**Lab 6**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 객체지향프로그래밍과자료구조 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.10.08.금 |

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* 다양한 도형에 대한 그림 그리기  \* Programmed by J. H. Kim  \* Last updated : 2021-10-08\*/  #include <iostream>  #include <string>  #include <fstream>  #include "Triangle.h"  #include "Circle.h"  #include "Rectang.h"  #include "Polygon.h"  #include "Star.h"  #include "AngledArc.h"  #include "Cylinder.h"  #include "Prism.h"  #include "Hexahedron.h"  using namespace std;  int main() {  // 여러가지 도형  Circle cir(100, 200, 80, 0, RGB\_BLACK, RGB\_RED, 3, "Circle");  Triangle tri(300, 200, 150, 130, 0, RGB\_BLACK, RGB\_YELLOW, 3, "Triangle");  Rectang rec(500, 200, 150, 150, 0, RGB\_BLACK, RGB\_BLUE, 4, "Rectangle");  PolyGon poly\_5(700, 200, 80, 5, 0, RGB\_BLACK, RGB\_GREEN, 4, "Polygon\_5");  PolyGon poly\_7(100, 400, 80, 7, 0, RGB\_BLACK, RGB\_MAGENTA, 4, "Polygon\_7");  Star star\_5(300, 400, 80, 5, 0, RGB\_BLACK, RGB\_GREEN, 4, "Star\_5");  AngledArc angle\_arc(500, 400, 80, 0, 45, 270, RGB\_RED, RGB\_BLUE, 4, "Angle\_Arc");  Cylinder cyl(700, 400, 80, 0, 100, RGB\_BLUE, RGB\_WHITE, 4, "Cylinder");  Prism psm(100, 600, 150, 130, 30 \* PI / 180, 100, RGB\_BLACK, RGB\_RED, 3, "Prism");  Hexahedron hxhd(300, 600, 150, 150, 70 \* PI / 180, 100, RGB\_BLACK, RGB\_YELLOW, 4, "Hexahedron");  ConsolePixelFrame frame(50, 50); // 시작 지점 설정  // 도형 추가  frame.addShape(&cir);  frame.addShape(&tri);  frame.addShape(&rec);  frame.addShape(&poly\_5);  frame.addShape(&poly\_7);  frame.addShape(&star\_5);  frame.addShape(&angle\_arc);  frame.addShape(&cyl);  frame.addShape(&psm);  frame.addShape(&hxhd);  // 추가된 도형 그리기  frame.drawShapes();  printf("hit any key to continue ....");  \_getch();  return 0;  } // end of main() |
| /\* Color.h \*/  #ifndef COLOR\_H  #define COLOR\_H  #include <Windows.h>  #include <string>  using namespace std;  // COLORREF is defined in <Windows.h>  // The COLORREF value is used to specify an RGB color,  // in hexadecimal form of 0x00bbggrr  const COLORREF RGB\_RED = 0x000000FF;  const COLORREF RGB\_GREEN = 0x0000FF00;  const COLORREF RGB\_BLUE = 0x00FF0000;  const COLORREF RGB\_BLACK = 0x00000000;  const COLORREF RGB\_ORANGE = 0x0000A5FF;  const COLORREF RGB\_YELLOW = 0x0000FFFF;  const COLORREF RGB\_MAGENTA = 0x00FF00FF;  const COLORREF RGB\_WHITE = 0x00FFFFFF;  ostream& fprintRGB(ostream&, COLORREF);  // RGB color code chart: https://www.rapidtables.com/web/color/RGB\_Color.html  /\* Note: RGB(red, green, blue) macro also provides COLORREF data  . RGB(FF, 00, 00) => 0x000000FF (RGB\_RED)  . RGB(00, FF, 00) => 0x0000FF00 (RGB\_GREEN)  . RGB(00, 00, FF) => 0x00FF0000 (RGB\_BLUE)  \*/  #endif |
| /\* Color.cpp \*/  #include <iostream>  #include <iomanip>  #include "Color.h"  ostream& fprintRGB(ostream& ostr, COLORREF color) {  int red, green, blue;  red = (color & 0x000000FF);  green = (color & 0x0000FF00) >> 8;  blue = (color & 0x00FF0000) >> 16;  ostr << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "  << setw(3) << blue << ")";  return ostr;  } |
| /\* Shape.h \*/  #ifndef S\_H  #define S\_H  #include <string>  #include <Windows.h>  #include <conio.h>  #include "ConsolePixelDrawing.h"  #include "Color.h"  using namespace std;  #define PI 3.141592  class ConsolePixelFrame;  class Shape { // 기반 클래스  friend ostream& operator<<(ostream&, Shape&); // 출력  public:  // 생성자  Shape(); // default constructor  Shape(string name);  Shape(int px, int py, double ang, COLORREF ln\_clr, COLORREF br\_clr,  int pen\_thick, string name);    // 가상함수  virtual ~Shape();  virtual void draw(ConsolePixelFrame cp\_frame);  void fprint(ostream&); // 출력  int get\_pos\_x() const { return pos\_x; } // x 좌표 반환  int get\_pos\_y() const { return pos\_y; } // y 좌표 반환  void set\_pos\_x(int x) { pos\_x = x; } // x 좌표 설정  void set\_pos\_y(int y) { pos\_y = y; } // y 좌표 설정  string getName() { return name; } // 이름 반환  void setName(string n) { name = n; } // 이름 설정  Shape& operator=(const Shape& s); // 대입연산자ㅛㅛㅛ  protected:  int pos\_x; // position x  int pos\_y; // position y  double angle;  string name;  int pen\_thickness;  COLORREF line\_color;  COLORREF brush\_color;  };  #endif |
| /\* Shape.cpp \*/  #include <iostream>  #include "Shape.h"  #include <iomanip>  using namespace std;  Shape::Shape() { // default constructor  pos\_x = pos\_y = 0;  angle = 0;  line\_color = brush\_color = RGB\_BLACK;  name = "no\_name";  //cout << "Shape::Default constructor (" << name << ").₩n";  }  Shape::Shape(string name)  :name(name) {  pos\_x = pos\_y = 0;  angle = 0;  pen\_thickness = 0;  line\_color = brush\_color = RGB\_BLACK;  //cout << "Shape::Constructor (" << name << ").₩n";  }  Shape::~Shape() {  //cout << "Shape::Destructor (" << name << ").₩n";  }  Shape::Shape(int px, int py, double ang, COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string nm) {  pos\_x = px;  pos\_y = py;  angle = ang;  pen\_thickness = pen\_thick;  line\_color = ln\_clr;  brush\_color = br\_clr;  name = nm;  //cout << "Shape::Constructor (" << name << ").