**Slip 9: Sample Solutions and Explanations**

**Q1. Circle Class: Area and Circumference**

**Approach**

* Define a Circle class with a private radius attribute.
* Provide member functions to calculate area and circumference.
* Accept radius from the user and display the results.

**Code**

#include <iostream>  
using namespace std;  
  
// [Circle Class Definition]  
class Circle {  
 double radius;  
public:  
 void accept() { cout << "Enter radius: "; cin >> radius; }  
 double area() { return 3.14159 \* radius \* radius; }  
 double circumference() { return 2 \* 3.14159 \* radius; }  
};  
  
int main() {  
 Circle c;  
 c.accept();  
 cout << "Area: " << c.area() << endl;  
 cout << "Circumference: " << c.circumference() << endl;  
 return 0;  
}

**Explanation**

* The Circle class encapsulates the radius and provides methods for area and circumference.
* The accept method reads the radius from the user.
* The results are displayed in main.

**Syntax Definitions**

* **class**: A user-defined type that groups data and functions.
* **double**: A data type for floating-point numbers.

**Q2. MyPoint Class (with Copy Constructor)**

**Approach**

* Define a MyPoint class with two coordinates (x, y).
* Implement a copy constructor to initialize a new object from an existing one.
* Demonstrate copying in main.

**Code**

#include <iostream>  
using namespace std;  
  
// [MyPoint Class Definition]  
class MyPoint {  
 int x, y;  
public:  
 MyPoint(int a, int b) : x(a), y(b) {}  
 MyPoint(const MyPoint &p) { x = p.x; y = p.y; }  
 void display() { cout << "x: " << x << " y: " << y << endl; }  
};  
  
int main() {  
 MyPoint p1(3,4);  
 MyPoint p2 = p1; // Copy  
 p1.display(); p2.display();  
 return 0;  
}

**Explanation**

* The MyPoint class has a parameterized constructor and a copy constructor.
* MyPoint p2 = p1; uses the copy constructor to create a new object with the same values as p1.
* The display method prints the coordinates.

**Syntax Definitions**

* **Copy Constructor**: A special constructor that initializes an object using another object of the same class.
* **Member Initializer List**: The : x(a), y(b) syntax initializes members before the constructor body runs.

**Q3. Financial Application: Money Class with Operator Overloading**

**Approach**

* Define a Money class with rupees and paise.
* Overload the + operator to add two Money objects, normalizing paise to rupees.
* Provide a display method.

**Code**

#include <iostream>  
using namespace std;  
  
// [Money Class Definition]  
class Money {  
 int rupees, paise;  
public:  
 Money(int r=0, int p=0): rupees(r), paise(p) { normalize(); }  
 void normalize() { rupees += paise/100; paise %= 100; }  
 Money operator+(const Money& m) { return Money(rupees + m.rupees, paise + m.paise); }  
 void display() { cout << rupees << " Rupees " << paise << " Paise\n"; }  
};  
  
int main() {  
 Money m1(7,190), m2(3,50), m3;  
 m3 = m1 + m2;  
 m3.display();  
 return 0;  
}

**Explanation**

* The Money class handles currency addition and normalization (e.g., 190 paise = 1 rupee 90 paise).
* The + operator is overloaded to add two Money objects.
* The display method prints the result in a readable format.

**Syntax Definitions**

* **operator+**: Overloads the + operator for user-defined types.
* **Constructor with default arguments**: Allows object creation with or without parameters.
* **Normalization**: Adjusts paise to rupees if paise >= 100.