**18-programs**

**1-**Create a base class Shape with a member function getArea(). Derive classes Rectangle and Circle from Shape. Implement the getArea() function in each derived class to calculate the area of a rectangle and a circle, respectively.

#include <iostream>

class Shape {

public:

virtual double getArea() const = 0;

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

Rectangle(double \_length, double \_width) : length(\_length), width(\_width) {}

double getArea() const override {

return length \* width;

}

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double \_radius) : radius(\_radius) {}

double getArea() const override {

return 3.14159265358979323846 \* radius \* radius;

}

};

int main() {

Rectangle rectangle(5, 4);

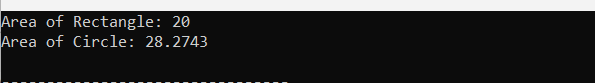
Circle circle(3);

std::cout << "Area of Rectangle: " << rectangle.getArea() << std::endl;

std::cout << "Area of Circle: " << circle.getArea() << std::endl;

return 0;

}



2-**.** Define a base class Animal with a virtual function makeSound(). Derive classes Dog and Cat from Animal. Implement the makeSound() function in each derived class to output appropriate sounds for a dog and a cat.

#include <iostream>

using namespace std;

class Animal {

public:

virtual void makeSound() const {

cout << "Some generic sound from an animal" <<endl;

}

};

class Dog : public Animal {

public:

void makeSound() const override {

cout << "Woof! Woof!" <<endl;

}

};

class Cat : public Animal {

public:

void makeSound() const override {

cout << "Meow! Meow!" <<endl;

}

};

int main() {

Animal \*animal1 = new Dog();

Animal \*animal2 = new Cat();

animal1->makeSound();

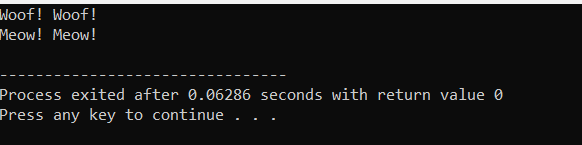
animal2->makeSound();

delete animal1;

delete animal2;

return 0;

}



3-Create a base class **Employee** with attributes **name** and **salary**. Derive classes **Manager** and **Worker** from **Employee**. Add an extra attribute **department** for the **Manager** class. Implement constructors in each class to initialize the attributes.

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

double salary;

public:

Employee(const string& name, double salary) : name(name), salary(salary) {}

void display() {

cout << "Name: " << name << ", Salary: $" << salary << endl;

}

};

class Manager : public Employee {

private:

string department;

public:

Manager(const string& name, double salary, const string& department) : Employee(name, salary), department(department) {}

void display() {

cout << "Name: " << name << ", Salary: $" << salary << ", Department: " << department << endl;

}

};

class Worker : public Employee {

public:

Worker(const string& name, double salary) : Employee(name, salary) {}

};

int main() {

Manager manager("John Doe", 50000.0, "Engineering");

Worker worker("Jane Smith", 30000.0);

cout << "Manager: ";

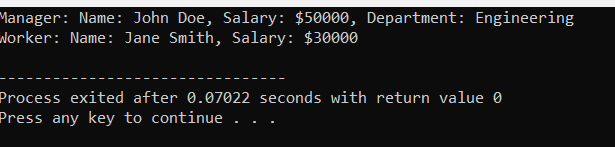
manager.display();

cout << "Worker: ";

worker.display();

return 0;

}



4-Define a base class **Shape** with a virtual function **draw()**. Derive classes **Circle**, **Square**, and **Triangle** from **Shape**. Implement the **draw()** function in each derived class to draw respective shapes using cout statements.

#include <iostream>

using namespace std;

class Shape {

public:

virtual void draw() const = 0; // Pure virtual function

};

class Circle : public Shape {

public:

void draw() const override {

cout << "Drawing a circle." << endl;

}

};

class Square : public Shape {

public:

void draw() const override {

cout << "Drawing a square." << endl;

}

};

class Triangle : public Shape {

public:

void draw() const override {

cout << "Drawing a triangle." << endl;

}

};

int main() {

Circle circle;

Square square;

Triangle triangle;

cout << "Drawing shapes:" << endl;

cout << "--------------" << endl;

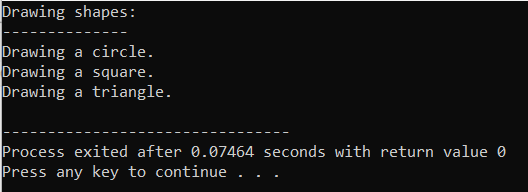
circle.draw();

square.draw();

triangle.draw();

return 0;

}



5-Create a base class **Vehicle** with attributes **speed** and **fuel**. Derive classes **Car** and **Bike** from **Vehicle**. Implement constructors in each class to initialize the attributes.

