Sample: Programming- Java

```
//1. main.cpp

#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include "Animals.h"
#include "Color.h"
#include "Fishes.h"
#include "Car.h"
#include "Prey.h"
using namespace std;

   //print out function for Vector and Algorithm containers
voidprintColor(Color a){
a.display();
}

int main(void)
{

       //using new and delete through pointers
       //Demonstrate dynamic binding non-virtual and virtual impact
//       Animals *P = new Fishes( "Fish",20, 2.3f, 13,"Shark");
//       P->display();
//       delete P;

       //demonstrate static state
```

```
//      Animals *a = new Fishes( "Fish",20, 2.3f, 13,"Shark");
//      a->display(); //the speed will be 20mph instead of 13mph because of static state

      //cast weight float to integer
      //type-safe downcast - cast base to derived
//      Animals *b = new Fishes( "Fish",20, 2.3f, 13,"Shark");
//      b->display();
//      Fishes *T = dynamic_cast<Fishes *>(b);
//      T->display();
//      delete b;
//      delete T;


      //Demonstrate dynamic binding virtual and non-virtual impact
//      Animals *c = new Fishes("Fish",20, 2.3f, 13,"Shark");
//      Animals *d = new Prey("Lion","Africa", 50.3f);
//      c->display();
//      d->display();
//      delete c;
//      delete d;

      //Fish class
      //demonstrate the abstract class
//      Fishes e("Fish", 20, 2.3f, 13,"Shark");
//      e.display();



      //Array pointers
//       Fishes f[2]{Fishes("Fish",20, 2.3f, 13,"Shark"),Fishes("Fish",20, 2.3f, 13,"whale")};
//       Fishes *L = &f[0];
```

```
//        L->display();
//        L->tempPrint();
//        (++L)->display();



        //Car class
        //demonstrate friend method
//        Car g(230,"Gray",16);
//        DublinReg(g);

        //Copy constructor of Animal class
        //Struct in C++
//        Prey h("Lion","Africa", 10.3f);
//        h.display();
//        Prey j(h);
//        j.display();



        //operator overloaded
//        Fishes k("Fish",15,4.5f,30,"Shark");
//        Fishes m("Fish", 15, 4.5f, 23,"Whale");
//        if(k==m){cout<< "a and b classes have same age and weight!!" <<endl;} //using operator ==
//        Fishes o = m;
//        o.display(); //using operator= which will assign c class to be same as b
//        Fishes q = k + m;   //using operator +
//        q.display();    //this will add age and weight from a and b

        //demonstrate passing by constant reference in Fish class
//        Fishes r("Fish",15,4.5f,30,"Shark");
//        Fishes s("Fish", 13, 4.5f, 23,"Whale");
//        r.AgeCalculator(s);
```

```
        //demonstrate vector and Algorithm containers
//        string fishName;
//        vector<string> V;  // declare a vector container of ints
//
//        cout<< "Please enter three Fish:" <<endl;
//        for(inti=0; i<3; i++)
//        {
//            cin>>fishName;
//            V.push_back(fishName);
//        }
//        sort(V.begin(), V.end());   // Algorithm
//        cout<<endl<< "Sorted Color in alphabet:" <<endl;
//        for_each(V.begin(), V.end(), printColor);
//

return 0;
}


//2. Animals.h

#include <string>
#ifndef ANIMALS_H
#define ANIMALS_H

class Animals{
   //using protected access specifier
protected:
   //constant age to demonstrate cast styles
int age;    //inherited in Fishes class
```

```
int speed;  //inherited in Car and Fishes classes
float weight;   //inherited in Fishes class

    //static state of class
staticintnextSpeed;

    //using public access specifier
public:
    // constructor
Animals(int, float, int);

    //multiple constructors for overloading method print
Animals(int,int);
Animals(float speed);

    //destructor
virtual ~Animals();

    //Abstract method
virtualintMaxTemp() = 0;

    //Overloading method
virtual void print(int,int);
virtual void print(float);

    //over-riding method
    //dynamic binding with non-virtual method
virtual void display();

    //Display max temp
virtual void tempPrint();
```

```
    //multiply weight
virtual void multiWeight();
};



#endif // ANIMALS_H



//3. Animals.cpp

#include "Animals.h"
#include <iostream>
#include <string>
#include <vector>
using namespace std;



//static state of class
int Animals::nextSpeed = 20;

// constructor

Animals::Animals(int age, float weight, int speed):age(age), weight(weight), speed(nextSpeed++){
    //cout<< "The animal has just been created !!"<<endl;
}

//multiple constructors for overloading method print
```

```
Animals::Animals(int age, int speed): age(age), speed(speed) {}

Animals::Animals(float weight){
this -> weight = weight;
}

//destructor
Animals::~Animals(){
    //cout<< "The animal has been destroyed !!"<<endl;
}

//overriding print method
void Animals::print(int speed, int age){
cout<< "Animal speed is " << speed << " mph and its estimated age is "<< age<< " week" <<endl;
}

void Animals::print(float weight){
cout<< "Animal estimated age is " << weight << " week"<<endl;
}

void Animals::display(){
    //Casting Animal weight from float to integer value
cout<< "Animal estimated age: "<<age <<  " week."<<endl;
cout<< "Animal estimated weight: "<<static_cast<int> (weight)<<" kg."<<endl;    //cast from float to integer
cout<<"Animal estimated speed: "<< speed<< " mph" <<endl;
cout<<"Animal tolerates max temperature of  "<<MaxTemp() << " degrees Celsius" <<endl;
}

//max temp display
void Animals::tempPrint(){
cout<<"Animal tolerates max temperature of  "<<MaxTemp() << " degrees Celsius" <<endl;
```

```
}


void Animals::multiWeight(){