₩n";  }  void Shape::fprint(ostream& fout) {  fout << name << ": pos(" << get\_pos\_x() << ", " << get\_pos\_y() << ")";  fout << ", angle(" << angle << ")";  fout << ", pen\_thickness(" << pen\_thickness << ")";  fout << ", line\_color(";  fprintRGB(fout, line\_color);  fout << "), brush\_color(";  fprintRGB(fout, brush\_color);  fout << ")";  }  void Shape::draw(ConsolePixelFrame cp\_frame) {  /\* virtual function that will be late-binded to sub-class's draw() \*/  }  Shape& Shape::operator=(const Shape& s) {  pos\_x = s.pos\_x;  pos\_y = s.pos\_y;  angle = s.angle;  name = s.name;  line\_color = s.line\_color;  brush\_color = s.brush\_color;  return \*this;  }  ostream& operator<<(ostream& ostr, Shape& shp) {  ostr << shp.name << ": pos(" << shp.get\_pos\_x() << ", " << shp.get\_pos\_y() << ")";  ostr << ", angle(" << shp.angle << ")";  ostr << ", pen\_thickness(" << shp.pen\_thickness << ")";  ostr << ", line\_color(";  fprintRGB(ostr, shp.line\_color);  ostr << "), brush\_color(";  fprintRGB(ostr, shp.brush\_color);  ostr << ")";  return ostr;  } |
| /\* Circle.h \*/  #ifndef Circle\_H  #define Circle\_H  #include <string>  #include "Shape.h"  using namespace std;  #define PI 3.141592  class Circle : public Shape { // 파생 클래스  friend ostream& operator<<(ostream&, Circle&); // 출력  public:  // 생성자  Circle();  Circle(string name);  Circle(int px, int py, int r, double ang, COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick,  string name);  //Circle(Circle &tr);  ~Circle(); // 소멸자  double getArea(); // 면적 반환  // 파생 클래스에 맞는 가상 함수 구현  virtual void draw(ConsolePixelFrame cp\_frame);    void fprint(ostream&); // 출력  int getRadius() const { return radius; } // Radius 반환  void setRadius(int r) { radius = r; } // Radius 설정  Circle& operator=(const Circle& right); // 대입연산자  protected:  int radius;  };  #endif |
| /\* Circle.cpp \*/  #include "Circle.h"  ostream& operator<<(ostream& fout, Circle& c) {  fout << c.name << ": pos(" << c.get\_pos\_x() << ", " << c.get\_pos\_y() << ")";  fout << ", angle(" << c.angle << ")";  fout << ", pen\_thickness(" << c.pen\_thickness << ")";  fout << ", line\_color(";  fprintRGB(fout, c.line\_color);  fout << "), brush\_color(";  fprintRGB(fout, c.brush\_color);  fout << ")";  return fout;  }  Circle::Circle()  : Shape() {  radius = 0;  }  Circle::Circle(string name)  : Shape(name) {  radius = 0;  }  Circle::Circle(int px, int py, int r, double ang, COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick,  string name)  :Shape(px, py, ang, ln\_clr, br\_clr, pen\_thick, name) {  radius = r;  }  //Circle(Circle &tr);  Circle::~Circle() {  //cout << "Circle::Destructor (" << name << ").₩n";  }  double Circle::getArea() {  return PI \* radius \* radius;  }  void Circle::draw(ConsolePixelFrame cp\_frame) {  HPEN new\_pen, old\_pen;  HBRUSH new\_brush, old\_brush;  HDC hdc;  int pos\_center\_x, pos\_center\_y;  pos\_center\_x = cp\_frame.get\_pos\_org\_x() + get\_pos\_x();  pos\_center\_y = cp\_frame.get\_pos\_org\_y() + get\_pos\_y();  hdc = cp\_frame.getConsole\_DC();  new\_pen = CreatePen(PS\_SOLID, pen\_thickness, line\_color);  old\_pen = (HPEN)SelectObject(hdc, new\_pen);  new\_brush = CreateSolidBrush(brush\_color);  old\_brush = (HBRUSH)SelectObject(hdc, new\_brush);  Ellipse(hdc, pos\_center\_x - radius, pos\_center\_y - radius, pos\_center\_x + radius,  pos\_center\_y + radius);  SelectObject(hdc, old\_pen);  DeleteObject(new\_pen);  SelectObject(hdc, old\_brush);  DeleteObject(new\_brush);  }  void Circle::fprint(ostream& fout) {  Shape::fprint(fout);  fout << ", radius(" << radius << ")";  }  Circle& Circle::operator=(const Circle& right) {  Shape::operator=(right);  radius = right.radius;  return \*this;  } |
| /\* Triangle.h \*/  #ifndef TRIANGLE\_H  #define TRIANGLE\_H  #include <string>  #include "Shape.h"  using namespace std;  class Triangle : public Shape { // 파생 클래스  //friend ostream& operator<<(ostream &, Triangle &);  public:  // 생성자  Triangle();  Triangle(string name);  Triangle(int px, int py, int b, int h, double ang, COLORREF ln\_clr, COLORREF br\_clr,  int pen\_thick, string name);  ~Triangle(); // 소멸자  double getArea(); // 표면적 반환  // 가상함수 구현  virtual void draw(ConsolePixelFrame cp\_frame);  void fprint(ostream&); // 출력  int getBase() { return base; } // 밑변 반환  int getHeight() { return tri\_height; } // 높이 반환  Triangle& operator=(const Triangle& tri); // 대입연산자  protected:  int base;  int tri\_height;  };  #endif |