#include <iostream>

class Vehicle {

protected:

int speed;

int fuel;

public:

Vehicle(int \_speed = 0, int \_fuel = 100) : speed(\_speed), fuel(\_fuel) {}

void accelerate(int amount) {

speed += amount;

}

void decelerate(int amount) {

speed -= amount;

}

void refuel(int amount) {

fuel += amount;

}

int getSpeed() const {

return speed;

}

int getFuel() const {

return fuel;

}

};

class Car : public Vehicle {

private:

int numWheels;

public:

Car(int \_speed = 0, int \_fuel = 100, int \_numWheels = 4)

: Vehicle(\_speed, \_fuel), numWheels(\_numWheels) {}

int getNumWheels() const {

return numWheels;

}

};

class Bike : public Vehicle {

private:

int numWheels;

public:

Bike(int \_speed = 0, int \_fuel = 100, int \_numWheels = 2)

: Vehicle(\_speed, \_fuel), numWheels(\_numWheels) {}

int getNumWheels() const {

return numWheels;

}

};

int main() {

Car car;

std::cout << "Car speed: " << car.getSpeed() << std::endl;

std::cout << "Car fuel: " << car.getFuel() << std::endl;

std::cout << "Car wheels: " << car.getNumWheels() << std::endl;

Bike bike;

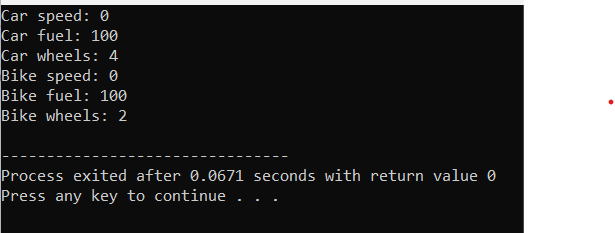
std::cout << "Bike speed: " << bike.getSpeed() << std::endl;

std::cout << "Bike fuel: " << bike.getFuel() << std::endl;

std::cout << "Bike wheels: " << bike.getNumWheels() << std::endl;

return 0;

}



1. 6-**.** Define a base class Bird with a virtual function fly(). Derive classes Eagle and Sparrow from Bird. Implement the fly() function in each derived class to output flying behavior.

#include <iostream>

using namespace std;

class Bird {

public:

virtual void fly() {

cout << "Bird flying\n";

}

};

class Eagle : public Bird {

public:

void fly() override {

cout << "Eagle soaring high\n";

}

};

class Sparrow : public Bird {

public:

void fly() override {

cout << "Sparrow fluttering around\n";

}

};

int main() {

Bird\* bird1 = new Eagle();

bird1->fly();

Bird\* bird2 = new Sparrow();

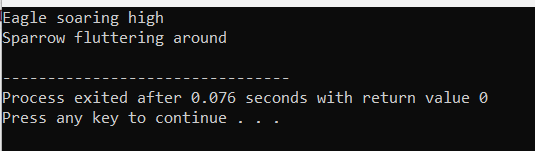
bird2->fly();

delete bird1;

delete bird2;

return 0;

}



7-Create an abstract base class **Shape** with a pure virtual function **calculateArea()**. Derive classes **Rectangle** and **Circle** from **Shape** and implement the **calculateArea()** function in each derived class.

#include <iostream>

class Shape {

public:

virtual float calculateArea() const = 0;

};

class Rectangle : public Shape {

private:

float length;

float width;

public:

Rectangle(float \_length, float \_width) : length(\_length), width(\_width) {}

float calculateArea() const override {

return length \* width;

}

};

class Circle : public Shape {

private:

float radius;

public:

Circle(float \_radius) : radius(\_radius) {}

float calculateArea() const override {

return 3.14159 \* radius \* radius;

}

};

int main() {

Rectangle rectangle(5, 3);

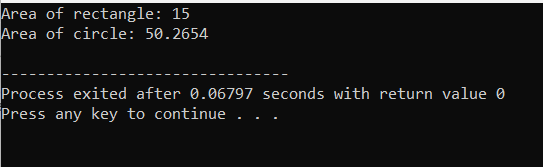
std::cout << "Area of rectangle: " << rectangle.calculateArea() << std::endl;

Circle circle(4);

std::cout << "Area of circle: " << circle.calculateArea() << std::endl;

return 0;

}



8-

#include <iostream>

#include <string>

class Person {

protected:

std::string name;

int age;

public:

Person(const std::string& \_name, int \_age) : name(\_name), age(\_age) {}

void display() const {

std::cout << "Name: " << name << ", Age: " << age;

}

};

class Student : public Person {

private:

int grade;

public:

Student(const std::string& \_name, int \_age, int \_grade) : Person(\_name, \_age), grade(\_grade) {}

void display() const {

Person::display();

std::cout << ", Grade: " << grade << std::endl;

}

};

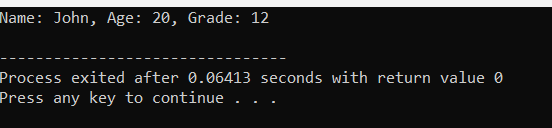
int main() {

Student student("John", 20, 12);

student.display();

return 0;

}



9-Write a C++ program to swap two integers using pointers.

#include <iostream>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int num1 = 5;

int num2 = 10;

std::cout << "Before swapping: " << std::endl;

std::cout << "num1 = " << num1 << ", num2 = " << num2 << std::endl;

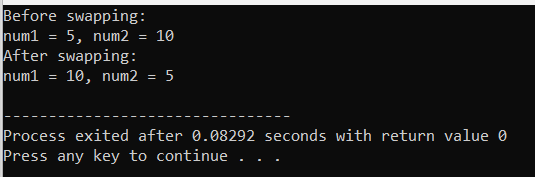
swap(&num1, &num2);

std::cout << "After swapping: " << std::endl;

std::cout << "num1 = " << num1 << ", num2 = " << num2 << std::endl;

return 0;

}



10-Create a class **Circle** with a private member **radius**. Write a C++ program to find the area of a circle using a pointer to the object

#include <iostream>

using namespace std;

class Circle {

private:

double radius;

public:

Circle(double \_radius) : radius(\_radius) {}

double calculateArea() const {

return 3.14159 \* radius \* radius;

}

double getRadius() const {

return radius;

}

};

int main() {

Circle\* circlePtr = new Circle(5.0);

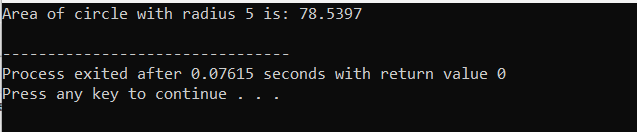
double area = circlePtr->calculateArea();

std::cout << "Area of circle with radius " << circlePtr->getRadius() << " is: " << area << std::endl;

delete circlePtr;

return 0;

}



11-Define a class **Rectangle** with private members **length** and **width**. Write a C++ program to calculate the perimeter of a rectangle using a pointer to an object.

#include <iostream>

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double \_length, double \_width) : length(\_length), width(\_width) {}

double calculatePerimeter() const {

return 2 \* (length + width);

}

double getLength() const {

return length;

}

double getWidth() const {

return width;

}

};

int main() {

Rectangle\* rectanglePtr = new Rectangle(5.0, 3.0);

double perimeter = rectanglePtr->calculatePerimeter();

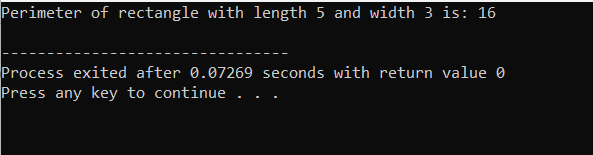
std::cout << "Perimeter of rectangle with length " << rectanglePtr->getLength()

<< " and width " << rectanglePtr->getWidth() << " is: " << perimeter << std::endl;

delete rectanglePtr;

return 0;

}



12-Implement a class **Employee** with private member variables **name** and **salary**. Write a C++ program to display the details of an employee using a pointer to an object.

