Assignment 3 – Inheritance (Clark University Student Application)

This report documents a Windows Forms application that models four student categories (Part-Time, Undergraduate, Graduate, PhD) via inheritance. A shared **Student** base type centralizes identity/contact data and letter-grade mapping, while each subclass overrides **CalculateAverage()** to encode its weighting policy. A Template Method exposes both **ComputeAverage()** (numeric) and **ComputeGrade()** (letter) to the UI. Inclusive thresholds ensure boundary correctness (e.g., 90.0 -> A+).

Inheritance Design

- **Base Class – Student**
  - Common fields: firstName, lastName, test1, test2, street, city, telephone.
  - Grading pipeline:
    - CalculateAverage() – **abstract** in base, implemented by each subclass with its weights.
    - ComputeAverage() – public numeric result used by the UI.
    - ComputeGrade() – maps the numeric average to a letter using one centralized scale.
- **Derived Classes**
  - **PartTimeStudent**: adds ssNum.
  - **UnderGradStudent**: adds id.
  - **GradStudent**: adds id, thesis.
  - **PhDStudent**: adds id, phDAdvisor, dissertation.

Grade Computation Logic

Each subclass overrides CalculateAverage() with the assignment's weights:

Student Type	**Formula**
PartTimeStudent	0.4 * test1 + 0.6 * test2
UnderGradStudent	0.45 * test1 + 0.55 * test2
GradStudent	0.35 * test1 + 0.65 * test2
PhDStudent	0.3 * test1 + 0.7 * test2

Letter Grade Scale

After computing the numeric average, the app assigns a letter using **inclusive** boundaries:

- A+ for **avg ≥ 90**
- A for **85 ≤ avg < 90**
- B+ for **80 ≤ avg < 85**
- B for **75 ≤ avg < 80**
- C+ for **70 ≤ avg < 75**
- C for **65 ≤ avg < 70**
- D for **60 ≤ avg < 65**
- F for **avg < 60**

Note: Boundaries are **inclusive** (e.g., 90.0 is A+).

UI Workflow

1. **Type selection.** The form toggles visibility of type-specific inputs (SSN, ID, Thesis, Advisor, Dissertation) to minimize entry errors.
2. **Data entry.** Users provide name, two test scores (0–100), and address (street, city, telephone).
3. **Real-time feedback.** On score changes, the UI constructs a *temporary* instance of the selected type and calls ComputeAverage() and ComputeGrade(). The Grade field shows "<average> (<letter>)".
4. **Commit.** On **Add**, the form validates inputs, materializes the correct subclass, and appends it to the collection/view.

End-to-End Example

- Selection: **Part-Time**; Scores: **85** and **92**.
- Domain: PartTimeStudent.CalculateAverage() applies **0.4×85 + 0.6×92 = 89.2**.
- Mapping: Base class converts **89.2** → **A** (85 ≤ 89.2 < 90).
- UI displays **89.2 (A)**.

Validation Rules

- **Scores** must be numeric 0–100.
- **Identifiers**:
  - Part-Time -> **SSN** required.
  - Undergrad/Grad/PhD -> **ID** required.
- **Grad** -> **Thesis** required.
- **PhD** -> **PhD Advisor** and **Dissertation** required.
- **Basic contact** -> street, city, telephone required.

