**TEACHERS ASSESMENT**

**NAME-**Reva .A. Jaiswal

**ROLL NO-07 SECTION-**O3

**Q1: Animal & Cat (override makeSound() to bark)**

**Answer**-

class Animal {

void makeSound() {

System.out.println("Animal makes a sound");

}

}

class Cat extends Animal {

@Override

void makeSound() {

System.out.println("Bark");

}

}

**Q2: Vehicle & Car (override drive() method)**

**Answer-**

class Vehicle {

void drive() {

System.out.println("Vehicle is being driven");

}

}

class Car extends Vehicle {

@Override

void drive() {

System.out.println("Repairing a car");

}

}

**Q3: Shape & Rectangle (override getArea())**

**Answer-**

class Shape {

double getArea() {

return 0;

}

}

class Rectangle extends Shape {

double length = 5;

double width = 3;

@Override

double getArea() {

return length \* width;

}

}

**Q4: Employee & HRManager**

**Answer**-

class Employee {

void work() {

System.out.println("Employee working");

}

double getSalary() {

return 50000;

}

}

class HRManager extends Employee {

@Override

void work() {

System.out.println("HRManager is recruiting");

}

void addEmployee() {

System.out.println("Adding a new employee");

}

}

**Q5: BankAccount & SavingsAccount**

**Answer-**

class BankAccount {

double balance = 500;

void deposit(double amount) {

balance += amount;

}

void withdraw(double amount) {

balance -= amount;

}

}

class SavingsAccount extends BankAccount {

@Override

void withdraw(double amount) {

if (balance - amount < 100) {

System.out.println("Withdrawal denied: Minimum balance of 100 required.");

} else {

balance -= amount;

}

}

}

**Q6: Animal & Cheetah (move() method override)**

**Answer-**

class Animal {

void move() {

System.out.println("Animal moves");

}

}

class Cheetah extends Animal {

@Override

void move() {

System.out.println("Cheetah runs swiftly");

}

}

**Q7: Person & Employee with overridden getLastName()**

**Answer-**

class Person {

String getFirstName() {

return "John";

}

String getLastName() {

return "Doe";

}

}

class Employee extends Person {

String jobTitle = "Software Engineer";

String getEmployeeId() {

return "EMP123";

}

@Override

String getLastName() {

return super.getLastName() + " - " + jobTitle;

}

}

**Q8: Shape & Circle with area and perimeter**

**Answer-**

class Shape {

double getPerimeter() {

return 0;

}

double getArea() {

return 0;

}

}

class Circle extends Shape {

double radius = 7;

@Override

double getPerimeter() {

return 2 \* Math.PI \* radius;

}

@Override

double getArea() {

return Math.PI \* radius \* radius;

}

}

**Q9: Vehicle Hierarchy**

**Answer-**

class Vehicle {

String make, model, fuelType;

int year;

double calculateFuelEfficiency() {

return 0;

}

double distanceTraveled(double time, double speed) {

return time \* speed;

}

double getMaxSpeed() {

return 0;

}

}

class Truck extends Vehicle {

@Override

double calculateFuelEfficiency() {

return 8.0;

}

@Override

double getMaxSpeed() {

return 100;

}

}

class Car extends Vehicle {

@Override

double calculateFuelEfficiency() {

return 15.0;

}

@Override

double getMaxSpeed() {

return 180;

}

}

class Motorcycle extends Vehicle {

@Override

double calculateFuelEfficiency() {

return 25.0;

}

@Override

double getMaxSpeed() {

return 160;

} }

**Q10: Employee Hierarchy**

**Answer-**

class Employee {

String name, address, jobTitle;

double salary;

double calculateBonus() {

return salary \* 0.1;

}

void generateReport() {

System.out.println("Performance report for " + name);

}

}

class Manager extends Employee {

void manageProject() {

System.out.println("Managing projects");

}

}

class Developer extends Employee {

void developSoftware() {

System.out.println("Developing software");

