C++ ASSIGNMENT

ASSIGNMENT NUMBER- 04 YASH CHANDRA

SEC-E

62

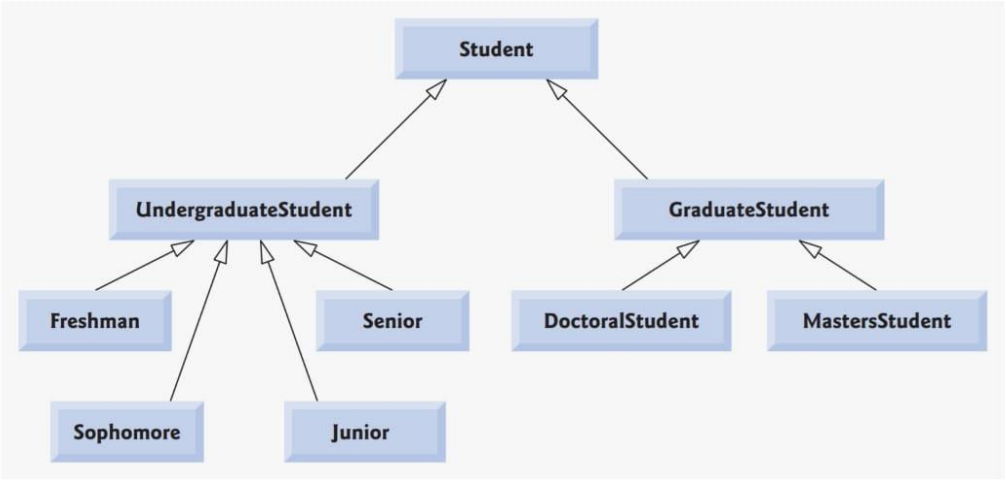
QUESTION-1 Discuss the ways using suitable examples in which inheritance promotes software reuse, saves time during program development and helps prevent errors

ANSWER-1 Inheritance allows developers to create subclasses that reuse code declared already in a superclass. Avoiding the duplication of common functionality between several classes by building a class inheritance hierarchy can save developers a considerable amount of time. Similarly, placing common functionality in a single superclass, rather than duplicating the code in multiple unrelated classes, helps prevent the same errors from appearing in multiple source-code files. If errors occur in the common functionality of the superclass, the software developer needs to modify only the superclass's.

QUESTION-2 Draw an inheritance hierarchy for students at a university. Use Student as the base class of the hierarchy, then include classes UndergraduateStudent and GraduateStudent that derive from Student . Continue to extend the hierarchy as deep (i.e., as many levels) as possible. For example, Freshman, Junior and Senior might derive from UndergraduateStudent , and DoctoralStudent and MastersStudent might derive from GraduateStudent . After drawing the hierarchy, discuss the relationships that exist between the classes. Also, implement the above hierarchy using C++ code.

ANSWER-2

ANS:



QUESTION -3

Draw an inheritance hierarchy for classes Quadrilateral, Trapezoid, Parallelogram, Rectangle and Square. Use Quadrilateral as the base class of the hierarchy. Make the hierarchy as deep as possible. Also, implement the above hierarchy using C++ code and create functions to calculate perimeter and area of the shape.

ANSWER-3

```
#include "QuadrilateralTest.h"

void QuadrilateralTest::main(std::vector<std::wstring> &args)
```

```
{
// NOTE: All coordinates are assumed to form the proper shapes
// A quadrilateral is a four-sided polygon
Quadrilateral *quadrilateral = new Quadrilateral(1.1, 1.2, 6.6, 2.8, 6.2, 9.9, 2.2, 7.4);
// A trapezoid is a quadrilateral having exactly two parallel sides
Trapezoid *trapezoid = new Trapezoid(0.0, 0.0, 10.0, 0.0, 8.0, 5.0, 3.3, 5.0);
// A parallelogram is a quadrilateral with opposite sides parallel
Parallelogram *parallelogram = new Parallelogram(5.0, 5.0, 11.0, 5.0, 12.0, 20.0, 6.0, 20.0);
// A rectangle is an equiangular parallelogram
Rectangle *rectangle = new Rectangle(17.0, 14.0, 30.0, 14.0, 30.0, 28.0, 17.0, 28.0);
// A square is an equiangular and equilateral parallelogram
Square *square = new Square(4.0, 0.0, 8.0, 0.0, 8.0, 4.0, 4.0, 4.0);
printf(L"%s %s %s %s %s\n", quadrilateral, trapezoid, parallelogram,rectangle, square);
} // end main


Point::Point(double xCoordinate, double yCoordinate)
```

```
{
x = xCoordinate; // set x
y = yCoordinate; // set y
} // end two-argument Point constructor

double Point::getX()
{
return x;
} // end method getX

double Point::getY()
{
return y;
} // end method getY
std::wstring Point::toString()
{
return std::wstring::format(L"( %.1f, %.1f )", getX(), getY());

}
Quadrilateral::Quadrilateral(double x1, double y1, double x2,
double y2, 13 double x3, double y3, double x4, double y4)
```

```
{
point1 = new Point(x1, y1);
point2 = new Point(x2, y2);
point3 = new Point(x3, y3);
point4 = new Point(x4, y4);

}
Point *Quadrilateral::getPoint1()
{
return point1;
}
Point *Quadrilateral::getPoint2()
{
return point2;
}

Point *Quadrilateral::getPoint3()
{
36 return point3;
37
}
```

```
38 39 40 Point *Quadrilateral::getPoint4()
{
42 return point4;
43
}

44 45 46 std::wstring Quadrilateral::toString()
{
return StringHelper::formatSimple(L"%s:\n%s", 49
L"Coordinates of Quadrilateral are", getCoordinatesAsString());
50
}

51 52 53 std::wstring Quadrilateral::getCoordinatesAsString()
{
55 return std::wstring::format(56 L"%s, %s, %s, %s\n", point1,
point2, point3, point4);
57
}
```

```
7 8 9 Trapezoid::Trapezoid(double x1, double y1, double x2,
double y2, 10 double x3, double y3, double x4, double y4)
{
12 Quadrilateral 5(x1, y1, x2, y2, x3, y3, x4, y4);
13
}

14 15 16 double Trapezoid::getHeight()
{
18 if (getPoint1().getY() == getPoint2().getY()) 19 return
std::abs(getPoint2().getY() - getPoint3().getY());
20 else 21 return std::abs(getPoint1().getY() -
getPoint2().getY());
22
}

23 24 25 double Trapezoid::getArea()
{
27 return getSumOfTwoSides() * getHeight() / 2.0;
28
}
```

```
29 30 31 double Trapezoid::getSumOfTwoSides()
{
if (getPoint1().getY() == getPoint2().getY()) return
std::abs(getPoint1().getX() - getPoint2().getX()) + 35
std::abs(getPoint3().getX() - getPoint4().getX());
else return std::abs(getPoint2().getX() - getPoint3().getX()) + 38
std::abs(getPoint4().getX() - getPoint1().getX());
}


std::wstring Trapezoid::toString()
{
return std::wstring::format(L"\n%s:\n%s%s: %s\n%s: %s\n", 45
L"Coordinates of Trapezoid are", getCoordinatesAsString(), 46
L"Height is", getHeight(), L"Area is", getArea());
}
Parallelogram::Parallelogram(double x1, double y1, double x2,
double y2, 8 double x3, double y3, double x4, double y4)
{
Trapezoid 5(x1, y1, x2, y2, x3, y3, x4, y4);
}
double Parallelogram::getWidth()
{
```