Sample Runs & Outputs

Sample 1 – Part-Time

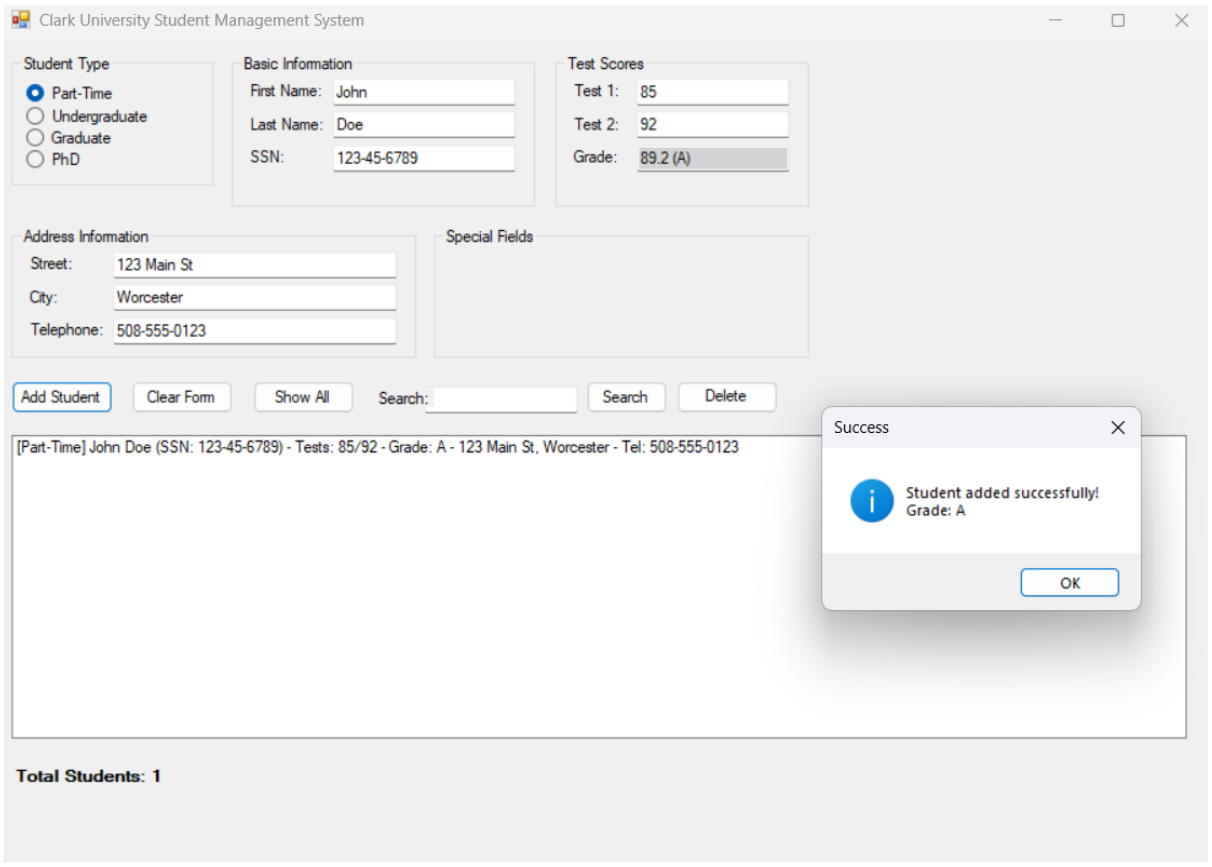
Input

- Name: John Doe
- Type: Part-Time
- SSN: 123-45-6789
- Test1: 85, Test2: 92
- Address: 123 Main St, Worcester, 508-555-0123

Computation

- Average = 0.4*85 + 0.6*92 = 34 + 55.2 = 89.2
- Letter = A (since 85 ≤ 89.2 < 90)

Displayed Grade: **89.2 (A)**



Student Type: [✓ Part-Time] Undergraduate Graduate PhD

First Name: John
Last Name: Doe
SSN: 123-45-6789

Test 1: 85
Test 2: 92
Grade: 89.2 (A)

Street: 123 Main St
City: Worcester
Telephone: 508-555-0123

Search:

```
[Part-Time] John Doe (SSN: 123-45-6789) - Tests: 85/92 - Grade: A - 123 Main St, Worcester - Tel: 508-555-0123
```

