**Name Haseeb Ullah**

**ID: F20232661009**

**Problem 1: Inheritance and Polymorphism (8 marks)**

**a)**

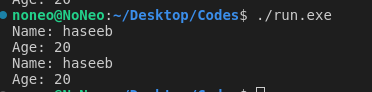
Create a base class called Person with protected member variables name and age. Provide a constructor to initialize

these variables and a virtual function displayDetails() to display the name and age.

**Source code:**

|  |
| --- |
| #include <iostream>  using namespace std;  class Person  {  protected:  string name;  int age;  public:  Person(string n = "", int a = 0) : name(n), age(a) {}  virtual void displayDetails()  {  cout << "Name: " << name << endl;  cout << "Age: " << age << endl;  }  };  int main()  {  Person\* p;  Person p1("haseeb", 20);  p1.displayDetails();  p = &p1;  p->displayDetails();  return 0;  } |

**Output:**



**b)**

Derive two classes, Student and Teacher, from Person. Add a private member variable studentId in the Student class

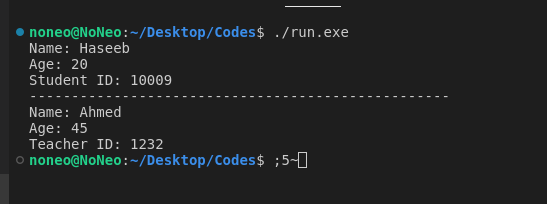
and teacherId in the Teacher class. Implement the displayDetails() function in each derived class to display the

additional information along with the name and age.

**Source code:**

|  |
| --- |
| // Derive two classes, Student and Teacher, from Person. Add a private member variable studentId in the Student class  // and teacherId in the Teacher class. Implement the displayDetails() function in each derived class to display the  // additional information along with the name and age.  #include <iostream>  using namespace std;  class Person  {  protected:  string name;  int age;  public:  Person(string n = "", int a = 0) : name(n), age(a) {}  virtual void displayDetails() {  cout << "Name: " << name << endl;  cout << "Age: " << age << endl;  }  };  class Student : public Person  {  private:  int studentId;  public:  Student(string n = "", int a = 0, int id = 0) : Person(n, a), studentId(id) {}  void displayDetails()  {  Person::displayDetails();  cout << "Student ID: " << studentId << endl;  }  };  class Teacher : public Person  {  private:  int teacherId;  public:  Teacher(string n = "", int a = 0, int id = 0) : Person(n, a), teacherId(id) {}  void displayDetails()  {  Person::displayDetails();  cout << "Teacher ID: " << teacherId << endl;  }  };  int main()  {  Student s("Haseeb", 20, 10009);  Teacher t("Ahmed", 45, 1232);  s.displayDetails();  cout << "--------------------------------------------------\n";  t.displayDetails();  return 0;  } |

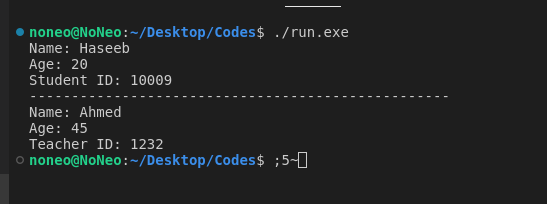
**Output:**

**C)**

**Source code:**

|  |
| --- |
| // Create objects of Student and Teacher and demonstrate polymorphism  // by calling the displayDetails() function on  // each object.  #include <iostream>  using namespace std;  class Person  {  protected:  string name;  int age;  public:  Person(string n = "", int a = 0) : name(n), age(a) {}  virtual void displayDetails() {  cout << "Name: " << name << endl;  cout << "Age: " << age << endl;  }  };  class Student : public Person  {  private:  int studentId;  public:  Student(string n = "", int a = 0, int id = 0) : Person(n, a), studentId(id) {}  void displayDetails()  {  Person::displayDetails();  cout << "Student ID: " << studentId << endl;  }  };  class Teacher : public Person  {  private:  int teacherId;  public:  Teacher(string n = "", int a = 0, int id = 0) : Person(n, a), teacherId(id) {}  void displayDetails()  {  Person::displayDetails();  cout << "Teacher ID: " << teacherId << endl;  }  };  int main()  {  Person\* p;  Student s("Haseeb", 20, 10009);  Teacher t("Ahmed", 45, 1232);  p = &s;  p->displayDetails();  cout << "--------------------------------------------------\n";  p = &t;  p->displayDetails();  return 0;  } |