| /\* Triangle.cpp \*/  #include "Triangle.h"  Triangle::Triangle()  : Shape() {  base = 0;  tri\_height = 0;  }  Triangle::Triangle(string name)  : Shape(name) {  base = 0;  tri\_height = 0;  }  Triangle::Triangle(int px, int py, int b, int h, double ang, COLORREF In\_clr, COLORREF br\_clr,  int pen\_thick, string name)  : Shape(px, py, ang, In\_clr, br\_clr, pen\_thick, name) {  base = b;  tri\_height = h;  }  Triangle::~Triangle() {  //cout << "Triangle::Destructor (" << name << ").₩n";  }  double Triangle::getArea() {  return (base / 2) \* tri\_height;  }  void Triangle::draw(ConsolePixelFrame cp\_frame) {  HDC hdc;  HPEN new\_pen, old\_pen;  HBRUSH new\_brush, old\_brush;  int pos\_center\_x, pos\_center\_y;  pos\_center\_x = cp\_frame.get\_pos\_org\_x() + get\_pos\_x();  pos\_center\_y = cp\_frame.get\_pos\_org\_y() + get\_pos\_y();  POINT p[3];  p[0].x = pos\_center\_x - base / 2;  p[0].y = pos\_center\_y + tri\_height \* 1.0 / 2.0;  p[1].x = pos\_center\_x + base / 2;  p[1].y = pos\_center\_y + tri\_height \* 1.0 / 2.0;  p[2].x = pos\_center\_x;  p[2].y = pos\_center\_y - tri\_height \* 1.0 / 2.0;  hdc = cp\_frame.getConsole\_DC();  new\_pen = CreatePen(PS\_SOLID, pen\_thickness, line\_color);  old\_pen = (HPEN)SelectObject(hdc, new\_pen);  new\_brush = CreateSolidBrush(brush\_color);  old\_brush = (HBRUSH)SelectObject(hdc, new\_brush);  Polygon(hdc, p, 3);  SelectObject(hdc, old\_pen);  DeleteObject(new\_pen);  SelectObject(hdc, old\_brush);  DeleteObject(new\_brush);  }  void Triangle::fprint(ostream& fout) {  Shape::fprint(fout);  fout << ", base(" << base << ")";  fout << ", tri\_height(" << tri\_height << ")";  }  Triangle& Triangle::operator=(const Triangle& tri) {  Shape::operator=(tri);  base = tri.base;  tri\_height = tri.tri\_height;  return \*this;  } |
| /\* Rectang.h \*/  #ifndef Rectang\_H  #define Rectang\_H  #include <string>  #include "Shape.h"  using namespace std;  class Rectang : public Shape { // 파생 클래스  //friend ostream& operator<<(ostream &, Rectangle &);  public:  // 생성자  Rectang();  Rectang(string name);  Rectang(int px, int py, int w, int l, double ang,  COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name);    ~Rectang(); // 소멸자  double getArea(); // 표면적 반환  // 가상함수 구현  virtual void draw(ConsolePixelFrame cp\_frame);  void fprint(ostream&); // 출력  int getWidth() { return width; } // 가로 반환  int getLength() { return length; } // 세로 반환  Rectang& operator=(Rectang& rec); // 대입연산자  protected:  int width;  int length;  };  #endif |
| /\* Rectang.cpp \*/  #include "Rectang.h"  Rectang::Rectang()  : Shape() {  width = 0;  length = 0;  }  Rectang::Rectang(string name)  : Shape(name) {  width = 0;  length = 0;  }  Rectang::Rectang(int px, int py, int w, int l, double ang,  COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name)  : Shape(px, py, ang, ln\_clr, br\_clr, pen\_thick, name) {  width = w;  length = l;  }  Rectang::~Rectang() {  //cout << "Rectang::Destructor (" << name << ").₩n";  }  double Rectang::getArea() {  return width \* length;  }  void Rectang::draw(ConsolePixelFrame cp\_frame) {  HDC hdc;  HPEN new\_pen, old\_pen;  HBRUSH new\_brush, old\_brush;  int pos\_center\_x, pos\_center\_y;  pos\_center\_x = cp\_frame.get\_pos\_org\_x() + get\_pos\_x();  pos\_center\_y = cp\_frame.get\_pos\_org\_y() + get\_pos\_y();  POINT p[4];  p[0].x = pos\_center\_x - width / 2;  p[0].y = pos\_center\_y - length / 2;  p[1].x = pos\_center\_x + width / 2;  p[1].y = pos\_center\_y - length / 2.0;  p[2].x = pos\_center\_x + width / 2;  p[2].y = pos\_center\_y + length / 2.0;  p[3].x = pos\_center\_x - width / 2;  p[3].y = pos\_center\_y + length / 2.0;  hdc = cp\_frame.getConsole\_DC();  new\_pen = CreatePen(PS\_SOLID, pen\_thickness, line\_color);  old\_pen = (HPEN)SelectObject(hdc, new\_pen);  new\_brush = CreateSolidBrush(brush\_color);  old\_brush = (HBRUSH)SelectObject(cp\_frame.getConsole\_DC(), new\_brush);  Polygon(hdc, p, 4);  SelectObject(hdc, old\_pen);  DeleteObject(new\_pen);  SelectObject(hdc, old\_brush);  DeleteObject(new\_brush);  }  void Rectang::fprint(ostream& fout) {  Shape::fprint(fout);  fout << ", width(" << width << ")";  fout << ", length(" << length << ")";  }  Rectang& Rectang::operator=(Rectang& rec) {  Shape::operator=(rec);  width = rec.width;  length = rec.length;  return \*this;  } |
| /\* Polygon.h \*/  #ifndef PolyGon\_H  #define PolyGon\_H  #include <string>  #include "Shape.h"  using namespace std;  class PolyGon : public Shape { // 파생 클래스  //friend ostream& operator<<(ostream &, PolyGonle &);  public:  // 생성자  PolyGon();  PolyGon(string name);  PolyGon(int px, int py, int radius, int num\_poly, double ang, COLORREF ln\_clr, COLORREF br\_clr,  int pen\_thick, string name);  ~PolyGon(); // 소멸자  // 가상함수 구현  virtual void draw(ConsolePixelFrame cp\_frame);  void fprint(ostream&); // 출력  int getRadius() { return radius; } // 반지름 반환  int getNumPoly() { return num\_poly; } // 꼭지점 반환  PolyGon& operator=(PolyGon& pg); // 대입연산자  protected:  int radius;  int num\_poly;  };  #endif |
| /\* Polygon.cpp \*/  #include "Polygon.h"  PolyGon::PolyGon()  : Shape() {  radius = 0;  num\_poly = 0;  }  PolyGon::PolyGon(string name)  : Shape(name) {  radius = 0;  num\_poly = 0;  }  PolyGon::PolyGon(int px, int py, int radius, int num\_poly, double ang,  COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name)  :Shape(px, py, ang, ln\_clr, br\_clr, pen\_thick, name) {  this->radius = radius;  this->num\_poly = num\_poly;  }  PolyGon::~PolyGon() {  //cout << "PolyGon::Destructor (" << name << ").