Student added successfully!
Grade: A

Total Students: 1

Sample 2 – PhD Student

Input

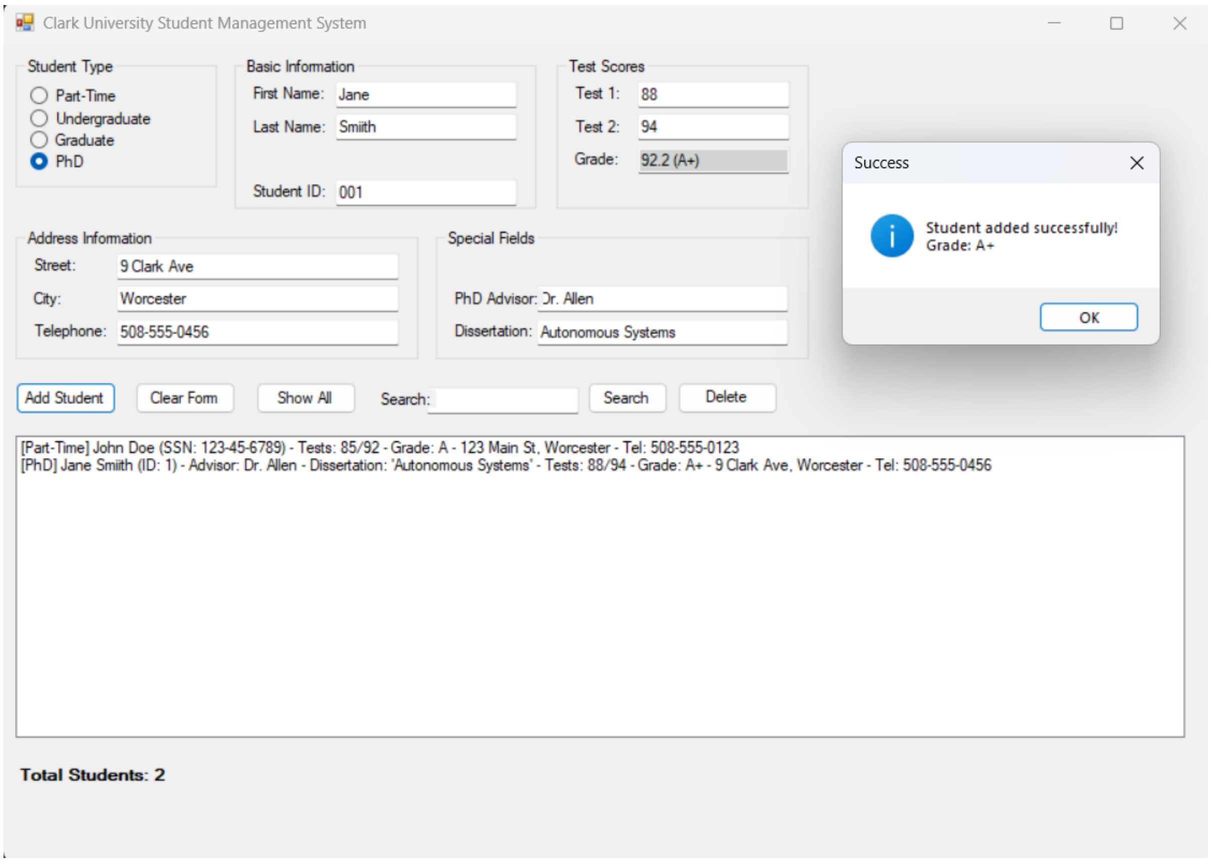
- Name: Jane Smith
- Type: PhD Student

- ID: 888-2025-001
- PhD Advisor: Dr. Allen
- Dissertation: Autonomous Systems
- Test1: 88, Test2: 94
- Address: 9 Clark Ave, Worcester, 508-555-0456

Computation

- Average = 0.3*88 + 0.7*94 = 26.4 + 65.8 = 92.2
- Letter = A+ (since 92.2 ≥ 90)

Displayed Grade: **92.2 (A+)**



Clark University Student Management System

Student Type	Part-Time [✓ PhD]
First Name:	Jane
Last Name:	Smith
Student ID:	001
Test 1:	88
Test 2:	94
Grade:	92.2 (A+)
Street:	9 Clark Ave
City:	Worcester
Telephone:	508-555-0456
PhD Advisor:	Dr. Allen
Dissertation:	Autonomous Systems

Success

Student added successfully!
Grade: A+

```
[Part-Time] John Doe (SSN: 123-45-6789) - Tests: 85/92 - Grade: A - 123 Main St, Worcester - Tel: 508-555-0123
[PhD] Jane Smith (ID: 1) - Advisor: Dr. Allen - Dissertation: 'Autonomous Systems' - Tests: 88/94 - Grade: A+ - 9 Clark Ave, Worcester - Tel: 508-555-0456
```

Total Students: 2

Boundary Case – Exactly 90.0

Example (Part-Time): test1 = 95, test2 = 87.5

- Average = 0.4*95 + 0.6*87.5 = 38 + 52.5 = 90.0
- With **inclusive** thresholds, **90.0** → **A+**.

If thresholds were exclusive (>), 90.0 would have produced A. The app uses **inclusive** rules to avoid ambiguity.

Requirements -> Implementation (Traceability)

Requirement	**Where implemented**
PartTimeStudent with **ssNum** and formula **0.4/0.6**	PartTimeStudent.CalculateAverage(); UI shows SSN field
UnderGradStudent with **id** and formula **0.45/0.55**	UnderGradStudent.CalculateAverage(); UI requires ID
GradStudent with **id, thesis** and formula **0.35/0.65**	GradStudent.CalculateAverage(); UI requires ID + Thesis
PhDStudent with **id, phDAdvisor, dissertation** and formula **0.3/0.7**	PhDStudent.CalculateAverage(); UI requires ID + Advisor + Dissertation
Shared scale for letter grades	Student.CalculateLetterGrade()
Real-time preview in UI	Score change handler → build temp object → ComputeAverage/ComputeGrade()
Validation of fields	UI validation before Add/Save

