Solution Code


```
/*  C++ Program of Non-Templated class derived from Templated base class
*/

#include <iostream>
#include <string>
#include<math.h>
using namespace std;
double M_PI = 3.14;
enum eColor { none = 0, red, white, blue, yellow, green, black };

class Color
{
public:
    Color(eColor color);
    void setColor(eColor color);
    eColor getColor() { return mColor; };
    std::string getStrColor();

protected:
    eColor mColor;
};

Color::Color(eColor _color)
{
   mColor = _color;
}

void Color::setColor(eColor _color)
{
    mColor = _color;
}
```

Solution Code

ç

```
std::string Color::getStrColor()
{
   switch(mColor)
   {
      case red:
         return "red";
      case white:
         return "white";
      case blue:
         return "blue";
      case yellow:
         return "yellow";
      case green:
         return "green";
      case black:
         return "black";
      case none:
      default:
         return "none";
   }
}



template <typename T>
class Circle : public Color
{
```

Solution Code


```
public:
    Circle(T centerX, T centerY, T radius, eColor color);
    Circle(T centerX, T centerY, T radius);
    Circle(T radius);
    Circle();

    T area();
    T circumference();
    T getX();
    T getY();
    T getRadius();

protected:
    T x;
    T y;
    T radius;
};

template <typename T>
Circle<T>::Circle(T _x, T _y, T _radius, eColor _color)
: Color(_color)
{
    x = _x;
    y = _y;
    radius = _radius;
}
```

Solution Code


```
template <typename T>
Circle<T>::Circle(T _x, T _y, T _radius)
: Color(none)
{
    x = _x;
    y = _y;
    radius = _radius;
}
template <typename T>
Circle<T>::Circle(T _radius)
: Color(none)
{
    x = static_cast<T>(0);
    y = static_cast<T>(0);
    radius = _radius;
}

template <typename T>
Circle<T>::Circle()
: Color(none)
{
    x = static_cast<T>(0);
    y = static_cast<T>(0);
    radius = static_cast<T>(1);
}
template <typename T>
T Circle<T>::area()
{
    return M_PI * radius * radius;
}
```

Solution Code


```
template <typename T>
T Circle<T>::circumference()
{
    return static_cast<T>(2) * M_PI * radius;
}
class Sphere : public Circle<float>
{
public:
    Sphere(float centerZ, float centerX, float centerY, float radius, eColor color);
    Sphere(float radius);
    Sphere();

    float surfaceArea();
    float volume();
    float getZ();
private:
    float z;
};
Sphere::Sphere(float _x, float _y, float _z, float _radius, eColor _color)
: Circle<float>::Circle (_x, _y, _radius, _color)
{
    this->z = _z;
}
Sphere::Sphere(float _x, float _y, float _z, float _radius, eColor _color)
: Circle<float>::Circle (_x, _y, _radius, _color)
{
    this->z = _z;
}

Sphere::Sphere(float _radius)
: Circle::Circle (_radius)
```

Solution Code


```
{
    // Defaults from Circle(_radius) constructor can also initialize x, y, z
    this->x = static_cast<float>(0);
    this->y = static_cast<float>(0);
    this->z = static_cast<float>(0);
    this->radius = _radius;
}

Sphere::Sphere()
{
    // Defaults from Circle() default constructor can also initialize values
    this->x = static_cast<float>(0);
    this->y = static_cast<float>(0);
    this->z = static_cast<float>(0);
    this->radius = static_cast<float>(1);
}

float Sphere::surfaceArea()
{
    return static_cast<float>(4) * M_PI * this->radius * this->radius;
}

float Sphere::volume()
{
    float three = 3;
    float four  = 4;
    return four * M_PI * this->radius * this->radius * this->radius / three;
}
```

Solution Code


```
int main(int argc, char* argv[])
{
    Sphere sphereA(0.0, 0.0, 0.0,10.0, blue);
    cout << "\nVolume of sphere A :: " << sphereA.volume() << endl;
    cout << "\nColor of sphere A :: "  << sphereA.getStrColor() << endl;
    return 0;
}
```