**Output:**



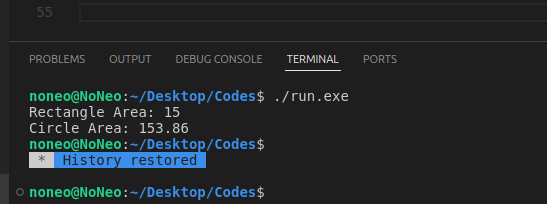
**Problem 2: Abstraction (7 marks)**

**a)**

**Source code:**

|  |
| --- |
| **#include <iostream>**  **using namespace std;**  **class Shape**  **{**  **public:**  **virtual double calculateArea() const = 0;**  **};**  **class Rectangle : public Shape**  **{**  **private:**  **double length;**  **double width;**  **public:**  **Rectangle(double l = 0, double w = 0) : length(l), width(w) {}**  **double calculateArea() const**  **{**  **return length \* width;**  **}**  **};**  **class Circle : public Shape**  **{**  **private:**  **double radius;**  **public:**  **Circle(double r = 0) : radius(r) {}**  **double calculateArea() const**  **{**  **return 3.14 \* radius \* radius;**  **}**  **};**  **int main()**  **{**  **Rectangle rect(5, 3);**  **Circle circ(7);**  **Shape\* shape;**  **shape = &rect;**  **cout << "Rectangle Area: " << shape->calculateArea() << endl;**  **cout << "------------------------------";**  **shape = &circ;**  **cout << "Circle Area: " << shape->calculateArea() << endl;**  **return 0;**  **}** |

**Output:**

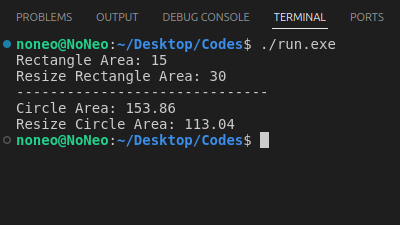


**b)**

**Source code:**

|  |
| --- |
| **#include <iostream>**  **using namespace std;**  **class Shape**  **{**  **public:**  **virtual double calculateArea() const = 0;**  **virtual void resize(double len, double wid) {}**  **virtual void resize(double r) {}**  **};**  **class Rectangle : public Shape**  **{**  **private:**  **double length;**  **double width;**  **public:**  **Rectangle(double l = 0, double w = 0) : length(l), width(w) {}**  **void resize(double len, double wid ) {**  **length = len;**  **width = wid;**  **}**  **double calculateArea() const**  **{**  **return length \* width;**  **}**  **};**  **class Circle : public Shape**  **{**  **private:**  **double radius;**  **public:**  **Circle(double r = 0) : radius(r) {}**  **void resize(double r) {**  **radius = r;**  **}**  **double calculateArea() const**  **{**  **return 3.14 \* radius \* radius;**  **}**  **};**  **int main()**  **{**  **Rectangle rect(5, 3);**  **Circle circ(7);**  **Shape\* shape;**  **shape = &rect;**  **cout << "Rectangle Area: " << shape->calculateArea() << endl;**  **rect.resize(6, 5);**  **cout << "Resize Rectangle Area: " << shape->calculateArea() << endl;**  **cout << "------------------------------" << endl;**  **shape = &circ;**  **cout << "Circle Area: " << shape->calculateArea() << endl;**  **circ.resize(6);**  **cout << "Resize Circle Area: " << shape->calculateArea() << endl;**  **return 0;**  **}** |

**Output:**



**Question 3: Encapsulation and Access Modifiers (5 marks)**

**a)**

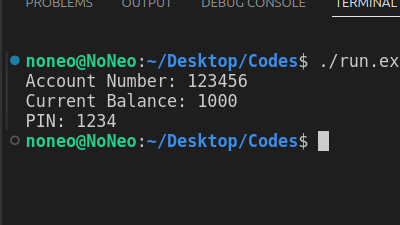
// a) Create a class called BankAccount with private member variables accountNumber (integer),

//balance (double), and

// pin (string). Provide appropriate accessor and mutator functions to access and modify these variables.

