Technical Project Report: HealthSync Advanced Billing

1. Executive Summary

The **HealthSync Advanced Billing System** is a C# application designed for Machine Masters to automate medical consultant payouts. The system utilizes **Object-Oriented Programming (OOP)** to handle complex payroll logic, multi-tier taxation (TDS), and strict data validation for two distinct employee categories: In-House and Visiting Consultants.

---

2. System Architecture

The application is built on three core OOP pillars:

A. Abstraction (The `Consultant` Class)

We implemented an `abstract` base class to act as a blueprint.

- **Why?** It prevents the system from creating a "generic" consultant. Every consultant must be either In-House or Visiting.
- **Key Feature:** The `CalculateGrossPayout()` method is abstract, forcing subclasses to define their own financial math.

B. Polymorphism (Method Overriding)

The system uses polymorphism to handle the specialized payout formulas:

- **In-House:** Uses a formula involving a `MonthlyStipend` plus calculated allowances and bonuses.
- **Visiting:** Uses a formula based on `ConsultationsCount` and `RatePerVisit`.

C. Virtual Logic (Dynamic Taxation)

We used a `virtual` method for the **TDS (Tax Deducted at Source)** calculation.

- **Default Logic:** A sliding scale (5% or 15%) based on earnings.
- **Overridden Logic:** For `Visiting` consultants, the method is overridden to return a flat 10% rate, demonstrating how subclasses can opt out of default behavior.

---

3. Functional Specifications

Validation Rules

All IDs must pass the `ValidateConsultantId()` check:

1. **Length:** Exactly 6 characters.
2. **Prefix:** Must start with "DR".
3. **Format:** The last 4 characters must be numeric.

Taxation Table

Consultant Type	Condition	Applied Rate
In-House	Payout $\le$ 5000	5%
In-House	Payout $>$ 5000	15%
Visiting	Any Amount	10% (Flat)

---

4. Sample Output Scenarios

Scenario 1: In-House Consultant (High Earner)

- **Input:** ID: `DR2001`, Stipend: `10000`
- **Process:** Gross = $10000 + 2000 + 1000 = 13000$. Tax = 15%.
- **Output:** `Gross: 13000.00 | TDS Applied: 15% | Net Payout: 11050.00`

Scenario 2: Visiting Consultant

- **Input:** ID: `DR8005`, 10 Visits @ `600`
- **Process:** Gross = 6000. Tax = 10% (Flat).
- **Output:** `Gross: 6000.00 | TDS Applied: 10% | Net Payout: 5400.00`

Scenario 3: Validation Failure

- **Input:** ID: `MD1001`
- **Output:** `Invalid doctor id` (Process terminates).

