QUIZ-05; 16-01-2023; TIME:40MINUTES; MARKS:50

NAME:____ID:____

Q1: Write the output of following program

```
#include<iostream>
using namespace std;
class Vehicle
{
        public:
                virtual void display()=0;
};
class Car: public Vehicle
{
        public:
                void display()
                {
                        cout<<"In Car Class\n";
                }
};
class Bike :public Vehicle
{
        public:
                void display()
                {
                        cout<<"In Bike Class\n";
                }
};

int main()
{
        Vehicle *vptr;
        Car c;
        vptr=&c;
        vptr->display();
        Bike b;
        vptr=&b;
        vptr->display();

        return 0;
}
```

OUTPUT:

Q2: Will the following program compile?. Give reason.

```
#include<iostream>
using namespace std;
class Vehicle
{
        public:
                virtual void display()=0;
};
class Car: public Vehicle
{
        public:
                void car_display()
                {
                        cout<<"In Car Class\n";
                }
};
class Bike :public Vehicle
{
        public:
                void bike_display()
                {
                        cout<<"In Bike Class\n";
                }
};

int main()
{
        Vehicle *vptr;
        Car c;
        vptr=&c;
        vptr->display();
        Bike b;
        vptr=&b;
        vptr->display();

        return 0;
}
```

Output/Reason:

Singleton design pattern is a software design principle that is used to restrict the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. For example, if you are using a logger, that writes logs to a file, you can use a singleton class to create such a logger. You can create a singleton class using the following code.

```
#include <iostream>

using namespace std;

class Singleton {
  static Singleton *instance;
  int data;

  // Private constructor so that no objects can be created.
  Singleton() {
    data = 0;
  }

  public:
  static Singleton *getInstance() {
    if (!instance)
    instance = new Singleton;
    return instance;
  }

  int getData() {
    return this -> data;
  }

  void setData(int data) {
    this -> data = data;
  }
};

//Initialize pointer to zero so that it can be initialized in first call to getInstance
Singleton *Singleton::instance = 0;

int main(){
  Singleton *s = s->getInstance();
  cout << s->getData() << endl;
  s->setData(100);
  cout << s->getData() << endl;
  return 0;
}
```

Q3: Read and understand above description of Singleton Design Pattern. Design a class Employee having at least 5 attributes, getter and setters, CalculateSalary function. Make sure only one instance of the class is created.

The factory method is a creational design pattern, i.e., related to object creation. In the Factory pattern, we create objects without exposing the creation logic to the client and the client uses the same common interface to create a new type of object. The idea is to use a static member-function (static factory method) that creates & returns instances, hiding the details of class modules from the user. A factory pattern is one of the core design principles to create an object, allowing clients to create objects of a library (explained below) in a way such that it doesn't have a tight coupling with the class hierarchy of the library. See the following sample code.

```
#include<iostream>
using namespace std;
class Vehicle
{
	public:
		virtual void display()=0;
		virtual ~Vehicle()
		{
		}
		static Vehicle* FactoryMethod(int);
};
class Car: public Vehicle
{
	public:
		void display()
		{
			cout<<"Car\n";
		}
};
class Bike: public Vehicle
{
	public:
		void display()
		{
			cout<<"Bike\n";
		}
};
Vehicle* Vehicle::FactoryMethod(int choice)
{
	Vehicle *vptr;
	if (choice==1)
	{

		Car c;
		vptr=&c;
		vptr->display();
	}
	else if (choice==2)
	{
		Bike b;
```

```
                vptr=&b;
                vptr->display();

        }
        else
        {
                cout<<"Not a valid choice";
        }
}

int main()
{
        Vehicle::FactoryMethod(1); //prints car
        Vehicle::FactoryMethod(2); //prints bike
        return 0;
}
```

Q4: Implement the Employee Class of Q3 using Factory Design pattern.