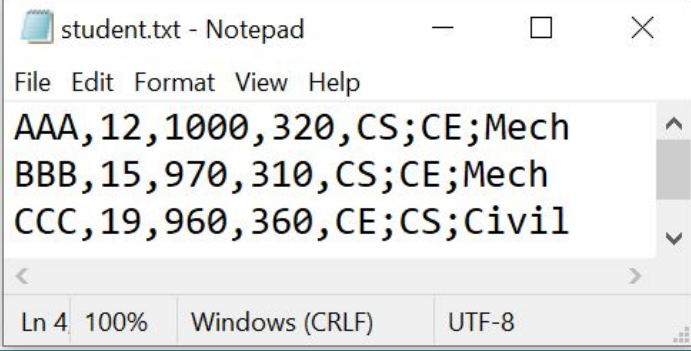
```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string name = splittedRecord[0];
            int age = int.Parse(splittedRecord[1]);
            double fscMarks = double.Parse(splittedRecord[2]);
            double ecatMarks = double.Parse(splittedRecord[3]);
            string[] splittedRecordForPreference = splittedRecord[4].Split(':');
            List<DegreeProgram> preferences = new List<DegreeProgram>();
            for (int x = 0; x < splittedRecordForPreference.Length; x++){
                DegreeProgram d = DegreeProgramDL.isDegreeExists(splittedRecordForPreference[x]);
                if (d != null){
                    if (!preferences.Contains(d))
                        preferences.Add(d);
                }
            }
            Student s = new Student(name, age, fscMarks, ecatMarks, preferences);
            studentList.Add(s);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



# LoadData for Student

```
public static bool readFromFile(string path){
    StreamReader f = new StreamReader(path);
    string record;
    if (File.Exists(path)){
        while ((record = f.ReadLine()) != null){
            string[] splittedRecord = record.Split(',');
            string name = splittedRecord[0];
            int age = int.Parse(splittedRecord[1]);
            double fscMarks = double.Parse(splittedRecord[2]);
            double ecatMarks = double.Parse(splittedRecord[3]);
            string[] splittedRecordForPreference = splittedRecord[4].Split(';');
            List<DegreeProgram> preferences = new List<DegreeProgram>();
            for (int x = 0; x < splittedRecordForPreference.Length; x++){
                DegreeProgram d = DegreeProgramDL.isDegreeExists(splittedRecordForPreference[x]);
                if (d != null){
                    if (!preferences.Contains(d))
                        preferences.Add(d);
                }
            }

            Student s = new Student(name, age, fscMarks, ecatMarks, preferences);
            studentList.Add(s);
        }
        f.Close();
        return true;
    }
    else
        return false;
}
```



# Driver Program: before displaying the menu

```
string subjectPath = "subject.txt";
string degreePath = "degree.txt";
string studentPath = "student.txt";
if (SubjectDL.readFromFile(subjectPath))
{
    Console.WriteLine("Subject Data Loaded Successfully");
}
if (DegreeProgramDL.readFromFile(degreePath))
{
    Console.WriteLine("DegreeProgram Data Loaded Successfully");
}
if (StudentDL.readFromFile(studentPath))
{
    Console.WriteLine("Student Data Loaded Successfully");
}
```

# | Store Data in Files

Lets see how we will store the data into files.



# StoreData for Subject

```
public static void storeintoFile(string path, Subject s)
{
    StreamWriter f = new StreamWriter(path, true);
    f.WriteLine(s.code + "," + s.type + "," + s.creditHours + "," + s.subjectFees);
    f.Flush();
    f.Close();
}
```

# StoreData for Student

```
public static void storeintoFile(string path, Student s)
{
    StreamWriter f = new StreamWriter(path, true);
    string degreeNames = "";
    for(int x = 0; x < s.preferences.Count - 1; x++)
    {
        degreeNames = degreeNames + s.preferences[x].degreeName + ",";
    }
    degreeNames = degreeNames + s.preferences[s.preferences.Count - 1].degreeName;
    f.WriteLine(s.name + "," + s.age + "," + s.fscMarks + "," + s.ecatMarks + "," + degreeNames);
    f.Flush();
    f.Close();
}
```

# StoreData for Student

First, we make a string by concatenation for degree names using semicolons.

```
public static void storeintoFile(string path, Student s)
{
    StreamWriter f = new StreamWriter(path, true);
    string degreeNames = "";
    for(int x = 0; x < s.preferences.Count - 1; x++)
    {
        degreeNames = degreeNames + s.preferences[x].degreeName + ";";
    }
    degreeNames = degreeNames + s.preferences[s.preferences.Count - 1].degreeName;
    f.WriteLine(s.name + "," + s.age + "," + s.fscMarks + "," + s.ecatMarks + "," + degreeNames);
    f.Flush();
    f.Close();
}
```

# StoreData for Student

Then write the data into the file.

```
public static void storeintoFile(string path, Student s)
{
    StreamWriter f = new StreamWriter(path, true);
    string degreeNames = "";
    for(int x = 0; x < s.preferences.Count - 1; x++)
    {
        degreeNames = degreeNames + s.preferences[x].degreeName + ",";
    }
    degreeNames = degreeNames + s.preferences[s.preferences.Count - 1].degreeName;
    f.WriteLine(s.name + "," + s.age + "," + s.fscMarks + "," + s.ecatMarks + "," + degreeNames);
    f.Flush();
    f.Close();
}
```

# StoreData for Student

We did not want to have **semicolon** after the last degree name therefore we concatenated it separately.

```
public static void storeintoFile(string path, Student s)
{
    StreamWriter f = new StreamWriter(path, true);
    string degreeNames = "";
    for(int x = 0; x < s.preferences.Count - 1; x++)
    {
        degreeNames = degreeNames + s.preferences[x].degreeName + ",";
    }
    degreeNames = degreeNames + s.preferences[s.preferences.Count - 1].degreeName;
    f.WriteLine(s.name + "," + s.age + "," + s.fscMarks + "," + s.ecatMarks + "," + degreeNames);
    f.Flush();
    f.Close();
}
```

# StoreData for DegreeProgram

Same goes for Degree Programs.

```
public static void storeintoFile(string path, DegreeProgram d)
{
    StreamWriter f = new StreamWriter(path, true);
    string SubjectNames = "";
    for(int x = 0; x < d.subjects.Count - 1; x++)
    {
        SubjectNames = SubjectNames + d.subjects[x].type + ",";
    }
    SubjectNames = SubjectNames + d.subjects[d.subjects.Count - 1].type;
    f.WriteLine(d.degreeName + "," + d.degreeDuration + "," + d.seats + "," + SubjectNames);
    f.Flush();
    f.Close();
}
```

# Driver Program

Just add one function call to store the data into the file as the data is added into the list.

```
if (option == 1)
{
    if (DegreeProgramDL.programList.Count > 0)
    {
        Student s = StudentUI.takeInputForStudent();
        StudentDL.addIntoStudentList(s);
        StudentDL.storeintoFile(studentPath, s);
    }
}
else if (option == 2)
{
    DegreeProgram d = DegreeProgramUI.takeInputForDegree();
    DegreeProgramDL.addIntoDegreeList(d);
    DegreeProgramDL.storeintoFile(degreePath, d);
}
```

# Driver Program

Just add one function call to store the data into the file as the data is added into the list.

```
if (option == 1)
{
    if (DegreeProgramDL.programList.Count > 0)
    {
        Student s = StudentUI.takeInputForStudent();
        StudentDL.addToStudentList(s);
        StudentDL.storeintoFile(studentPath, s);
    }
}
else if (option == 2)
{
    DegreeProgram d = DegreeProgramUI.takeInputForDegree();
    DegreeProgramDL.addToDegreeList(d);
    DegreeProgramDL.storeintoFile(degreePath, d);
}
```



# Conclusion

- The technique for converting data of objects into permanent storage using object-oriented programming languages is Called **Object-Relational Mapping (ORM)**



# Learning Objective

Write Code for **Storing and Loading** data of objects from files.