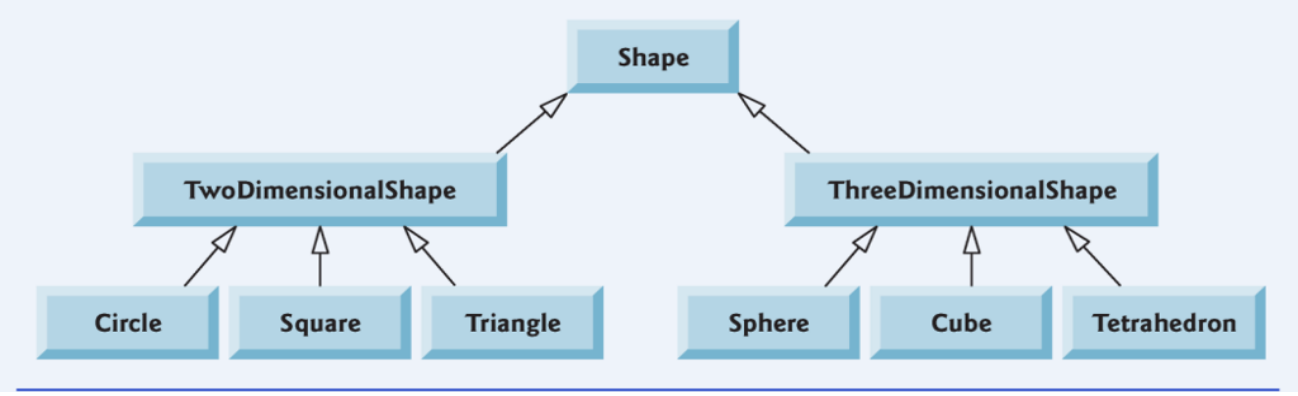
```
if (getPoint1().getY() == getPoint2().getY()) 17 return
std::abs(getPoint1().getX() - getPoint2().getX());

else 19 return std::abs(getPoint2().getX() - getPoint3().getX());

}

std::wstring Parallelogram::toString()

{

return std::wstring::format(L"\n%s:\n%s%s: %s\n%s: %s\n%s:
%s\n", L"Coordinates of Parallelogram are",
getCoordinatesAsString(), L"Width is", getWidth(), L"Height is",
getHeight(), L"Area is", getArea());

} // end method toString


Rectangle::Rectangle(double x1, double y1, double x2, double
y2, double x3, double y3, double x4, double y4) :
Parallelogram(x1, y1, x2, y2, x3, y3, x4, y4)

{

} // end eight-argument Rectangle constructor

Inheritance std::wstring Rectangle::toString()

{

return std::wstring::format(L"\n%s:\n%s%s: %s\n%s: %s\n%s:
%s\n", L"Coordinates of Rectangle are",
getCoordinatesAsString(), L"Width is", getWidth(), L"Height is",
getHeight(), L"Area is", getArea());

} // end method toString
```

```
Square::Square(double x1, double y1, double x2, double y2,
double x3, double y3, double x4, double y4) : Parallelogram(x1,
y1, x2, y2, x3, y3, x4, y4)

{

} // end eight-argument Square constructor// return string
representation of Square object


std::wstring Square::toString()

{

return std::wstring::format(L"\n%s:\n%s%s: %s\n%s: %s\n",
L"Coordinates of Square are", getCoordinatesAsString(), L"Side
is", getHeight(), L"Area is", getArea());

} // end method toString
```

QUESTION-4

Create a base Shape class which is inherited by TwoDShape and ThreeDShape and these shapes are further extended by their derived classes like square, rectangle, sphere etc. You should provide necessary methods like calculateArea, calculateVolume in these shapes

ANSWER -4



QUESTION-5 Create any base class as per your wish and extend that into other derived class. You should be able to understand the calling order or default constructors, parameterised constructors and destructors of all the classes. You should implement hybrid inheritance in this example.

ANSWER-5