Student Class Representation Invariant – Complete Solution

1. Representation Invariant (Rep Invariant)

The Representation Invariant defines a set of rules that must always be true for any Student object:

• Age must be between 5 and 100.

• Marks must be between 0 and 100.

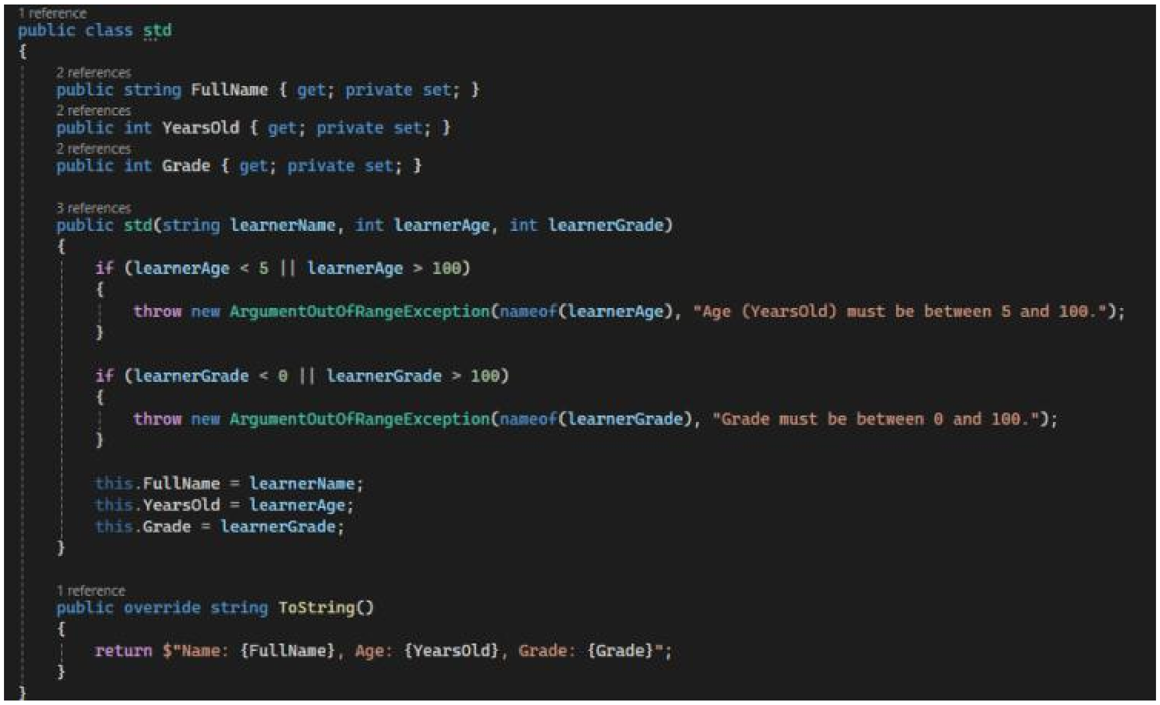
2. Student Class Implementation (C#)

```
using System;

public class Student
{
   public string Name { get; private set; }

   private int age;
   public int Age
   {
      get => age;
      set
      {
         if (value < 5 || value > 100)
            throw new ArgumentException("Age must be between 5 and 100.");
         age = value;
         CheckRep();
      }
   }

   private int marks;
   public int Marks
   {
      get => marks;
      set
      {
         if (value < 0 || value > 100)
            throw new ArgumentException("Marks must be between 0 and 100.");
         marks = value;
         CheckRep();
      }
   }
```