Expected Code Solution

C#

```
using System;

namespace HealthSync
{
   public abstract class Consultant
   {
      public string Id { get; set; }
      public string Name { get; set; }
      public double PayoutAmount { get; set; }

      public bool ValidateConsultantId()
      {
         if (Id.Length == 6 && Id.StartsWith("DR"))
         {
            return int.TryParse(Id.Substring(2), out _);
         }
         return false;
      }

      public abstract Consultant CalculatePayout();
   }

   public class InHouse : Consultant
   {
      public double MonthlyStipend { get; set; }

      public override Consultant CalculatePayout()
      {
         double allowance = 0.20 * MonthlyStipend;
         double bonus = 0.10 * MonthlyStipend;
         this.PayoutAmount = MonthlyStipend + allowance + bonus;
         return this;
      }
   }

   public class Visiting : Consultant
   {
      public int ConsultationsCount { get; set; }
      public int RatePerVisit { get; set; }

      public override Consultant CalculatePayout()
      {
         this.PayoutAmount = ConsultationsCount * RatePerVisit;
         return this;
      }
   }

   class Program
   {
```

```
    static void Main()
    {
        Console.WriteLine("1.In-House\n2.Visiting\nChoose consultant type");
        int choice = int.Parse(Console.ReadLine());

        Consultant doc = (choice == 1) ? new InHouse() : new Visiting();

        Console.WriteLine("Enter the doctor id");
        doc.Id = Console.ReadLine();

        if (doc.ValidateConsultantId())
        {
            Console.WriteLine("Enter name");
            doc.Name = Console.ReadLine();

            if (doc is InHouse ih)
            {
                Console.WriteLine("Enter monthly stipend");
                ih.MonthlyStipend = double.Parse(Console.ReadLine());
            }
            else if (doc is Visiting v)
            {
                Console.WriteLine("Enter consultations count");
                v.ConsultationsCount = int.Parse(Console.ReadLine());
                Console.WriteLine("Enter rate per visit");
                v.RatePerVisit = int.Parse(Console.ReadLine());
            }

            doc.CalculatePayout();
            Console.WriteLine($"Doctor ID : {doc.Id}, Name : {doc.Name}, Payout : {doc.PayoutAmount}");
        }
        else
        {
            Console.WriteLine("Invalid doctor id");
        }
    }
  }
}
```

```
using System;


namespace HealthSyncApp

{

  // ========================================

  // 1. ABSTRACT BASE CLASS: Consultant

  // ========================================
```

```
public abstract class Consultant
{
    // Public properties
    public string Id { get; set; }
    public string Name { get; set; }
    public double PayoutAmount { get; set; }

    /// <summary>
    /// Validates ID: Length 6, starts with "DR", ends with 4 digits.
    /// </summary>
    public bool ValidateConsultantId()
    {
        if (string.IsNullOrEmpty(Id) || Id.Length != 6)
            return false;

        // Check prefix and ensure the remaining 4 characters are numeric
        return Id.StartsWith("DR") && int.TryParse(Id.Substring(2), out _);
    }

    /// <summary>
    /// Abstract method to be implemented by derived classes for unique pay logic.
    /// </summary>
    public abstract void CalculateGrossPayout();

    /// <summary>
    /// Virtual method: Default tax logic (Dynamic Brackets for In-House).
    /// Can be overridden by specific consultant types.
    /// </summary>
    public virtual double GetTaxRate()
```

```
    {
        // Logic: 15% for high earners (>5000), 5% otherwise
        return (this.PayoutAmount > 5000) ? 0.15 : 0.05;
    }

    /// <summary>
    /// Concrete method: Applies the calculated/overridden tax rate.
    /// </summary>
    public void ApplyTax()
    {
        double rate = GetTaxRate();
        double taxAmount = this.PayoutAmount * rate;
        this.PayoutAmount -= taxAmount;

        Console.WriteLine($"Tax Computation: Rate {rate * 100}% | Amount Deducted: {taxAmount:F2}");
    }
  }

  // ========================================
  // 2. DERIVED CLASS: InHouse
  // ========================================
  public class InHouse : Consultant
  {
    public double MonthlyStipend { get; set; }

    public override void CalculateGrossPayout()
    {
        // Formula: Stipend + 20% Allowance + 10% Bonus
```

```
        double allowance = 0.20 * MonthlyStipend;
        double bonus = 0.10 * MonthlyStipend;
        this.PayoutAmount = MonthlyStipend + allowance + bonus;
    }
}

// ========================================
// 3. DERIVED CLASS: Visiting
// ========================================
public class Visiting : Consultant
{
    public int ConsultationsCount { get; set; }
    public int RatePerVisit { get; set; }

    public override void CalculateGrossPayout()
    {
        // Formula: Count * Rate
        this.PayoutAmount = ConsultationsCount * RatePerVisit;
    }

    /// <summary>
    /// Overriding the default dynamic tax with a Flat 10% for Visiting Consultants.
    /// </summary>
    public override double GetTaxRate()
    {
        return 0.10;
    }
}
```

```
// ========================================
// 4. MAIN EXECUTION CLASS
// ========================================
public class Program
{
   static void Main(string[] args)
   {
      try
      {
         Console.WriteLine("=== HealthSync Consultant Billing System ===");
         Console.WriteLine("1. In-House Consultant");
         Console.WriteLine("2. Visiting Consultant");
         Console.Write("Select Employment Type: ");
         int choice = int.Parse(Console.ReadLine());

         // Polymorphism: Instantiate based on user choice
         Consultant consultant = (choice == 1) ? new InHouse() : new Visiting();

         Console.Write("Enter Consultant ID (e.g., DR1001): ");
         consultant.Id = Console.ReadLine();

         // Step 1: Validation
         if (consultant.ValidateConsultantId())
         {
            Console.Write("Enter Consultant Name: ");
            consultant.Name = Console.ReadLine();

            // Step 2: Specific Data Entry
            if (consultant is InHouse ih)
```

```
    {
        Console.Write("Enter Monthly Stipend: ");
        ih.MonthlyStipend = double.Parse(Console.ReadLine());
    }
    else if (consultant is Visiting v)
    {
        Console.Write("Enter Total Consultations: ");
        v.ConsultationsCount = int.Parse(Console.ReadLine());
        Console.Write("Enter Rate per Visit: ");
        v.RatePerVisit = int.Parse(Console.ReadLine());
    }

    // Step 3: Payout Sequence
    consultant.CalculateGrossPayout();
    double gross = consultant.PayoutAmount; // Store gross for summary

    Console.WriteLine("\n--- Processing Financials ---");
    consultant.ApplyTax(); // Applies tax based on virtual/override logic

    // Step 4: Final Output
    Console.WriteLine("------------------------------");
    Console.WriteLine($"ID: {consultant.Id}");
    Console.WriteLine($"Consultant Name: {consultant.Name}");
    Console.WriteLine($"Gross Payout: {gross:F2}");
    Console.WriteLine($"Net Payout (After TDS): {consultant.PayoutAmount:F2}");
    Console.WriteLine("------------------------------");
}
else
{
```

```
                Console.WriteLine("\nError: Invalid doctor id. Access Denied.");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: Invalid input format. {ex.Message}");
        }

        Console.WriteLine("\nPress any key to exit...");
        Console.ReadKey();
    }
  }
}
```