#include <iostream>

#include <string>

using namespace std;

class Employee {

private:

string name;

double salary;

public:

Employee(string n, double s) : name(n), salary(s) {}

string getName() const {

return name;

}

double getSalary() const {

return salary;

}

};

int main() {

Employee\* empPtr = new Employee("John Doe", 50000.0);

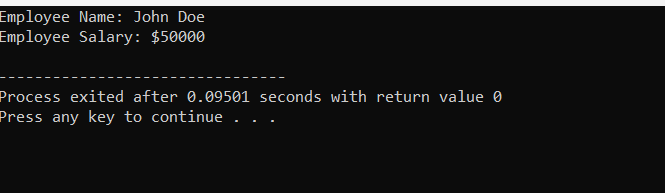
cout << "Employee Name: " << empPtr->getName() << endl;

cout << "Employee Salary: $" << empPtr->getSalary() << endl;

delete empPtr;

return 0;

}



13-Create a class **Student** with private member variables **name** and **grade**. Write a C++ program to display the name and grade of a student using the **this** pointer.

#include <iostream>

#include <string>

class Student {

private:

std::string name;

char grade;

public:

Student(std::string name, char grade) {

this->name = name;

this->grade = grade;

}

void display() {

std::cout << "Name: " << this->name << std::endl;

std::cout << "Grade: " << this->grade << std::endl;

}

};

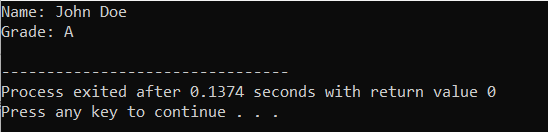
int main() {

Student student1("John Doe", 'A');

student1.display();

return 0;

}



14-Define a base class **Shape** with a virtual function **draw()**. Implement a derived class **Circle** with its own implementation of the **draw()** function. Write a C++ program to demonstrate polymorphism using a pointer to the base class.

#include <iostream>

class Shape {

public:

virtual void draw() {

std::cout << "Drawing a shape" << std::endl;

}

};

class Circle : public Shape {

public:

void draw() override {

std::cout << "Drawing a circle" << std::endl;

}

};

int main() {

Shape \*shapePtr;

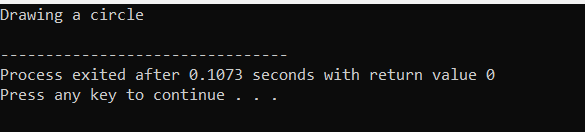
Circle circle;

shapePtr = &circle;

shapePtr->draw();

return 0;

}



15-Create a class **Person** with private member variables **name** and **age**. Implement a derived class **Student** with its own implementation of the **display()** function to display the name and age of a student. Write a C++ program to demonstrate function overriding using a pointer to the base class.

#include <iostream>

#include <string>

class Person {

protected:

std::string name;

int age;

public:

Person(std::string name, int age) : name(name), age(age) {}

virtual void display() {

std::cout << "Name: " << name << std::endl;

std::cout << "Age: " << age << std::endl;

}

};

class Student : public Person {

public:

Student(std::string name, int age) : Person(name, age) {}

void display() override {

std::cout << "Student Name: " << name << std::endl;

std::cout << "Student Age: " << age << std::endl;

}

};

int main() {

Person \*personPtr;

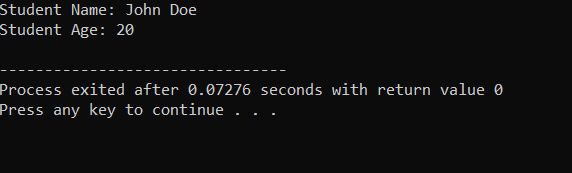
Student student("John Doe", 20);

personPtr = &student;

personPtr->display();

return 0;

}



16-Define a base class Vehicle with attributes speed and fuel. Derive classes Car and Bike from Vehicle. Implement constructors in each class to initialize the attributes. Also, create a member function displayInfo() in each derived class to display the speed and fuel of the vehicle.

#include <iostream>

using namespace std;

class Vehicle {

protected:

int speed;

int fuel;

public:

Vehicle(int \_speed, int \_fuel) : speed(\_speed), fuel(\_fuel) {}

virtual void displayInfo() const {

cout << "Vehicle Speed: " << speed << " km/h" <<endl;

cout << "Fuel Level: " << fuel << " %" <<endl;

}

};

class Car : public Vehicle {

public:

Car(int \_speed, int \_fuel) : Vehicle(\_speed, \_fuel) {}

void displayInfo() const override {

cout << "Car Speed: " << speed << " km/h" <<endl;

cout << "Car Fuel Level: " << fuel << " %" <<endl;