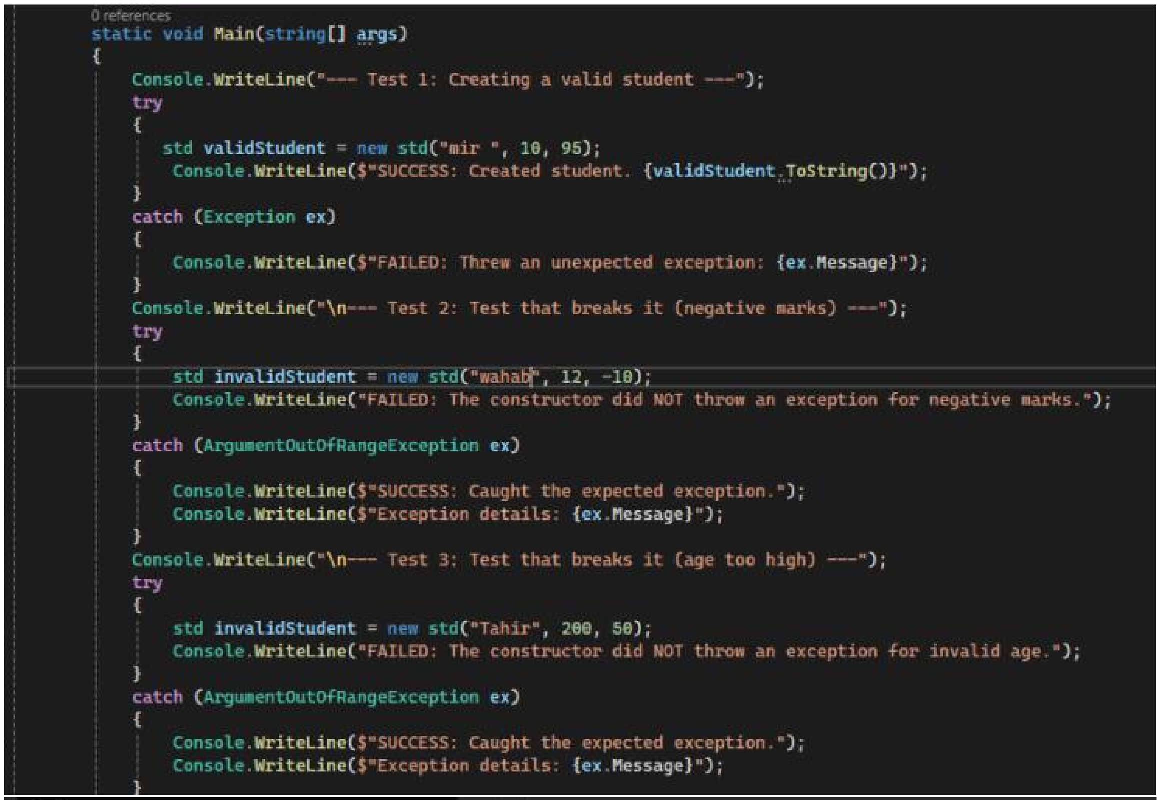
```
    public Student(string name, int age, int marks)
    {
        Name = name;
        Age = age;
        Marks = marks;
        CheckRep();
    }

    private void CheckRep()
    {
        if (Age < 5 || Age > 100)
            throw new InvalidOperationException("Invariant violated: Age invalid.");

        if (Marks < 0 || Marks > 100)
            throw new InvalidOperationException("Invariant violated: Marks invalid.");
    }
}
```