₩n";  }  void PolyGon::draw(ConsolePixelFrame cp\_frame) {  POINT\* points = (POINT\*)calloc(num\_poly, sizeof(POINT));  double rad\_angle, delta\_angle; // angle in radian  int pos\_center\_x, pos\_center\_y;  pos\_center\_x = cp\_frame.get\_pos\_org\_x() + get\_pos\_x();  pos\_center\_y = cp\_frame.get\_pos\_org\_y() + get\_pos\_y();  int x, y;    HDC hdc;  HPEN new\_pen, old\_pen;  HBRUSH new\_brush, old\_brush;  hdc = cp\_frame.getConsole\_DC();  delta\_angle = 2.0 \* PI / num\_poly;  rad\_angle = PI / 2.0;  for (int i = 0; i < num\_poly; i++, rad\_angle += delta\_angle) {  x = pos\_center\_x + radius \* cos(rad\_angle);  y = pos\_center\_y - radius \* sin(rad\_angle);  points[i].x = x;  points[i].y = y;  }  new\_pen = CreatePen(PS\_SOLID, pen\_thickness, line\_color);  old\_pen = (HPEN)SelectObject(hdc, new\_pen);  new\_brush = CreateSolidBrush(brush\_color);  old\_brush = (HBRUSH)SelectObject(cp\_frame.getConsole\_DC(), new\_brush);  Polygon(hdc, points, num\_poly);    SelectObject(hdc, old\_pen);  DeleteObject(new\_pen);  SelectObject(hdc, old\_brush);  DeleteObject(new\_brush);  }  void PolyGon::fprint(ostream& fout) {  Shape::fprint(fout);  fout << ", radius(" << radius << ")";  fout << ", num\_poly(" << num\_poly << ")";  }  PolyGon& PolyGon::operator=(PolyGon& pg) {  Shape::operator=(pg);  radius = pg.radius;  num\_poly = pg.num\_poly;  return \*this;  } |
| /\* Star.h \*/  #ifndef Star\_H  #define Star\_H  #include <string>  #include "Shape.h"  using namespace std;  class Star : public Shape { // 파생 클래스  //friend ostream& operator<<(ostream &, PolyGonle &);  public:  // 생성자  Star();  Star(string name);  Star(int px, int py, int radius, int num\_vertices, double ang, COLORREF ln\_clr,  COLORREF br\_clr, int pen\_thick, string name);  //PolyGonle(PolyGonle &pg);  ~Star(); // 소멸자  //double getArea();  // 가상함수 구현  virtual void draw(ConsolePixelFrame cp\_frame);  virtual void draw(); // // used for testing of late binding  void fprint(ostream&); // 출력  int getRadius() { return radius; } // 반지름 반환  int getNumPoly() { return num\_vertices; } // 꼭짓점 반환  Star& operator=(Star& pg); // 대입연산자  protected:  int radius;  int num\_vertices;  };  #endif |
| /\* Star.cpp \*/  #include "Star.h"  Star::Star()  : Shape() {  radius = 0;  num\_vertices = 0;  }  Star::Star(string name)  : Shape(name) {  radius = 0;  num\_vertices = 0;  }  Star::Star(int px, int py, int radius, int num\_vertices, double ang, COLORREF ln\_clr,  COLORREF br\_clr, int pen\_thick, string name)  : Shape(px, py, ang, ln\_clr, br\_clr, pen\_thick, name) {  this->radius = radius;  this->num\_vertices = num\_vertices;  }  //PolyGonle(PolyGonle &pg);  Star::~Star() {  //cout << "Star::Destructor (" << name << ").₩n";  }  //double getArea();  void Star::draw(ConsolePixelFrame cp\_frame) {  POINT\* points = (POINT\*)malloc(5.0 \* sizeof(POINT));  double rad\_angle, delta\_angle;  // angle in radian  int pos\_center\_x, pos\_center\_y;  pos\_center\_x = cp\_frame.get\_pos\_org\_x() + get\_pos\_x();  pos\_center\_y = cp\_frame.get\_pos\_org\_y() + get\_pos\_y();  int x, y;  HDC hdc;  HPEN new\_pen, old\_pen;  HBRUSH new\_brush, old\_brush;  hdc = cp\_frame.getConsole\_DC();  delta\_angle = 2.0 \* PI / 5.0;  rad\_angle = PI / 2.0;  for (int i = 0; i < 5.0; i++, rad\_angle += delta\_angle) {  x = pos\_center\_x + radius \* cos(rad\_angle);  y = pos\_center\_y - radius \* sin(rad\_angle);  points[i].x = x;  points[i].y = y;  }  new\_pen = CreatePen(PS\_SOLID, pen\_thickness, line\_color);  old\_pen = (HPEN)SelectObject(hdc, new\_pen);  new\_brush = CreateSolidBrush(brush\_color);  old\_brush = (HBRUSH)SelectObject(hdc, new\_brush);  //instead of Polygon(hdc, points, num\_poly);  MoveToEx(hdc, points[0].x, points[0].y, NULL);  LineTo(hdc, points[2].x, points[2].y);  LineTo(hdc, points[4].x, points[4].y);  LineTo(hdc, points[1].x, points[1].y);  LineTo(hdc, points[3].x, points[3].y);  LineTo(hdc, points[0].x, points[0].y);  SelectObject(hdc, old\_pen);  DeleteObject(new\_pen);  SelectObject(hdc, old\_brush);  DeleteObject(new\_brush);  }  void Star::draw() {  cout << "draw() of Star";  fprint(cout);  }  void Star::fprint(ostream& fout) {  Shape::fprint(fout);  fout << ", radius(" << radius << ")";  fout << ", num\_vertices(" << num\_vertices << ")";  }  Star& Star::operator=(Star& pg) {  Shape::operator=(pg);  radius = pg.radius;  num\_vertices = pg.num\_vertices;  return \*this;  } |
| /\* AngledArc.h \*/  #ifndef ANGLE\_ARC\_H  #define ANGLE\_ARC\_H  #include <string>  #include "Shape.h"  using namespace std;  class AngledArc : public Shape { // 파생 클래스  friend ostream& operator<<(ostream&, const AngledArc&); // 출력  public:  // 생성자  AngledArc();  AngledArc(string name);  AngledArc(int px, int py, int r, int ang, int start\_ang, int sweep\_ang,  COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name);  //AngledArc(AngledArc &angarc);  ~AngledArc();// 소멸자    // 가상함수  virtual void draw(ConsolePixelFrame cp\_frame);  virtual void draw(); // // used for testing of late binding  void fprint(ostream&); // 출력  int getRadius() const { return radius; } // Radius 반환  void setRadius(int r) { radius = r; } // Radius 설정  AngledArc& operator=(const AngledArc& right); // 대입연산자  protected:  int radius;  int start\_angle;  int sweep\_angle;  };  #endif |