Key Methods Summary

Base class: Student

```
public abstract class Student
{
    // Common identity/contact + scores
    protected string firstName = string.Empty;
    protected string lastName = string.Empty;
    protected double test1;
    protected double test2;
    protected string street = string.Empty;
    protected string city = string.Empty;
    protected string telephone = string.Empty;

    // Each subclass supplies its own weighting
    protected abstract double CalculateAverage();
    // Numeric average used by the UI
    public double ComputeAverage()
    {
        return CalculateAverage();
    }

    // Maps numeric average to a letter once, in the base class
    public string ComputeGrade()
    {
        var average = CalculateAverage();
        return CalculateLetterGrade(average);
    }

    // Inclusive thresholds (e.g., 90.0 => A+)
    protected string CalculateLetterGrade(double average)
    {
        if (average >= 90) return "A+";
        else if (average >= 85) return "A";
        else if (average >= 80) return "B+";
        else if (average >= 75) return "B";
        else if (average >= 70) return "C+";
        else if (average >= 65) return "C";
        else if (average >= 60) return "D";
        else return "F";
    }
}
```

Example override: PartTimeStudent

```
public sealed class PartTimeStudent : Student
{
    public string ssNum { get; set; } = string.Empty;
    // 0.4 * test1 + 0.6 * test2
    protected override double CalculateAverage()
    {
        return 0.4 * test1 + 0.6 * test2;
    }

    public override string ToString()
    {
        // Show numeric + letter for clarity in lists
        return $"{firstName} {lastName} - Grade: {ComputeAverage():0.##}
({ComputeGrade()})";
    }
}
```

Additional Override Snippets

UnderGradStudent

```
public sealed class UnderGradStudent : Student
{
    public int Id { get; set; }

    // 0.45 * test1 + 0.55 * test2
    protected override double CalculateAverage()
    {
        return 0.45 * test1 + 0.55 * test2;
    }
}
```

GradStudent

```
public sealed class GradStudent : Student
{
    public int Id { get; set; }
    public string Thesis { get; set; } = string.Empty;
    // 0.35 * test1 + 0.65 * test2
    protected override double CalculateAverage()
    {
        return 0.35 * test1 + 0.65 * test2;
    }
}
```

PhDStudent

```
public sealed class PhDStudent : Student
{
    public int Id { get; set; }
    public string PhDAdvisor { get; set; } = string.Empty;
    public string Dissertation { get; set; } = string.Empty;

    // 0.3 * test1 + 0.7 * test2
    protected override double CalculateAverage()
    {
        return 0.3 * test1 + 0.7 * test2;
    }
}
```