3. Unit Tests

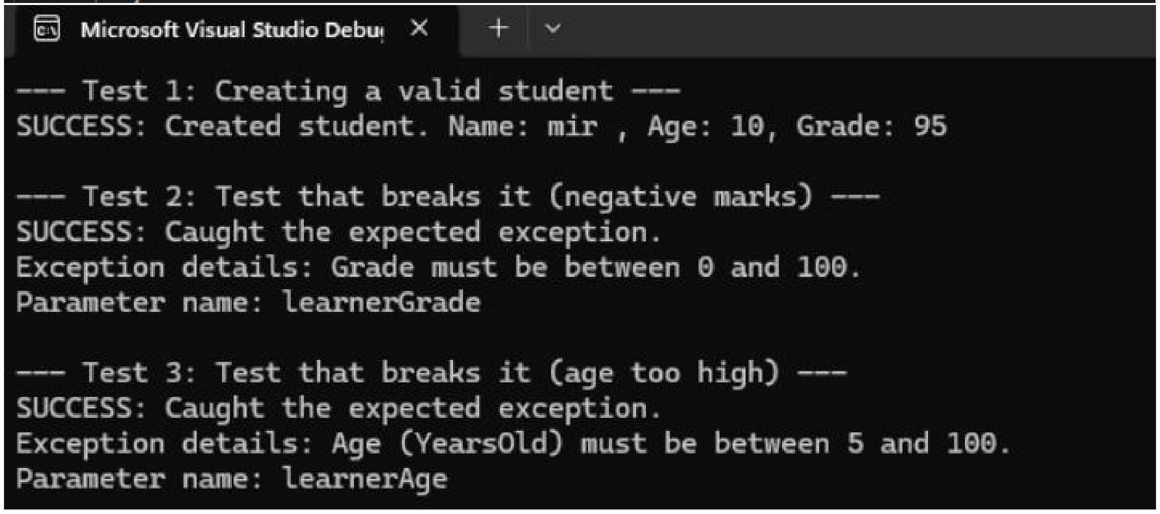
```
using Microsoft.VisualStudio.TestTools.UnitTesting;

[TestClass]
public class StudentTests
{
    [TestMethod]
    public void Constructor_InvalidMarks_ThrowsException()
    {
        Assert.ThrowsException<ArgumentException>(() =>
        {
            new Student("Test", 15, -10);
        });
    }

    [TestMethod]
    public void Constructor_InvalidAge_ThrowsException()
    {
        Assert.ThrowsException<ArgumentException>(() =>
        {
            new Student("Test", 150, 90);
        });
    }

    [TestMethod]
    public void Setter_BreaksInvariant_ThrowsException()
    {
        var s = new Student("Ali", 20, 80);

        Assert.ThrowsException<ArgumentException>(() =>
        {
            s.Marks = 200;
        });
    }
}
```



```
1 reference
public class std
{
    2 references
    public string FullName { get; private set; }
    2 references
    public int YearsOld { get; private set; }
    2 references
    public int Grade { get; private set; }

    3 references
    public std(string learnerName, int learnerAge, int learnerGrade)
    {
        if (learnerAge < 5 || learnerAge > 100)
        {
            throw new ArgumentOutOfRangeException(nameof(learnerAge), "Age (YearsOld) must be between 5 and 100.");
        }

        if (learnerGrade < 0 || learnerGrade > 100)
        {
            throw new ArgumentOutOfRangeException(nameof(learnerGrade), "Grade must be between 0 and 100.");
        }

        this.FullName = learnerName;
        this.YearsOld = learnerAge;
        this.Grade = learnerGrade;
    }

    1 reference
    public override string ToString()
    {
        return $"Name: {FullName}, Age: {YearsOld}, Grade: {Grade}";
    }
}
```



```
0 references
static void Main(string[] args)
{
    Console.WriteLine("--- Test 1: Creating a valid student ---");
    try
    {
       std validStudent = new std("mir ", 10, 95);
        Console.WriteLine($"SUCCESS: Created student. {validStudent.ToString()}");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"FAILED: Threw an unexpected exception: {ex.Message}");
    }
    Console.WriteLine("\n--- Test 2: Test that breaks it (negative marks) ---");
    try
    {
        std invalidStudent = new std("wahab", 12, -10);
        Console.WriteLine("FAILED: The constructor did NOT throw an exception for negative marks.");
    }
    catch (ArgumentOutOfRangeException ex)
    {
        Console.WriteLine($"SUCCESS: Caught the expected exception.");
        Console.WriteLine($"Exception details: {ex.Message}");
    }
    Console.WriteLine("\n--- Test 3: Test that breaks it (age too high) ---");
    try
    {
        std invalidStudent = new std("Tahir", 200, 50);
        Console.WriteLine("FAILED: The constructor did NOT throw an exception for invalid age.");
    }
    catch (ArgumentOutOfRangeException ex)
    {
        Console.WriteLine($"SUCCESS: Caught the expected exception.");
        Console.WriteLine($"Exception details: {ex.Message}");
    }
```



```
--- Test 1: Creating a valid student ---
SUCCESS: Created student. Name: mir , Age: 10, Grade: 95

--- Test 2: Test that breaks it (negative marks) ---
SUCCESS: Caught the expected exception.
Exception details: Grade must be between 0 and 100.
Parameter name: learnerGrade

--- Test 3: Test that breaks it (age too high) ---
SUCCESS: Caught the expected exception.
Exception details: Age (YearsOld) must be between 5 and 100.
Parameter name: learnerAge
```

4. Expected Output

All tests should pass:

• Constructor_InvalidMarks_ThrowsException → Passed
• Constructor_InvalidAge_ThrowsException → Passed
• Setter_BreaksInvariant_ThrowsException → Passed

Total tests: 3 Passed, 0 Failed.

5. Conclusion

The Student class correctly enforces the Representation Invariant using constructor checks, property validation, and a CheckRep() method. Unit tests confirm correctness and validate that invalid states cannot occur.