weight = weight *2;
}


//4. Color.h

#include <string>
#ifndef COLOR_H
#define COLOR_H

//this class used just to be inherited
//Fish color will be inherited by Car class
class Color{
    //using protected access specifier
protected:
std::string color;

    //using public access specifier
public:
    //constructor
Color(std::string);

    //destructor
virtual ~Color();

    //Abstract method
    //virtual std::string type()=0;
```

```
virtual void display();
voidprintColor(Color);

};

#endif // COLOR_H


//5. Color.cpp

#include "Color.h"
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

//constructor
Color::Color(string color): color(color){
    //cout<< "The Color class has just been created !!"<<endl;
}

//destructor
Color::~Color(){
    //cout<< "The Color "<< color <<" class has just been created !!" <<endl;
}

//method display for test
void Color::display(){
cout<< " The color is " << color <<endl;
```

```
}


//6. Car.h

#ifndef CAR_H
#define CAR_H
#include "Animals.h"
#include "Color.h"
#include <string>

//using template


//Car class is a part of Color class and Animal class
//the class will inherit string color from Clor class and int speed from Animal class
class Car : public Animals, public Color{
    //using private access specifier
private:
intreg;

    //use of friend function of reg object in Car class
friend void DublinReg(Car &);

    //using public access specifier
public:
    //new constructor will inherit color and speed and the class itself includes registration number
Car(int, std::string, int);

    //template function
```

```
voidOverSpeed();

    //destructor
virtual ~Car();

    //Abstract method from abstract class Color
    //virtual std::string type(){return "Light";}

    //Abstract method from Animal class
virtualintMaxTemp(){return 135;}

    //method for test
void display();
};

#endif // CAR_H



//7. Car.cpp

#include "Car.h"
#include <iostream>
#include <string>
using namespace std;




//constructor
Car::Car(int speed, string color, intreg): Animals(speed), Color(color), reg(reg){
    //cout<< "Car class has just been created !!"<<endl;
}
```

```
//destructor to clean up the destroyed object
Car::~Car(){
    //cout<< "The car with "<<reg<<" registration number has just been destroyed!!" <<endl;
}

//display method inherits from Color and Animal classes
void Car::display(){
cout<< "Car speed: " << speed<<endl;
    Color::display();
cout<< "Registration number of the car: " <<reg<<endl;
    Animals::tempPrint();
}

//friend function
voidDublinReg(Car &R){
    R.reg = 12345;
cout<< "Dublin cars have special registration numbers which begin with " << R.reg <<endl;
}


//template function
void Car::OverSpeed(){
if(speed > 120){
cout<< "A fine is given to the car with " <<reg
<<" number because the speed is over 120. Car speed " << speed <<endl;
    }
else{
cout<< "No fine is given to the car with " <<reg
<<" number because the speed is not over 120. Car speed " << speed <<endl;
```

```
    }
}


//8. Fishes.h

#ifndef FISHES_H
#define FISHES_H
#include "Animals.h"
#include <string>


//inherits for Animals Class
class Fishes: public Animals{
    //using private access specifier
private:
std::string name;
std::string species;

public:
    //inherited constructor from Animal class
Fishes(std::string, int, float, int, std::string);

    //overloaded operators
    Fishes operator +(Fishes);
    Fishes operator =(Fishes);
bool operator == (Fishes);

    //destructor
virtual ~Fishes();
```

```
    //Abstract method from Animal class
virtualintMaxTemp(){return 50;}

    //overriding method
void display();

    //passing by constant reference object
virtual void AgeCalculator(const Fishes &);

    //Use of Vector and Algorithm to sort Fishes comparing to weight


};

#endif // FISHES_H


//9. Fishes.cpp

#include "Fishes.h"
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

//inherit constructor from Animal class
Fishes::Fishes(string name, int age, float weight, intspeed,string species):
name(name),Animals(age,weight,speed), species(species){
    //cout<< "The animal with name "<< name <<" has just been created !!" <<endl;
}
```

```
//overloaded operator +
Fishes Fishes::operator + (Fishes f){
return Fishes(name, age+f.age, weight+f.weight,speed, species);
}

//overloaded operator ==
Fishes Fishes::operator = (Fishes f){
return Fishes(name, f.age, f.weight, speed, species);
}

//overloaded operator ==
bool Fishes::operator == (Fishes f){
return ((f.age == age) && (f.weight == weight));
}

//destructor
Fishes::~Fishes(){
    //cout<< "The animal with name "<< name <<" has just been destroyed !!" <<endl;
}

//overriding method
void Fishes::display(){
cout<< "Animal name: "<<name<<endl;
   Animals::display();
cout<< "Animal species is: "<< species <<endl;
}

//passing by constant reference object
void Fishes::AgeCalculator(const Fishes &source){
```

```
intnewAge = (age + source.age);
cout<< "Calculator result: " <<newAge<<endl;
}


//9. Prey.h

#ifndef PREY_H
#define PREY_H
#include "Animals.h"
#include <string>

//This struct used to demonstrate difference between class and struct and modified copy constructor
struct Prey : public Animals{
    //Members are public by default
Prey(std::string, std::string, float);

    //modified copy constructor passing by reference
Prey(const Prey &);

    //Abstract method from Animal class
virtualintMaxTemp(){return 63;}

    //Display function
void display();
private:
std::string name;
std::string home;
};

#endif // PREY_H
```