==========Unit Test==========

```
using NUnit.Framework;
using HealthSyncApp;

namespace HealthSyncTests
{
  [TestFixture]
  public class BillingTests
  {
    // 1. TEST ID VALIDATION
    [Test]
    [TestCase("DR1234", ExpectedResult = true)]  // Valid
    [TestCase("DR999", ExpectedResult = false)]  // Too short
    [TestCase("MD1234", ExpectedResult = false)]  // Wrong prefix
    [TestCase("DRABCD", ExpectedResult = false)] // Not numeric
    public bool Test_ConsultantId_Validation(string id)
```

```
{
    var doc = new InHouse { Id = id };
    return doc.ValidateConsultantId();
}

// 2. TEST IN-HOUSE GROSS PAY CALCULATION
[Test]
public void Test_InHouse_GrossCalculation()
{
    // Arrange
    var ih = new InHouse { MonthlyStipend = 10000 };

    // Act
    ih.CalculateGrossPayout();

    // Assert: 10000 + 2000 (20%) + 1000 (10%) = 13000
    Assert.AreEqual(13000, ih.PayoutAmount);
}

// 3. TEST IN-HOUSE DYNAMIC TAX (High Bracket)
[Test]
public void Test_InHouse_Tax_HighBracket()
{
    var ih = new InHouse { MonthlyStipend = 10000 };
    ih.CalculateGrossPayout(); // 13000

    // Act
    ih.ApplyTax(); // Should apply 15% because > 5000
```

```
    // Assert: 13000 - 15% (1950) = 11050
    Assert.AreEqual(11050, ih.PayoutAmount);
}

// 4. TEST IN-HOUSE DYNAMIC TAX (Low Bracket)
[Test]
public void Test_InHouse_Tax_LowBracket()
{
    var ih = new InHouse { MonthlyStipend = 2000 };
    ih.CalculateGrossPayout(); // 2600

    // Act
    ih.ApplyTax(); // Should apply 5% because <= 5000

    // Assert: 2600 - 5% (130) = 2470
    Assert.AreEqual(2470, ih.PayoutAmount);
}

// 5. TEST VISITING FLAT TAX RATE
[Test]
public void Test_Visiting_FlatTax()
{
    // Arrange
    var v = new Visiting { ConsultationsCount = 20, RatePerVisit = 500 };
    v.CalculateGrossPayout(); // 10000

    // Act
    v.ApplyTax(); // Should apply 10% (Overridden) regardless of amount
```

```
            // Assert: 10000 - 10% (1000) = 9000
            Assert.AreEqual(9000, v.PayoutAmount);
        }
    }
}
```

================================ Boilerplate Code==========================

```
using System;

namespace HealthSyncApp
{
    // --- ABSTRACT LAYER ---
    public abstract class Consultant
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public double PayoutAmount { get; set; }

        // Method to be implemented for ID validation
        public bool ValidateConsultantId()
        {
            // TODO: Implement logic (Length 6, Starts with "DR", 4 digits)
            return false;
        }

        // Abstract method for payout logic
        public abstract void CalculateGrossPayout();

        // Virtual method for dynamic tax rates
        public virtual double GetTaxRate()
```

```
    {
        // TODO: Implement sliding scale (5% or 15%)
        return 0.0;
    }

    // Method to apply tax
    public void ApplyTax()
    {
        // TODO: Call GetTaxRate and deduct from PayoutAmount
    }
}

// --- IMPLEMENTATION LAYER ---
public class InHouse : Consultant
{
    public double MonthlyStipend { get; set; }

    public override void CalculateGrossPayout()
    {
        // TODO: Implement Stipend + Allowance + Bonus
    }
}

public class Visiting : Consultant
{
    public int ConsultationsCount { get; set; }
    public int RatePerVisit { get; set; }

    public override void CalculateGrossPayout()
```

```
        {
            // TODO: Implement Count * Rate
        }

        public override double GetTaxRate()
        {
            // TODO: Override with flat 10%
            return 0.10;
        }
    }

    // --- ENTRY POINT ---
    class Program
    {
        static void Main(string[] args)
        {
            // TODO: Implement menu selection and user input flow
            Console.WriteLine("System Initialized...");
        }
    }
}
```