| /\* AngledArc.cpp \*/  #include <iostream>  #include <math.h>  #include "AngledArc.h"  #include "ConsolePixelDrawing.h"  #include <iomanip>  using namespace std;  AngledArc::AngledArc() // default constructor  :Shape("no\_name"), radius(0), start\_angle(0), sweep\_angle(0) {  //cout << "AngleArc::Default AngleArc constructor (" << name << ").₩n";  }  AngledArc::AngledArc(string name)  : Shape(name), radius(0), start\_angle(0), sweep\_angle(0) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  }  AngledArc::AngledArc(int px, int py, int r, int ang, int start\_ang, int sweep\_ang,  COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name)  : Shape(px, py, ang, ln\_clr, br\_clr, pen\_thick, name) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  radius = r;  start\_angle = start\_ang;  sweep\_angle = sweep\_ang;  }  AngledArc::~AngledArc() {  //cout << "AngleArc::Destructor (" << name << ").₩n";  }  void AngledArc::draw() {  cout << "draw() of AngleArc";  fprint(cout);  }  void AngledArc::draw(ConsolePixelFrame cp\_frame) {  HPEN new\_pen, old\_pen;  HBRUSH new\_brush, old\_brush;  HDC hdc;  int fr\_px, fr\_py;  int start\_px, start\_py;  double start\_ang\_rad;  fr\_px = cp\_frame.get\_pos\_org\_x() + get\_pos\_x();  fr\_py = cp\_frame.get\_pos\_org\_y() + get\_pos\_y();  hdc = cp\_frame.getConsole\_DC();  new\_pen = CreatePen(PS\_SOLID, pen\_thickness, line\_color);  old\_pen = (HPEN)SelectObject(hdc, new\_pen);  new\_brush = CreateSolidBrush(brush\_color);  old\_brush = (HBRUSH)SelectObject(hdc, new\_brush);  start\_ang\_rad = start\_angle \* 3.141592 / 180.0;  start\_px = fr\_px + radius \* cos(start\_ang\_rad);  start\_py = fr\_py - radius \* sin(start\_ang\_rad);  MoveToEx(hdc, fr\_px, fr\_py, (LPPOINT)NULL);  LineTo(hdc, start\_px, start\_py);  AngleArc(hdc, fr\_px, fr\_py, radius, start\_angle, sweep\_angle);  LineTo(hdc, fr\_px, fr\_py);  SelectObject(hdc, old\_pen);  DeleteObject(new\_pen);  SelectObject(hdc, old\_brush);  DeleteObject(new\_brush);  }  ostream& operator<<(ostream& fout, const AngledArc& ang\_arc) {  fout << ang\_arc.name << ": pos(" << ang\_arc.get\_pos\_x() << ", " << ang\_arc.get\_pos\_y() << ")";  fout << ", angle(" << ang\_arc.angle << ")";  fout << ", pen\_thickness(" << ang\_arc.pen\_thickness << ")";  fout << ", line\_color("; fprintRGB(fout, ang\_arc.line\_color);  fout << "), brush\_color("; fprintRGB(fout, ang\_arc.brush\_color);  fout << ")";  fout << ", radius(" << ang\_arc.radius << ")";  fout << ", start\_ang (" << ang\_arc.start\_angle << "), sweep\_ang(" << ang\_arc.sweep\_angle << ")";  fout << endl;  return fout;  }  AngledArc& AngledArc::operator=(const AngledArc& right) {  Shape::operator=(right);  radius = right.radius;  return \*this;  }  void AngledArc::fprint(ostream& fout) {  Shape::fprint(fout);  fout << ", radius (" << radius << "), start\_angle (" << start\_angle;  fout << "), sweep\_angle (" << sweep\_angle << ")";  fout << endl;  } |
| /\* Cylinder.h \*/  #ifndef CYLINDER\_H  #define CYLINDER\_H  #include <string>  #include "Shape.h"  using namespace std;  class Cylinder : public Shape { // 파생 클래스  friend ostream& operator<<(ostream&, const Cylinder&); // 출력  public:  // 생성자  Cylinder();  Cylinder(string name);  Cylinder(int px, int py, int r, int ang, int height,  COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name);    //Cylinder(Cylinder &cyl);  ~Cylinder(); // 소멸자  double getArea(); // 넓이  // 가상함수 구현  virtual void draw(ConsolePixelFrame cp\_frame);  virtual void draw(); // // used for testing of late binding  void fprint(ostream&); // 출력  int getRadius() const { return radius; } // 반지름 반환  void setRadius(int r) { radius = r; } // 반지름 설정  Cylinder& operator=(const Cylinder& right); // 대입연산자  protected:  int radius;  int height;  };  #endif |
| /\* Cylinder.cpp \*/  #include <iostream>  #include <math.h>  #include "Cylinder.h"  #include <iomanip>  using namespace std;  Cylinder::Cylinder() // default constructor  : Shape("no\_name"), radius(0), height(0) {  //cout << "Cylinder::Default AngleArc constructor (" << name << ").₩n";  }  Cylinder::Cylinder(string name)  : Shape(name), radius(0), height(0) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  }  Cylinder::Cylinder(int px, int py, int r, int ang, int ht, COLORREF ln\_clr, COLORREF br\_clr,  int pen\_thick, string name)  : Shape(px, py, ang, ln\_clr, br\_clr, pen\_thick, name) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  radius = r;  height = ht;  }  Cylinder::~Cylinder() {  //cout << "Cylinder::Destructor (" << name << ").₩n";  }  double Cylinder::getArea() {  return 2 \* PI \* radius \* radius + 2 \* PI \* radius \* height;  }  void Cylinder::draw() {  cout << "draw() of Cylinder";  fprint(cout);  }  void Cylinder::draw(ConsolePixelFrame cp\_frame) {  HPEN new\_pen, old\_pen;  HBRUSH new\_brush, old\_brush;  HDC hdc;  int center\_px, center\_py;  int start\_px, start\_py;  double start\_ang\_rad;    center\_px = cp\_frame.get\_pos\_org\_x() + get\_pos\_x();  center\_py = cp\_frame.get\_pos\_org\_y() + get\_pos\_y();    hdc = cp\_frame.getConsole\_DC();  new\_pen = CreatePen(PS\_SOLID, pen\_thickness, line\_color);  old\_pen = (HPEN)SelectObject(hdc, new\_pen);  new\_brush = CreateSolidBrush(brush\_color);  old\_brush = (HBRUSH)SelectObject(hdc, new\_brush);    MoveToEx(hdc, center\_px - radius, center\_py - height / 2, (LPPOINT)NULL);  LineTo(hdc, center\_px - radius, center\_py + height / 2); // 왼쪽 줄  MoveToEx(hdc, center\_px + radius, center\_py - height / 2, (LPPOINT)NULL);  LineTo(hdc, center\_px + radius, center\_py + height / 2); // 오른쪽 줄  Ellipse(hdc, center\_px - radius, center\_py - height / 2 - 20, // 윗면  center\_px + radius, center\_py - height / 2 + 20);  MoveToEx(hdc, center\_px - radius, center\_py + height / 2, (LPPOINT)NULL);  ArcTo(hdc, center\_px - radius, center\_py + height / 2 - 20, center\_px + radius, // 아랫면 곡선  center\_py + height / 2 + 20, center\_px - radius, center\_py + height / 2,  center\_px + radius, center\_py + height / 2);    SelectObject(hdc, old\_pen); DeleteObject(new\_pen);  SelectObject(hdc, old\_brush); DeleteObject(new\_brush);  }  ostream& operator<<(ostream& fout, const Cylinder& cyl) {  fout << cyl.name << ": pos(" << cyl.get\_pos\_x() << ", " << cyl.get\_pos\_y() << ")";  fout << ", angle(" << cyl.angle << ")";  fout << ", pen\_thickness(" << cyl.pen\_thickness << ")";  fout << ", line\_color(";  fprintRGB(fout, cyl.line\_color);  fout << "), brush\_color(";  fprintRGB(fout, cyl.brush\_color);  fout << ")";  fout << ", radius(" << cyl.radius << ")";  fout << ", height (" << cyl.height << ")";  fout << endl;  return fout;  }  Cylinder& Cylinder::operator=(const Cylinder& right) {  Shape::operator=(right);  radius = right.radius;  height = right.height;  return \*this;  }  void Cylinder::fprint(ostream& fout) {  Shape::fprint(fout);  fout << ", radius (" << radius << "), height (" << height << ")";  fout << endl;  } |
| /\* Prism.h \*/  #ifndef PSM\_H  #define PSM\_H  #include <string>  #include "Shape.h"  using namespace std;  class Prism : public Shape { // 파생 클래스  friend ostream& operator<<(ostream&, const Prism&); // 출력  public:  // 생성자  Prism();  Prism(string name);  Prism(int px, int py, int b, int th, double ang, int ht, COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name);    //Cylinder(Cylinder &cyl);  ~Prism(); // 소멸자  double getArea(); // 표면적 반환  // 가상함수 구현  virtual void draw(ConsolePixelFrame cp\_frame);  virtual void draw(); // // used for testing of late binding  void fprint(ostream&); // 출력  int getBase() const { return base; } // 밑변 반환  int getTriHeight() { return tri\_height; } // 밑면 높이 반환  Prism& operator=(const Prism& right); // 대입연산자  protected:  int base;  int tri\_height;  int height;  };  #endif // !PSM\_H |
| /\* Prism.cpp \*/  #include <iostream>  #include <math.h>  #include "Prism.h"  #include <iomanip>  using namespace std;  Prism::Prism() // default constructor  : Shape("no\_name"), base(0), tri\_height(0), height(0) {  //cout << "Cylinder::Default AngleArc constructor (" << name << ").₩n";  }  Prism::Prism(string name)  : Shape(name), base(0), tri\_height(0), height(0) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  }  Prism::Prism(int px, int py, int b, int th, double ang, int ht,  COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name)  : Shape(px, py, ang, ln\_clr, br\_clr, pen\_thick, name) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  base = b;  tri\_height = th;  height = ht;  }  Prism::~Prism() {  //cout << "Cylinder::Destructor (" << name << ").₩n";  }  double Prism::getArea() {  return base \* tri\_height + base \* height +  2 \* sqrt(base \* base / 4.0 + tri\_height \* tri\_height) \* height;  }  void Prism::draw() {  cout << "draw() of Cylinder";  fprint(cout);  }  void Prism::draw(ConsolePixelFrame cp\_frame){  HPEN new\_pen, old\_pen; HBRUSH new\_brush, old\_brush; HDC hdc;  int center\_px, center\_py; int start\_px, start\_py; double start\_ang\_rad;  center\_px = cp\_frame.get\_pos\_org\_x() + get\_pos\_x();  center\_py = cp\_frame.get\_pos\_org\_y() + get\_pos\_y();  POINT p[3];  p[0].x = center\_px - base / 2;  p[0].y = center\_py - height / 2;  p[1].x = p[0].x + base;  p[1].y = p[0].y;  p[2].x = p[0].x + base / 2.0;  p[2].y = p[0].y - (base / 2.0) \* tan(angle);  hdc = cp\_frame.getConsole\_DC();  new\_pen = CreatePen(PS\_SOLID, pen\_thickness, line\_color);  old\_pen = (HPEN)SelectObject(hdc, new\_pen);  new\_brush = CreateSolidBrush(brush\_color);  old\_brush = (HBRUSH)SelectObject(hdc, new\_brush);  Polygon(hdc, p, 3);  MoveToEx(hdc, p[0].x, p[0].y, (LPPOINT)NULL);  LineTo(hdc, p[0].x, p[0].y + height);  MoveToEx(hdc, p[1].x, p[1].y, (LPPOINT)NULL);  LineTo(hdc, p[1].x, p[1].y + height);    MoveToEx(hdc, center\_px - base / 2, center\_py + height / 2, (LPPOINT)NULL);  LineTo(hdc, center\_px + base / 2, center\_py + height / 2);  SelectObject(hdc, old\_pen);  DeleteObject(new\_pen);  SelectObject(hdc, old\_brush);  DeleteObject(new\_brush);  }  ostream& operator<<(ostream& fout, const Prism& prm) {  fout << prm.name << ": pos(" << prm.get\_pos\_x() << ", " << prm.get\_pos\_y() << ")";  fout << ", angle(" << prm.angle << ")";  fout << ", pen\_thickness(" << prm.pen\_thickness << ")";  fout << ", line\_color(";  fprintRGB(fout, prm.line\_color);  fout << "), brush\_color(";  fprintRGB(fout, prm.brush\_color);  fout << ")";  fout << ", base(" << prm.base << ")";  fout << ", tri\_height(" << prm.tri\_height << ")";  fout << ", height (" << prm.height << ")";  fout << endl;  return fout;  }  Prism& Prism::operator=(const Prism& right) {  Shape::operator=(right);  base = right.base;  tri\_height = right.tri\_height;  height = right.height;  return \*this;  }  void Prism::fprint(ostream& fout) {  Shape::fprint(fout);  fout << ", base (" << base << "), tri\_height (" << tri\_height << ")";  fout << ", height(" << height << ")";  fout << endl;  } |