|  |
| --- |
| **#include <iostream>**  **using namespace std;**  **class BankAccount**  **{**  **int accountNumber;**  **double balance;**  **string pin;**  **public:**  **BankAccount() {}**  **BankAccount(int account\_Number, double bal, string PIN): accountNumber(account\_Number), balance(bal), pin(PIN) {}**  **int getAccountNumber()**  **{**  **return accountNumber;**  **}**  **double getBalance()**  **{**  **return balance;**  **}**  **string getPin()**  **{**  **return pin;**  **}**  **void setBalance (double newBalance)**  **{**  **balance = newBalance;**  **}**  **void setPin (string newPin)**  **{**  **pin = newPin;**  **}**  **};**  **int main()**  **{**  **BankAccount a(123456, 1000.0, "1234");**  **cout << "Account Number: " << a.getAccountNumber() << endl;**  **cout << "Current Balance: " << a.getBalance() << endl;**  **cout << "PIN: " << a.getPin() << endl;**  **a.setBalance(1500.0);**  **a.setPin("4321");**  **return 0;**  **}** |

**Output:**



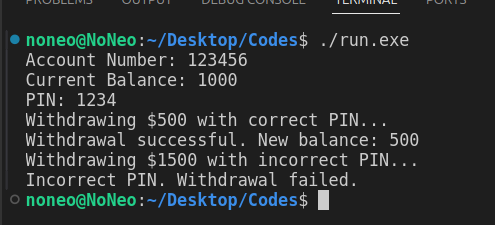
**b)**

Implement a public member function withdraw(double amount) in the BankAccount class to withdraw money from the account. Apply appropriate encapsulation techniques to ensure that the withdrawal is only allowed if the correct pin is provided and if the account has sufficient balance.

Source code:

|  |
| --- |
| #include <iostream>  using namespace std;    class BankAccount  {  int accountNumber;  double balance;  string pin;    public:  BankAccount() {}  BankAccount(int account\_Number, double bal, string PIN) : accountNumber(account\_Number), balance(bal), pin(PIN) {}    int getAccountNumber()  {  return accountNumber;  }  double getBalance()  {  return balance;  }  string getPin()  {  return pin;  }    void setBalance(double newBalance)  {  balance = newBalance;  }  void setPin(string newPin)  {  pin = newPin;  }    void withdraw(double amount, string enteredPin)  {  if (enteredPin == pin)  {  if (amount > 0 && amount <= balance)  {  balance -= amount;  cout << "Withdrawal successful. New balance: " << balance << endl;  }  else  {  cout << "Insufficient balance." << endl;  }  }  else  {  cout << "Incorrect PIN. Withdrawal failed." << endl;  }  }  };    int main()  {  BankAccount a(123456, 1000.0, "1234");    cout << "Account Number: " << a.getAccountNumber() << endl;  cout << "Current Balance: " << a.getBalance() << endl;  cout << "PIN: " << a.getPin() << endl;    cout << "Withdrawing $500 with correct PIN..." << endl;  a.withdraw(500, "1234");    cout << "Withdrawing $1500 with incorrect PIN..." << endl;  a.withdraw(1500, "4321");    return 0;  } |

Output:



c)

Create an object of BankAccount and demonstrate the encapsulation and withdrawal functionality by calling the appropriate functions.

Source code:

|  |
| --- |
| #include <iostream>  using namespace std;    class BankAccount  {  int accountNumber;  double balance;  string pin;    public:  BankAccount() {}  BankAccount(int account\_Number, double bal, string PIN) : accountNumber(account\_Number), balance(bal), pin(PIN) {}    int getAccountNumber()  {  return accountNumber;  }  double getBalance()  {  return balance;  }  string getPin()  {  return pin;  }    void setBalance(double newBalance)  {  balance = newBalance;  }  void setPin(string newPin)  {  pin = newPin;  }    void withdraw(double amount, string enteredPin)  {  if (enteredPin == pin)  {  if (amount > 0 && amount <= balance)  {  balance -= amount;  cout << "Withdrawal successful. New balance: " << balance << endl;  }  else  {  cout << "Insufficient balance." << endl;  }  }  else  {  cout << "Incorrect PIN. Withdrawal failed." << endl;  }  }  };    int main()  {  BankAccount a(123456, 1000.0, "1234");    cout << "Account Number: " << a.getAccountNumber() << endl;  cout << "Current Balance: " << a.getBalance() << endl;  cout << "PIN: " << a.getPin() << endl;    cout << "Withdrawing $500 with correct PIN..." << endl;  a.withdraw(500, "1234");    cout << "Withdrawing $1500 with incorrect PIN..." << endl;  a.withdraw(1500, "4321");    return 0;  } |

Output:

