LAB10: Polymorphism

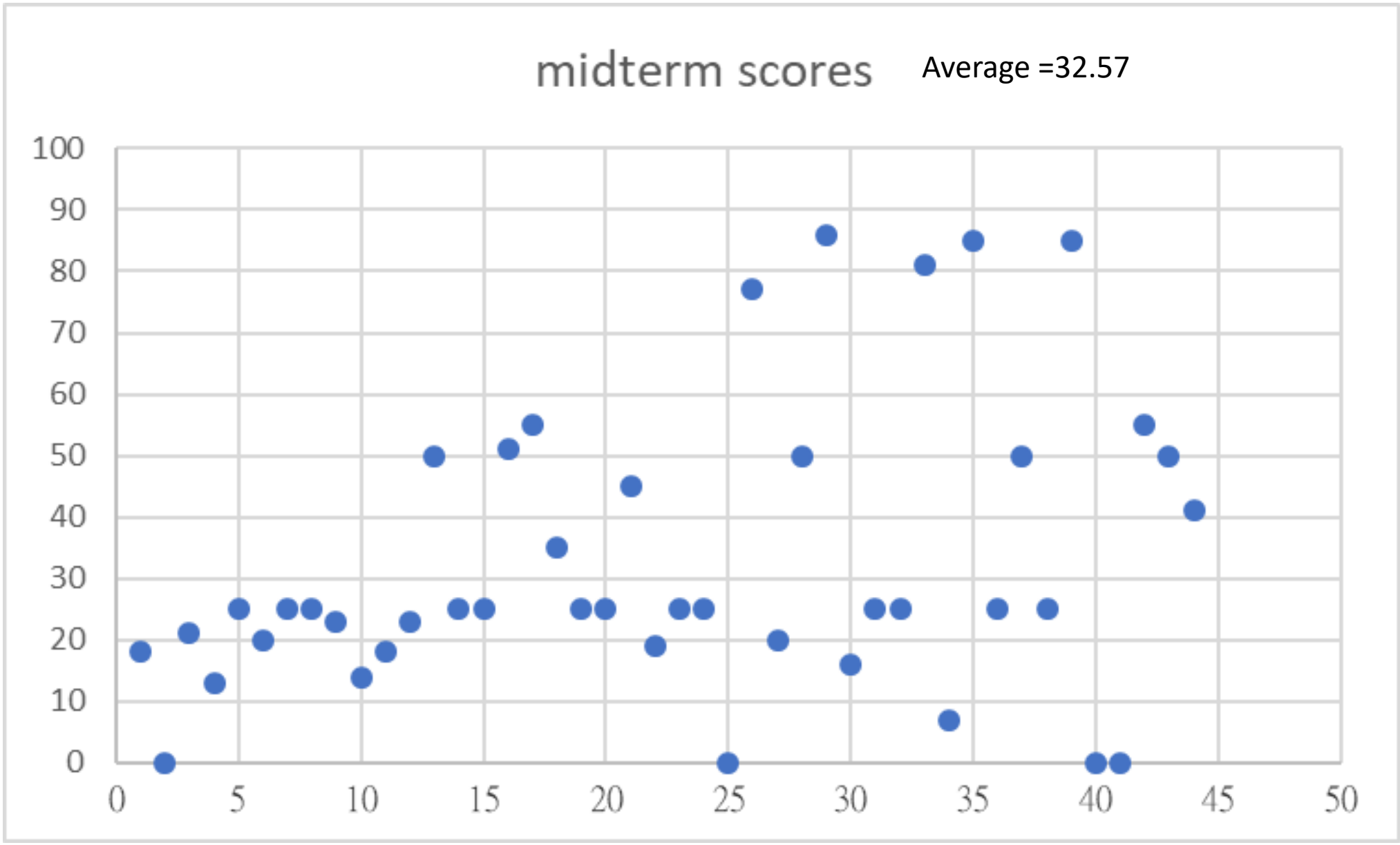
Shapes with Polymorphism

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

04/28/2022

Midterm Scores



Object Access in Inheritance Hierarchies without Polymorphism

- **In the hierarchy of inheritance, an object of derived class is an object of its base class. So we can assign an object of derived class to an object of base class.**

```
class Animal
{
public:
    void move();
}
class Frog: public Animal
{
Public:
    void move();
}
```

```
class Fish: public Animal
{
public:
    void move();
}
class Bird: public Animal
{
Public:
    void move();
}
```

Access without Polymorphism

- **To invoke a member function of an object without polymorphism, the member function called depends on the type of handles defined, not depending on the type of objects the handles are pointing to.**
  - Three types of handles: pointer, reference, name

```
Animal anAnimal;  // name handle
Frog aFrog;           // name handle
Frog *aFrogPtr = &aFrog;  // pointer handle
Frog &aliasFrog=*aFrogPtr;   // reference handle
anAnimal.move();  // name handle, call member function(MB) move()
of Animal
aFrog.move(); // name handle, call MB move() of Frog
aFrogPtr->move();  // pointer handle, call MB move() of Frog
aliasFrog.move();  // reference handle, call MB move() of Frog
anAnimal = aFrog; //OK
anAnimal.move(); // call MB move of Animal
Frog anotherFrog = anAnimal; // NOT Ok
```

Class Definition for Polymorphism

- **Polymorphism enables us to program in the general rather than in the specific.**
  - **Taking the animal's move as an example, the action of move of one animal differs from the action of move of another animal. Then we can define a virtual "move()" member function in base class generally and give specific implementation of move() in each derived class**

```
class Animal
{
public:
    virtual void move();
}
class Frog: public Animal
{
Public:
    virtual void move();
}
```

```
class Fish: public Animal
{
public:
    virtual void move();
}
class bird: public Animal
{
Public:
    virtual void move();
}
```

Access with Polymorphism

- **To call the "move()" member function of a particular animal, we can also use one of two types of base class handles: reference and pointer. However, the member function called is the one in the object pointed by the handle, ie., depending on the object pointed by the handle.**

```
Animal anAnimal;  // name handle
Frog aFrog;           // name handle
Frog *aFrogPtr = &aFrog;  // pointer handle
anAnimal.move();  // name handle, call member function(MB) move() of Animal
aFrog.move(); // name handle, call MB move() of Frog
aFrogPtr->move();  // pointer handle, call MB move() of Frog
anAnimal = aFrog; //OK
anAnimal.move(); // call MB move of Animal
Animal &isAnimal=*aFrogPtr;   // reference handle, *aFrogPtr can be replaced by aFrog
isAnimal.move();  // reference handle, call MB move() of Frog
Bird aBird;
Animal *anotherAnimal = &aBird; // pointer handle
anotherAnimal->move(); // call MB move() of Bird
```

Abstract Class and Pure Virtual Function

- **What is Animal move()?**
  - **No specific actions of move for animal, however, a move for a frog, a bird, or a fish can be defined precisely. So the move() in Animal class can have no implementation. Only the move() of derived classes such as Frog, Fish, and Bird has an implementation, i.e., having a function body.**
- **The classes in a hierarchy used to describe this sort of concept are called abstract classes.**
- **Abstract classes are used as base classes for deriving more concrete classes.**
- **Attempting to instantiate an object of an abstract class causes compilation error.**
  - **What we can do is to create pointers of abstract class objects.**

Forming Pure Virtual Function

- **virtual void move() = 0 in Animal Class definition indicating that move() is a pure virtual function.**

```
class Animal
{
public:
    virtual void move() = 0;
}
// No function body (implementation) of move() for Animal class.

class Frog: public Animal
{
Public:
    virtual void move();
}
```

```
class Fish: public Animal
{
public:
    virtual void move();
}
class bird: public Animal
{
Public:
    virtual void move();
}
```

Virtual Function & Pure Virtual Function

- **Virtual function can have or have not an implementation. The derived class can have no implementation of a virtual function if its base class has one.**
- **Pure virtual function should not have an implementation in base class. The member function of a derived class should provide its own implementation.**

Downcasting

- **Down casting enables a program to determine the type of object at execution time and act on that object accordingly.**
  - **For example, what if we would like to do something extra when a fish moves?**

```
vector <Animal *> animalArray(3);
Frog aFrog;
Bird aBird;
Fish aFish;
animalArray[0] = &aFrog;
animalArray[1] = &aBird;
animalArray[2] = &aFish;
for(int i=0; i<3; i++)
animalArray[i].move();
```

```
vector <Animal *> anArray(3);
Frog aFrog;
Bird aBird;
Fish aFish;
anArray[0] = &aFrog;
anArray[1] = &aBird;
anArray[2] = &aFish;
for(int i=0; i<3; i++){
anArray[i].move();
Fish *fishPtr = dynamic_cast<Fish *> (anArray[i]);
If( fishPtr != 0 ) { …..}
}
```


Do when anArray[i] points to a Fish object.

Lab10: Shapes with Polymorphism

- **Based on the base class Shape below, design three shape classes Triangle, Rectangle, and Circle.**

```
class Shape
{
public:
   Shape() { };
   virtual double area()  const = 0;
   virtual bool inside(const Pt &) const = 0 ;
   virtual double perimeter() const = 0;
   virtual bool degenerate() const = 0;
   virtual void print() const =0;
};
```

area() calculates the area of a shape.
Inside(const Pt& pt) checks whether a point pt is inside a shape.
perimeter() calculates the perimeter of a shape.
degenerate() checks whether a shape is degenerated. For example, a circle becomes a point; a rectangle becomes a line; a triangle becomes a line, etc.
print() should print shape type, coordinates of a shape, radius for a circle, area, and perimeter of a shape. Then, check whether a shape is degenerated. If yes, print "This shape is degenerated."

main() Function

```
int main()
{
    Pt p1(1, 2); // a point defined by X and Y coordinates
    Pt p2(5, 8);
    Pt p3(-2, 10);
    Pt p4(0,0);
    Pt p5(1.5, 4);
    Pt p6(2, 8);
    Pt p7(10, 8);
    Triangle tr1(p1, p2, p3);  // defined by giving three points
    Triangle tr2(p2, p3, p4);
    Triangle tr3(p2, p6, p7);
    Rectangle rect1(p1, p3); // defined by giving two points
    Rectangle rect2(p1, p1);
    Circle cir1(p4, 10.0);  // defined by a center point and  radius
    Circle cir2(p2, 4);
    Rectangle rect3(p1, p2);
    const int numShape = 8;
    vector < Shape*> baseShape(numShape);
    baseShape[0] = &tr1;
    baseShape[1] = &tr2;
    baseShape[2] = &rect1;
    baseShape[3] = &cir1;
    baseShape[4] = &tr3;
    baseShape[5] = &rect2;
    baseShape[6] = &cir2;
    baseShape[7] = &rect3;
```

```
    for(int i=0; i<numShape; i++){
        baseShape[i]->print();
        p5.print();
        if(baseShape[i]->inside(p5))
            cout << " is inside this shape." << endl;
        else
            cout << " is outside this shape." << endl;
        ** Adding a statement here to complete the main()
        if(cPtr != 0)
            cout << "## This shape is a circle. " << endl;
        ** Adding a statement here to complete the main()
        if(tPtr != 0)
            cout << "## This shape is a triangle. " << endl;
        ** Adding a statement here to complete the main()
        if(rPtr != 0)
            cout << "## This shape is a rectangle. " << endl;
    }
    return 0;
}
```

A line marked with ** in the beginning of the line should be replaced by a statement so that the desired output can be generated.

Pt is a class for creating point object.

Output

```
Traingle: (1, 2) (5, 8) (-2, 10)
Area: 25
Perimeter: 23.0352
(1.5, 4)  is inside this shape.
## This shape is a triangle.

Traingle: (5, 8) (-2, 10) (0, 0)
Area: 33
Perimeter: 26.9121
(1.5, 4)  is inside this shape.
## This shape is a triangle.

Rectangle: (1, 2) (-2, 10)
Area: 24
Perimeter: 22
(1.5, 4)  is outside this shape.
## This shape is a rectangle.

Circle: 10 (0, 0)
Area: 314.16
Perimeter: 62.8319
(1.5, 4)  is inside this shape.
## This shape is a circle.
```

```
Traingle: (5, 8) (2, 8) (10, 8)
Area: 0
Perimeter: 8
This shape is degenerated.
(1.5, 4)  is outside this shape.
## This shape is a triangle.

Rectangle: (1, 2) (1, 2)
Area: 0
Perimeter: 0
This shape is degenerated.
(1.5, 4)  is outside this shape.
## This shape is a rectangle.

Circle: 4 (5, 8)
Area: 50.2655
Perimeter: 25.1328
(1.5, 4)  is outside this shape.
## This shape is a circle.

Rectangle: (1, 2) (5, 8)
Area: 24
Perimeter: 20
(1.5, 4)  is inside this shape.
## This shape is a rectangle.
```

Pay attention to this number

Key Points for TA Grading

- **The base class Shape must not be modified.**
- **All the three classes Triangle, Rectangle, and Circle should be implemented**
- **The output should be the same as that given in the lab.**
- **main() should not be modified except the three lines marked with a **. Each line marked with a ** should be replaced by a statement.**