//10. Prey.cpp

```
#include "Prey.h"
#include <iostream>
#include <string>
using namespace std;


Prey::Prey(string name, string home, float weight): name(name), home(home), Animals(weight){}

//modified copy constructor
Prey::Prey(const Prey &Source): Animals(weight){
cout<< "Copy constructor is now built !!" <<endl;
name = Source.name;
home = Source.home;
weight = Source.weight;
}

void Prey::display(){
cout<< "Animal name: " << name <<endl;
cout<< "Animal home land: " << home <<endl;
cout<< "Animal weight in KG: " << weight <<endl;
}
```

11.

```
//Modified copy constructor
Car::Car(int speed, string color, intreg): Animals(speed), Color(color), reg(reg){
```

```
        //The modified copy constructor
        Car::Car(const Car &Property): Animals(speed){
cout<< "Car class has just been created !!" <<endl;
   Color = Property.color;
reg =    Property.reg;
speed = Property.speed;
}

}
```

12.

```
//Casting

//Dynamic Casting

Animals *b = new Fishes( "Fish",20, 2.3f, 13,"Shark");
                        b->display();
                        Fishes *T = dynamic_cast<Fishes *>(b);
                        T->display();
                        delete b;
                        delete T;
```

cast weight float to integer

```
      //type-safe downcast - cast base to derived
      Animals *b = new Fishes( "Fish",20, 2.3f, 13,"Shark");
b->display();
      Fishes *T = dynamic_cast<Fishes *>(b);
      T->display();
```

```
delete b;
delete T;



//Static Cast
void Car::display(){
cout<< "Car speed: " <<static_cast<int> (speed)<<mph<<endl;    //casting from float values to integers
    Color::display();
cout<< "Registration number of the car: " <<reg<<endl;
   Animals::tempPrint();
}

//const_cast
protected:
   //constant age to demonstrate cast styles
int speed;
float weight;
            constint age = 10;

            int *b = const_cast<int*>(&age);



//Reinterprete Cast

protected:
   //constant age to demonstrate cast styles
float weight;
            constint age;
            int speed = 0x1233254;
            char *ptr = reinterpret_cast<char*> (speed);
```

```
        *ptr = 3;
        return 0;
```

13.

```
//Class Template

//template function
void Car::OverSpeed(){
if(speed > 120){
cout<< "A fine is given to the car with " <<reg
<<" number because the speed is over 120. Car speed " << speed <<endl;
   }
else{
cout<< "No fine is given to the car with " <<reg
<<" number because the speed is not over 120. Car speed " << speed <<endl;
   }
}
```

14.

```
//Dynamic Binding

Animals *c = new Fishes("Fish",20, 2.3f, 13,"Shark");
     Animals *d = new Prey("Lion","Africa", 50.3f);
c->display();
d->display();
delete c;
delete d;
```