| /\* Hexahedron.h \*/  #ifndef HXHD\_H  #define HXHD\_H  #include <string>  #include "Shape.h"  using namespace std;  class Hexahedron : public Shape { // 파생 클래스  friend ostream& operator<<(ostream&, const Hexahedron&); // 출력  public:  // 생성자  Hexahedron();  Hexahedron(string name);  Hexahedron(int px, int py, int w, int l, double ang, int ht, COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name);    //Cylinder(Cylinder &cyl);  ~Hexahedron(); // 소멸자  double getArea(); // 표면적  // 가상함수 구현  virtual void draw(ConsolePixelFrame cp\_frame);  virtual void draw(); // // used for testing of late binding  void fprint(ostream&); // 출력  int getWidth() const { return width; } // 가로 반환  int getLength() { return length; } // 세로 반환  Hexahedron& operator=(const Hexahedron& right); // 대입연산자  protected:  int width;  int length;  int height;  };  #endif // !HXHD\_H |
| /\* Hexahedron.cpp \*/  #include <iostream>  #include <math.h>  #include "Hexahedron.h"  #include <iomanip>  using namespace std;  Hexahedron::Hexahedron() // default constructor  : Shape("no\_name"), width(0), length(0), height(0) {  //cout << "Cylinder::Default AngleArc constructor (" << name << ").₩n";  }  Hexahedron::Hexahedron(string name)  : Shape(name), width(0), length(0), height(0) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  }  Hexahedron::Hexahedron(int px, int py, int w, int l, double ang, int ht,  COLORREF ln\_clr, COLORREF br\_clr, int pen\_thick, string name)  : Shape(px, py, ang, ln\_clr, br\_clr, pen\_thick, name) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  width = w;  length = l;  height = ht;  }  Hexahedron::~Hexahedron() {  //cout << "Cylinder::Destructor (" << name << ").₩n";  }  double Hexahedron::getArea() {  return 2 \* (width \* length + width \* height + length \* height);  }  void Hexahedron::draw() {  cout << "draw() of Cylinder";  fprint(cout);  }  void Hexahedron::draw(ConsolePixelFrame cp\_frame) {  HPEN new\_pen, old\_pen; HBRUSH new\_brush, old\_brush; HDC hdc;  int center\_px, center\_py; int start\_px, start\_py; double start\_ang\_rad;  double slp = length \* cos(angle);  center\_px = cp\_frame.get\_pos\_org\_x() + get\_pos\_x();  center\_py = cp\_frame.get\_pos\_org\_y() + get\_pos\_y();  POINT p[4];  p[2].x = center\_px + width / 2;  p[2].y = center\_py - height / 2;  p[3].x = center\_px - width / 2;  p[3].y = center\_py - height / 2;  p[0].x = p[3].x + slp \* cos(angle);  p[0].y = p[3].y - slp \* sin(angle);  p[1].x = p[2].x + slp \* cos(angle);  p[1].y = p[2].y - slp \* sin(angle);  hdc = cp\_frame.getConsole\_DC();  new\_pen = CreatePen(PS\_SOLID, pen\_thickness, line\_color);  old\_pen = (HPEN)SelectObject(hdc, new\_pen);  new\_brush = CreateSolidBrush(brush\_color);  old\_brush = (HBRUSH)SelectObject(hdc, new\_brush);  MoveToEx(hdc, p[3].x, p[3].y, (LPPOINT)NULL);  LineTo(hdc, p[3].x, p[3].y + height);  MoveToEx(hdc, p[2].x, p[2].y, (LPPOINT)NULL);  LineTo(hdc, p[2].x, p[2].y + height);  MoveToEx(hdc, p[1].x, p[1].y, (LPPOINT)NULL);  LineTo(hdc, p[1].x, p[1].y + height);  MoveToEx(hdc, p[3].x, p[3].y + height, (LPPOINT)NULL);  LineTo(hdc, p[2].x, p[2].y + height);  MoveToEx(hdc, p[2].x, p[2].y + height, (LPPOINT)NULL);  LineTo(hdc, p[2].x + slp \* cos(angle), p[2].y - slp \* sin(angle) + height);  Polygon(hdc, p, 4); // 윗면  SelectObject(hdc, old\_pen);  DeleteObject(new\_pen);  SelectObject(hdc, old\_brush);  DeleteObject(new\_brush);  }  ostream& operator<<(ostream& fout, const Hexahedron& hxhd) {  fout << hxhd.name << ": pos(" << hxhd.get\_pos\_x() << ", " << hxhd.get\_pos\_y() << ")";  fout << ", angle(" << hxhd.angle << ")";  fout << ", pen\_thickness(" << hxhd.pen\_thickness << ")";  fout << ", line\_color(";  fprintRGB(fout, hxhd.line\_color);  fout << "), brush\_color(";  fprintRGB(fout, hxhd.brush\_color);  fout << ")";  fout << ", width(" << hxhd.width << ")";  fout << ", length(" << hxhd.length << ")";  fout << ", height (" << hxhd.height << ")";  fout << endl;  return fout;  }  Hexahedron& Hexahedron::operator=(const Hexahedron& right) {  Shape::operator=(right);  width = right.width;  length = right.length;  height = right.height;  return \*this;  }  void Hexahedron::fprint(ostream& fout) {  Shape::fprint(fout);  fout << ", width (" << width << "), length (" << length << ")";  fout << ", height(" << height << ")";  fout << endl;  } |
| /\* ConsolePixelDrawing.h \*/  #ifndef PIXEL\_DRAWING\_H  #define PIXEL\_DRAWING\_H  #include <iostream>  #include <string>  #include <Windows.h>  #include <conio.h>  #include "Shape.h"  #include "Color.h"  using namespace std;  /\* PEN\_Styles \*/  #define PS\_SOLID 0  #define PS\_DASH 1 // -------  #define PS\_DOT 2 // .......  #define PS\_DASHDOT 3 // \_.\_.\_.\_  #define PS\_DASHDOTDOT 4 // \_..\_..\_  #define PS\_NULL 5  #define PS\_INSIDEFRAME 6  #define PS\_USERSTYLE 7  #define PS\_ALTERNATE 8  #define MAX\_NUM\_SHAPES 100  class Shape;  class ConsolePixelFrame {  public:  ConsolePixelFrame(int org\_x, int org\_y); // 생성자  ~ConsolePixelFrame(); // 소멸자  // 기반 클래스의 포인터 배열에 파생 클래스의 포인터를 추가  void addShape(Shape\* new\_shape);    // 저장된 포인터의 클래스에 맞는 가상함수를 실행  void drawShapes();  int get\_pos\_org\_x() { return pos\_org\_x; }  int get\_pos\_org\_y() { return pos\_org\_y; }  HDC getConsole\_DC() { return console\_DC; }  private:  HWND console;  HDC console\_DC; // device context  Shape\*\* pShapes; // Array of Shape Pointers  int num\_shapes;  int capacity;  int pos\_org\_x;  int pos\_org\_y;  bool isValidIndex(int index);  };  #endif // !PIXEL\_DRAWING\_H |
| /\* ConsolePixelDrawing.cpp \*/  #include "ConsolePixelDrawing.h"  ConsolePixelFrame::ConsolePixelFrame(int px, int py) {  console = GetConsoleWindow();  console\_DC = GetDC(console);  pShapes = new Shape \* [MAX\_NUM\_SHAPES];  num\_shapes = 0;  capacity = MAX\_NUM\_SHAPES;  pos\_org\_x = px;  pos\_org\_y = py;  }  ConsolePixelFrame::~ConsolePixelFrame() {  //delete[] shapes;  //ReleaseDC(console, console\_DC);  }  void ConsolePixelFrame::addShape(Shape\* pNew\_shape) {  if (num\_shapes >= capacity) {  cout << "ConsolePixelFrame::addShape ==> Expanding capacity to "  << capacity \* 2 << "shapes " << endl;  Shape\*\* old\_shapes = pShapes;  pShapes = new Shape \* [capacity \* 2];  if (pShapes == NULL) {  cout << "Error in expanding dynamic array for shapes capacity "  << capacity \* 2 << "shapes " << endl;  exit;  }  for (int i = 0; i < num\_shapes; i++)  pShapes[i] = old\_shapes[i];  capacity = capacity \* 2;  delete[] old\_shapes;  }  pShapes[num\_shapes] = pNew\_shape;  num\_shapes++;  }  void ConsolePixelFrame::drawShapes() {  cout << "Drawing " << num\_shapes << " shapes :" << endl;  if (num\_shapes > 0)  for (int i = 0; i < num\_shapes; i++)  pShapes[i]->draw(\*this);  }  bool ConsolePixelFrame::isValidIndex(int index) {  if ((index < 0) || (index >= num\_shapes)) {  cout << "Error in ConsolePixelFrame::isValidIndex : current number of shapes ("  << num\_shapes << "), index : " << index << ") !!" << endl;  return false;  }  else  return true;  } |
|  |

**2. 2021-2 객체지향형 프로그래밍과 자료구조 실습 Oral Test**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 학번 | 21812158 | 성명 | 김주환 | 점수 |  |

|  |
| --- |
| **6.1 다형성 (polymorphism)이 무엇이며, 왜 필요한가에 대하여 예를 들어 설명하라.**   * 다형성 : 동일한 기반 클래스로부터 파생 클래스 객체들을 하나의 체계로 관리 가능한 성질이다. 동일한 이름을 가지는 하나의 함수가 적용되는 객체에 따라 다른 기능을 가지게 한다. * 구현 가상함수로 구현된다. 실제 실행 기능이 결정되기 이전에 해당 함수의 사용을 프로그램에 구성할 수 있다. 프로그램 소스에 포함된 가상함수가 실제 어떤 함수로 실행될 것인지는 실행단계에서 결정된다. -> late binding * 필요성 동일한 기반 클래스로부터 상속받아 생성된 파생 클래스의 객체들을 기반 클래스의 동일한 멤버함수를 사용하여 하나의 체계로 쉽게 관리할 수 있다. * 예 같은 부모를 둔 서로 다른 자식 클래스의 객체들을 up-casting을 통해 부모 클래스 객체 배열에 대입하고, 이를 가상함수를 통해 하나의 체계로 관리할 수 있다. 이 때, 함수가 어떤 기능을 할지는 실행 단계에서 결정된다. |
| **6.2 다형성 (polymorphism)을 구현하기 위하여 사용되는 가상함수 (vritual function)과 지연 바인딩 (late binding)이 무엇이며, 어떻게 동작하는지에 대하여 예를 들어 설명하라.**   * 가상함수 (virtual function) 실제 실행되는 기능이 결정되기 이전에 해당 함수의 기능을 프로그램에 구현할 수 있다. 기반 클래스의 객체에 저장된 포인터에 해당하는 파생 클래스의 가상함수를 실행한다. * 지연 바인딩 (late binding) 프로그램 소스에 포함된 가상함수가 실제 어떤 함수로 실행될 것인지는 실행단계에서 결정된다. * 구현 다양한 도형의 공통적인 속성을 가지는 기반 클래스(class Shape)를 설계하고 이를 상속하는 파생 클래스를 설계한다. 다양한 도형을 하나의 체계로 관리할 수 있도록 가상함수(draw())를 구현하며, 실제 파생 클래스의 도형을 그릴 때에는 해당 파생 클래스의 draw() 함수가 되도록 late binding 해준다.  기반 클래스(class Shape)로부터 상속받아 생성되는 파생 클래스는 가상함수(draw())를 사용하며, 각 도형에 따라 다른 기능을 가진다  컴파일러에게 실제 실행될 함수가 어떻게 구현되어 있는지 아직 모르며, 실제 프로그램이 실행될 때까지 기다리도록 하며, 실제 프로그램이 실행될 때 결정되도록 프로그램을 구성한다. * 동작 1) 기반 클래스의 포인터에 파생 클래스의 포인터를 upcasting한다.   2) 기반 클래스의 포인터에서 가상 함수를 호출한다.   3) 기반 클래스의 포인터에 저장된 파생 클래스에 해당하는 가상함수가 late binding된다. |
| **6.3 가상함수와 late binding 기능을 사용하여 화면에 class Shape으로부터 상속받은 다수의 도형들을 class Shape의 포인터로 drawing하는 방법에 대하여 상세하게 설명하라.**  베이스 클래스에서 공통적으로 사용되는 기능을 구현할 때, 조작에 해당하는 동작이 실제 베이스 클래스의 포인터가 누구를 가리키는지 따져서 실제 도형에 해당하는 기능이 실행될 수 있도록 한다.  다양한 도형을 하나의 체계로 관리할 수 있도록 draw()를 구현하며, 실제 파생 클래스의 도형을 그릴 때에는 해당 파생클래스의 draw() 함수가 되도록 late binding 해준다.  컴파일러에게 실제 실행될 함수가 어떻게 구현되어 있는지 아직 모르며, 실제 프로그램이 실행될 때까지 기다리도록 하며, 실제 프로그램이 실행될 때 결정되도록 프로그램을 구성한다.  이를 Late binding 또는 dynamic binding이라고 한다.   * 다양한 도형의 공통적인 속성을 가지는 기반 클래스와 이를 상속하는 파생 클래스 설계 다양한 도형 : 직사각형, 원, 삼각형, …… 각 도형은 서로 다른 속성(멤버)를 가진다. 각 도형의 클래스는 기반 클래스로부터 상속받아야한다. * 가상함수 구현 기반 클래스에는 순수 가상함수를 선언하여 상속받은 클래스의 멤버함수를 연결할 수 있는 기능만 구현한다. 각 파생 클래스에는 각 도형을 하나의 체계로 관리할 수 있도록 draw()를 구현하며, 실제 파생 클래스의 도형을 그릴 때에는 해당 파생 클래스의 draw() 함수가 되도록 late binding한다. * 다양한 도형을 하나의 체계로 관리 여러 개의 도형에 대한 각 클래스 객체 포인터를 기반 클래스의 포인터 배열에 upcasting한다. upcasting이 실행된 이후로 해당 포인터에 가상 함수를 연결할 수 있다. 반복문(체계)을 통해 배열에 저장된 다양한 도형을 그릴 수 있다. 실제 파생 클래스의 도형을 그릴 때에는 해당 파생 클래스의 draw() 함수를 late binding해준다. |
| **6.4 Upcasting slicing이 어떤 문제이며, 왜 발생하는가에 대하여 예를 들어 설명하라.**   * upcasting : 파생 클래스의 객체를 기반 클래스의 객체에 대입하는 것이다. 반대의 경우(downcasting은 성립하지 않는다. / 정의문제) * problem upcasting을 할 경우, 파생 클래스의 멤버 중 기반 클래스의 멤버에 속하는 범위만 전달된다. 즉, 범위 외의 정보는 생략된다. * solution 멤버에 바로 접근하지 않고, 가상함수를 통해서 멤버를 불러온다. 기반 클래스의 포인터를 사용하여 파생 클래스의 가상 함수를 호출한다. 한번 upcasting한 후라면 가상함수로 연결된다. |