}

}

class Programmer extends Employee {

void writeCode() {

System.out.println("Writing code"); } }

**JOURNAL CODE-**

**Experiment 1 – Class and Object**

**Part A: Basic Class**

**class Student {**

**public**

**string name;**

**int roll;**

**void getDetails() {**

**cout << "Enter name: ";**

**cin >> name;**

**cout << "Enter roll no: ";**

**cin >> roll;**

**}**

**void showDetails() {**

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

**cout << "Roll No: " << roll << endl;**

**}**

**};**

**int main() {**

**Student s;**

**s.getDetails();**

**s.showDetails();**

**return 0;**

**}**

**Part B: Multiple Objects**

**class Student {**

**public:**

**string name;**

**int roll;**

**void getDetails() {**

**cout << "Enter name: ";**

**cin >> name;**

**cout << "Enter roll no: ";**

**cin >> roll;**

**}**

**void showDetails() {**

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

**cout << "Roll No: " << roll << endl;**

**}**

**};**

**int main() {**

**Student s1, s2;**

**s1.getDetails();**

**s2.getDetails();**

**s1.showDetails();**

**s2.showDetails();**

**return 0;**

**}**

**---**

**Experiment 2 – Constructor and Destructor**

**Part A: Default Constructor**

**#include<iostream>**

**using namespace std;**

**class Demo {**

**public:**

**Demo() {**

**cout << "Default constructor called" << endl;**

**}**

**};**

**int main() {**

**Demo d;**

**return 0;**

**}**

**Part B: Parameterized Constructor**

**#include<iostream>**

**using namespace std;**

**class Demo {**

**int x;**

**public:**

**Demo(int a) {**

**x = a;**

**cout << "Value: " << x << endl;**

**}**

**};**

**int main() {**

**Demo d(10);**

**return 0;**

**}**

**Part C: Constructor Overloading**

**#include<iostream>**

**using namespace std;**

**class Demo {**

**int a, b;**

**public:**

**Demo() {**

**a = 0; b = 0;**

**}**

**Demo(int x, int y) {**

**a = x; b = y;**

**}**

**void display() {**

**cout << "a = " << a << ", b = " << b << endl;**

**}**

**};**

**int main() {**

**Demo d1, d2(10, 20);**

**d1.display();**

**d2.display();**

**return 0;**

**}**

**Part D: Destructor**

**#include<iostream>**

**using namespace std;**

**class Demo {**

**public:**

**Demo() {**

**cout << "Constructor called" << endl;**

**}**

**~Demo() {**

**cout << "Destructor called" << endl;**

**}**

**};**

**int main() {**

**Demo d;**

**return 0;**

**}**

**---**

**Experiment 3 – Inheritance**

**Part A: Single Inheritance**

**#include<iostream>**

**using namespace std;**

**class A {**

**public:**

**void showA() {**

**cout << "Base class A" << endl;**

**}**

**};**

**class B : public A {**

**public:**

**void showB() {**

**cout << "Derived class B" << endl;**

**}**

**};**

**int main() {**

**B obj;**

**obj.showA();**

**obj.showB();**

**return 0;**

**}**

**Part B: Multilevel Inheritance**

**#include<iostream>**

**using namespace std;**

**class A {**

**public:**

**void showA() {**

**cout << "A class" << endl;**

**}**

**};**

**class B : public A {**

**public:**

**void showB() {**

**cout << "B class" << endl;**

**}**

**};**

**class C : public B {**

**public:**

**void showC() {**

**cout << "C class" << endl;**

**}**

**};**

**int main() {**

**C obj;**

**obj.showA();**

**obj.showB();**

**obj.showC();**

**return 0;**

**}**

**Part C: Multiple Inheritance**

**#include<iostream>**

**using namespace std;**

**class A {**

**public:**

**void showA() {**

**cout << "Class A" << endl;**

**}**

**};**

**class B {**

**public:**

**void showB() {**

**cout << "Class B" << endl;**

**}**

**};**

**class C : public A, public B {**

**public:**

**void showC() {**

**cout << "Class C" << endl;**

**}**

**};**

**int main() {**

**C obj;**

**obj.showA();**

**obj.showB();**

**obj.showC();**

**return 0;**

**}**

**---**

**Experiment 4 – Polymorphism**

**Part A: Function Overloading**

**#include<iostream>**

**using namespace std;**

**class Print {**

**public:**

**void show(int i) {**

**cout << "Integer: " << i << endl;**

**}**

**void show(double d) {**

**cout << "Double: " << d << endl;**

**}**

**void show(string s) {**

**cout << "String: " << s << endl;**

**}**

**};**

**int main() {**

**Print p;**

**p.show(5);**

**p.show(3.14);**

**p.show("Hello");**

**return 0;**

**}**

**Part B: Function Overriding**

**#include<iostream>**

**using namespace std;**

**class Base {**

**public:**

**void show() {**

**cout << "Base class show" << endl;**

**}**

**};**

**class Derived : public Base {**

**public:**

**void show() {**

**cout << "Derived class show" << endl;**

**}**

**};**

**int main() {**

**Derived d;**

**d.show();**

**return 0;**

**}**

**---**

**Experiment 5 – Abstract Class and Virtual Function**

**Part A: Abstract Class**

**#include<iostream>**

**using namespace std;**

**class Shape {**

**public:**

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

**};**

**class Circle : public Shape {**

**public:**

**void draw() {**

**cout << "Drawing Circle" << endl;**

**}**

**};**

**int main() {**

**Circle c;**

**c.draw();**

**return 0;**

**}**

**Part B: Virtual Function**

**#include<iostream>**

**using namespace std;**

**class Base {**

**public:**

**virtual void display() {**

**cout << "Base class" << endl;**

**}**

**};**

**class Derived : public Base {**

**public:**

**void display() {**

**cout << "Derived class" << endl;**

**}**

**};**

**int main() {**

**Base \*bptr;**

**Derived d;**

**bptr = &d;**

**bptr->display();**

**return 0;**

**}**

**---**

**Experiment 6 – File Handling**

**Part A: Writing to a File**

**#include<iostream>**

**#include<fstream>**

**using namespace std;**

**int main() {**

**ofstream file("data.txt");**

**file << "Hello File!";**

**file.close();**

**return 0;**

**}**

**Part B: Reading from a File**

**#include<iostream>**

**#include<fstream>**

**using namespace std;**

**int main() {**

**ifstream file("data.txt");**

**string str;**

**while(getline(file, str)) {**

**cout << str << endl;**

**}**

**file.close();**

**return 0;**

**}**

**---**

**Experiment 7 – Templates**

**Part A: Function Template**

**#include<iostream>**

**using namespace std;**

**template <class T>**

**T maximum(T a, T b) {**

**return (a > b) ? a : b;**

**}**

**int main() {**

**cout << "Max int: " << maximum(3, 7) << endl;**

**cout << "Max float: " << maximum(3.5f, 2.5f) << endl;**

**cout << "Max char: " << maximum('a', 'z') << endl;**

**return 0;**

**}**

**Part B: Class Template**

**#include<iostream>**

**using namespace std;**

**template <class T>**

**class Calc {**

**T a, b;**

**public:**

**Calc(T x, T y) {**

**a = x; b = y;**

**}**

**void add() {**

**cout << "Addition: " << a + b << endl;**

**}**

**};**

**int main() {**

**Calc<int> c1(3, 4);**

**c1.add();**

**Calc<float> c2(2.5, 3.5);**

**c2.add();**

**return 0;**

**}**