}

};

class Bike : public Vehicle {

public:

Bike(int \_speed, int \_fuel) : Vehicle(\_speed, \_fuel) {}

void displayInfo() const override {

cout << "Bike Speed: " << speed << " km/h" <<endl;

cout << "Bike Fuel Level: " << fuel << " %" <<endl;

}

};

int main() {

Car car(120, 70);

Bike bike(80, 50);

cout << "Car Information:" <<endl;

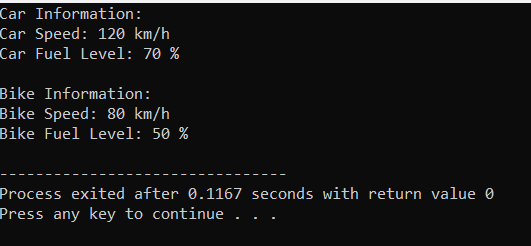
car.displayInfo();

cout << "\nBike Information:" <<endl;

bike.displayInfo();

return 0;

}



17-Create a base class Shape with virtual functions area() and perimeter(). Derive classes Rectangle and Circle from Shape. Implement these functions in each derived class to calculate the area and perimeter of a rectangle and a circle, respectively.

#include <iostream>

class Shape {

public:

virtual double area() const = 0;

virtual double perimeter() const = 0;

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double area() const override {

return length \* width;

}

double perimeter() const override {

return 2 \* (length + width);

}

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double area() const override {

return 3.14159 \* radius \* radius;

}

double perimeter() const override {

return 2 \* 3.14159 \* radius;

}

};

int main() {

Rectangle rectangle(5, 4);

Circle circle(3);

std::cout << "Rectangle:" << std::endl;

std::cout << "Area: " << rectangle.area() << std::endl;

std::cout << "Perimeter: " << rectangle.perimeter() << std::endl;

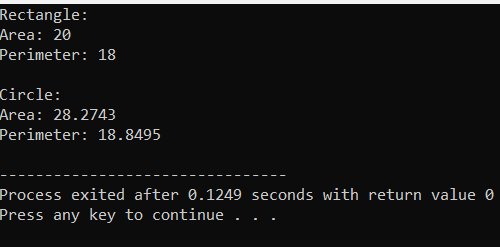
std::cout << "\nCircle:" << std::endl;

std::cout << "Area: " << circle.area() << std::endl;

std::cout << "Perimeter: " << circle.perimeter() << std::endl;

return 0;

}



18-Create a base class Employee with attributes name, id, and salary. Derive classes Manager and Worker from Employee. Implement constructors in each class to initialize the attributes. Also, implement member functions to display the details of each type of employee.

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

int emp\_id;

double salary;

public:

Employee(string name, int emp\_id, double salary) : name(name), emp\_id(emp\_id), salary(salary) {}

virtual void display\_details() {

cout << "Name: " << name << endl;

cout << "Employee ID: " << emp\_id << endl;

cout << "Salary: $" << salary << endl;

}

};

class Manager : public Employee {

private:

string department;

public:

Manager(string name, int emp\_id, double salary, string department) : Employee(name, emp\_id, salary), department(department) {}

void display\_details() override {

Employee::display\_details();

cout << "Department: " << department << endl;

}

};

class Worker : public Employee {

private:

int hours\_worked;

public:

Worker(string name, int emp\_id, double salary, int hours\_worked) : Employee(name, emp\_id, salary), hours\_worked(hours\_worked) {}

void display\_details() override {

Employee::display\_details();

cout << "Hours Worked: " << hours\_worked << endl;

}

};

int main() {

Manager manager1("John Doe", 1001, 60000, "Sales");

Worker worker1("Jane Smith", 2001, 35000, 40);

cout << "Manager Details:" << endl;

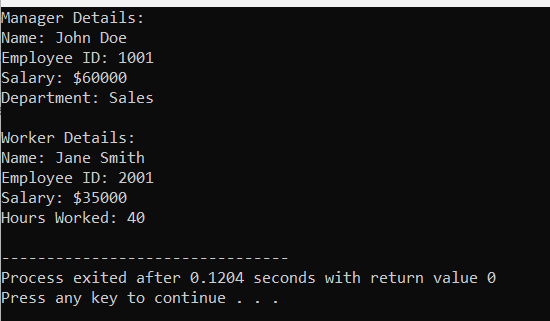
manager1.display\_details();

cout << "\nWorker Details:" << endl;

worker1.display\_details();

return 0;

}



19-Define a base class Shape with a virtual function draw(). Derive classes Square, Circle, and Triangle from Shape. Implement the draw() function in each derived class to draw respective shapes using cout statements.