Key UI Code Snippets

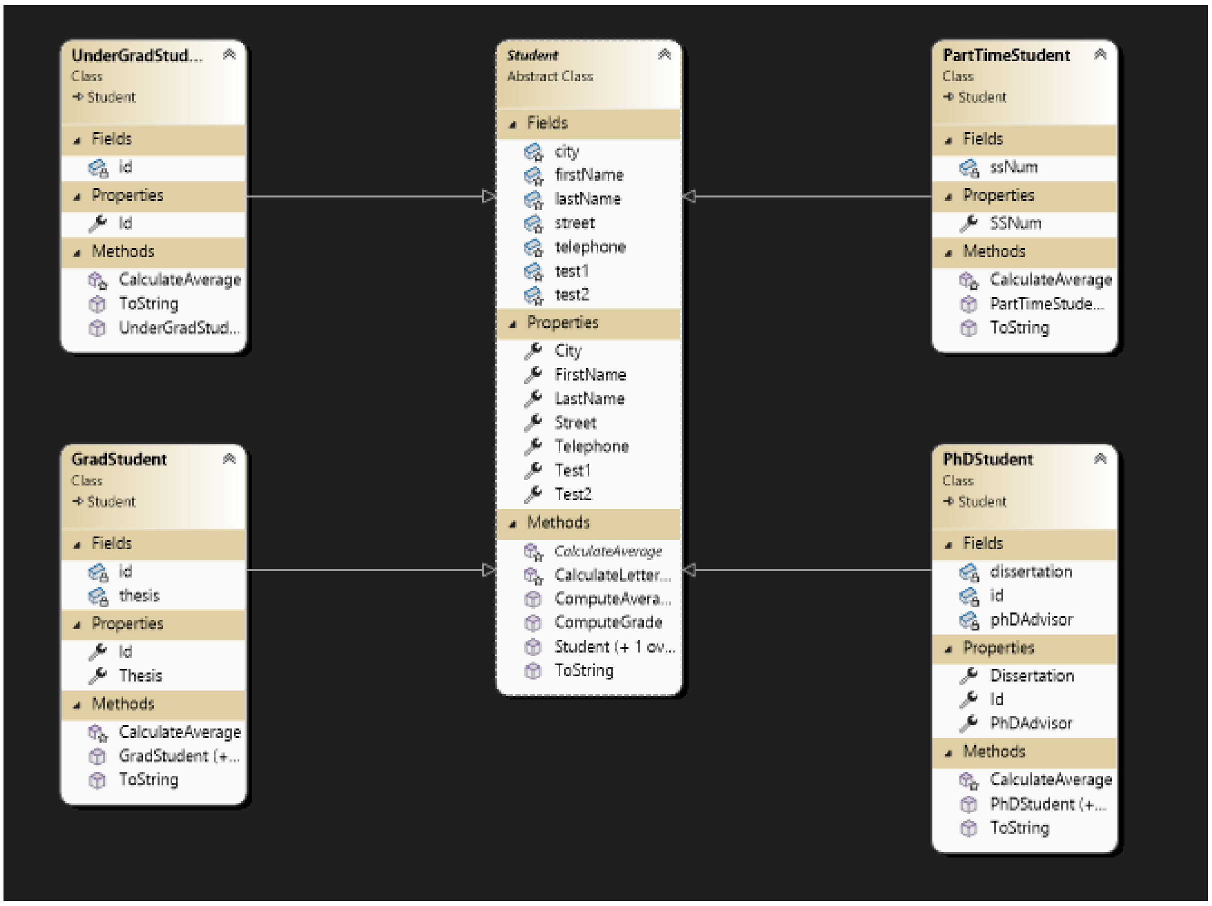
Real-time Grade Preview (as user types scores)

```
private void CalculateGradePolymorphically()
{
    if (!double.TryParse(txtTest1.Text, out var t1)) return;
    if (!double.TryParse(txtTest2.Text, out var t2)) return;
    Student temp = CreateStudentFromForm(t1, t2);
    if (temp == null) { txtGrade.Text = string.Empty; return; }

    double avg = temp.ComputeAverage();
    string letter = temp.ComputeGrade();
    txtGrade.Text = $"{avg:0.##} ({letter})";
}
```

Factory: Create the Correct Subclass Based on Selected Type

```
private Student? CreateStudentFromForm(double t1, double t2)
{
    string first = txtFirstName.Text.Trim();
    string last  = txtLastName.Text.Trim();
    string street = txtStreet.Text.Trim();
    string city = txtCity.Text.Trim();
    string tel = txtTelephone.Text.Trim();

    if (rbPartTime.Checked)
    {
        var ssn = txtSSN.Text.Trim();
        return new PartTimeStudent { firstName = first, lastName = last, street = street, city = city, telephone = tel, ssNum = ssn, test1 = t1, test2 = t2 };
    }
    if (rbUndergrad.Checked)
    {
        int.TryParse(txtID.Text, out int id);
        return new UnderGradStudent { firstName = first, lastName = last, street = street, city = city, telephone = tel, Id = id, test1 = t1, test2 = t2 };
    }
    if (rbGrad.Checked)
    {
        int.TryParse(txtID.Text, out int id);
        string thesis = txtThesis.Text.Trim();
        return new GradStudent { firstName = first, lastName = last, street = street, city = city, telephone = tel, Id = id, Thesis = thesis, test1 = t1, test2 = t2 };
    }
    if (rbPhD.Checked)
    {
        int.TryParse(txtID.Text, out int id);
        string advisor = txtAdvisor.Text.Trim();
        string diss = txtDissertation.Text.Trim();
        return new PhDStudent { firstName = first, lastName = last, street = street, city = city, telephone = tel, Id = id, PhDAdvisor = advisor, Dissertation = diss, test1 = t1, test2 = t2 };
    }
    return null;
}
```

Class Hierarchy



File Structure (source files)

- Student.cs – base class with template (CalculateAverage(), ComputeAverage(), ComputeGrade() + inclusive letter scale)
- PartTimeStudent.cs – 0.4/0.6 override
- UnderGradStudent.cs – 0.45/0.55 override
- GradStudent.cs – 0.35/0.65 override
- PhDStudent.cs – 0.3/0.7 override
- Form1.cs – WinForms logic (event handlers, validation, factory, live preview)
- Form1.Designer.cs – WinForms layout and control declarations