#include <iostream>

using namespace std;

class Shape {

public:

virtual void draw() = 0;

};

class Square : public Shape {

public:

void draw() override {

cout << "Drawing a square." << endl;

}

};

class Circle : public Shape {

public:

void draw() override {

cout << "Drawing a circle." << endl;

}

};

class Triangle : public Shape {

public:

void draw() override {

cout << "Drawing a triangle." << endl;

}

};

int main() {

Square square;

Circle circle;

Triangle triangle;

cout << "Drawing shapes:" << endl;

cout << "----------------" << endl;

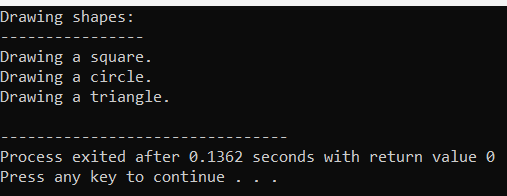
square.draw();

circle.draw();

triangle.draw();

return 0;

}



20-Create a base class Person with attributes name and age. Derive class Student from Person with an additional attribute grade. Implement constructors in each class to initialize the attributes. Also, include a member function displayInfo() in each class to display the details of the person or student.

#include <iostream>

#include <string>

using namespace std;

class Person {

protected:

string name;

int age;

public:

Person(string name, int age) : name(name), age(age) {}

virtual void displayInfo() {

cout << "Name: " << name << endl;

cout << "Age: " << age << endl;

}

};

class Student : public Person {

private:

int grade;

public:

Student(string name, int age, int grade) : Person(name, age), grade(grade) {}

void displayInfo() override {

Person::displayInfo();

cout << "Grade: " << grade << endl;

}

};

int main() {

// Example usage

Person person("John", 25);

Student student("Jane", 20, 11);

cout << "Person's Info:" << endl;

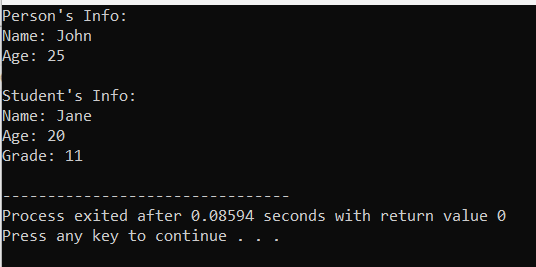
person.displayInfo();

cout << "\nStudent's Info:" << endl;

student.displayInfo();

return 0;

}



21-Define a base class BankAccount with attributes accountNumber and balance. Derive classes SavingsAccount and CurrentAccount from BankAccount. Implement constructors in each class to initialize the attributes. Also, include member functions to deposit and withdraw money.

#include <iostream>

class BankAccount {

protected:

std::string accountNumber;

double balance;

public:

BankAccount(const std::string& accNumber, double initialBalance) : accountNumber(accNumber), balance(initialBalance) {}

void deposit(double amount) {

balance += amount;

std::cout << "Deposited: $" << amount << std::endl;

std::cout << "Current Balance: $" << balance << std::endl;

}

virtual void withdraw(double amount) {

if (balance >= amount) {

balance -= amount;

std::cout << "Withdrawn: $" << amount << std::endl;

std::cout << "Current Balance: $" << balance << std::endl;

} else {

std::cout << "Insufficient balance." << std::endl;

}

}

};

class SavingsAccount : public BankAccount {

public:

SavingsAccount(const std::string& accNumber, double initialBalance) : BankAccount(accNumber, initialBalance) {}

void withdraw(double amount) override {

if (balance - amount >= 100.0) {

BankAccount::withdraw(amount);

} else {

std::cout << "Minimum balance requirement not met." << std::endl;

}

}

};

class CurrentAccount : public BankAccount {

public:

CurrentAccount(const std::string& accNumber, double initialBalance) : BankAccount(accNumber, initialBalance) {}

};

int main() {

SavingsAccount savings("SAV-12345", 1000.0);

CurrentAccount current("CUR-67890", 2000.0);

savings.deposit(500.0);

savings.withdraw(200.0);

current.deposit(1000.0);

current.withdraw(3000.0);

return 0;

}

