CSC 133

Object-Oriented Computer Graphics Programming

OOP Concepts SP

Dr. Kin Chung Kwan

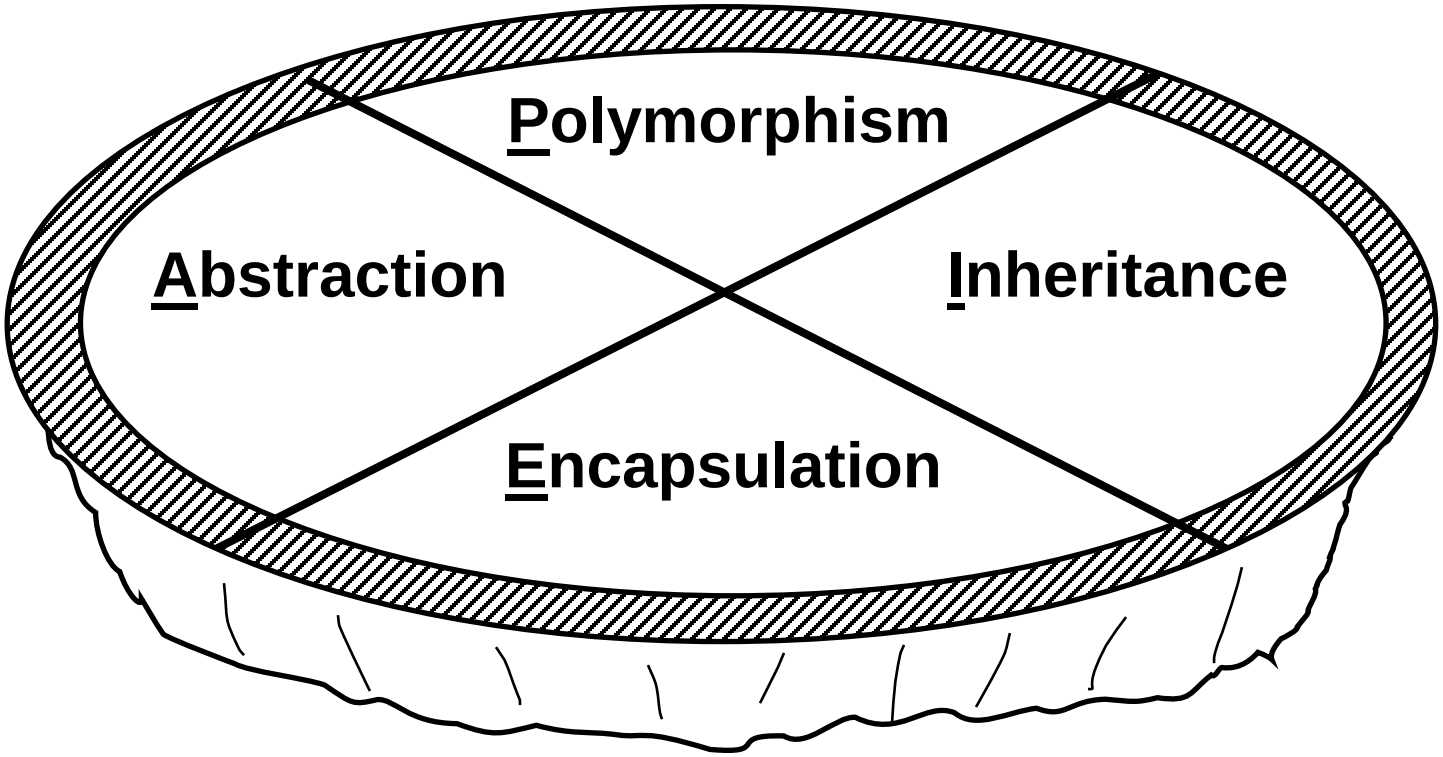
Spring 2023

Computer Science Department
California State University, Sacramento

SACRAMENTO STATE

“A Pie”

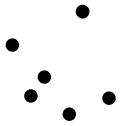
Four distinct OOP Concepts (or Pillars)



Last Lecture

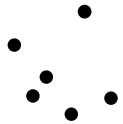
We ate the last part.

Time to dance.



Wait!

Some little things remain



Four Concepts

- **Abstraction**
- **Polymorphism**
- **Inheritance**
- **Encapsulation**

A vs E

Abstraction

- Implementation detail is not needed
- Usually, require encapsulation
- E.g., put into invisible box

Encapsulation

- Pack into boxes
- Abstraction is not necessary
- E.g., pack in a transparent bag

Encapsulation

Problem

Students asked:

"If the function is too simple, is it still necessary to use get/set for data?"

Let's see the following

Point Class

`Point` (without "Accessors"):

```
public class Point {
   public double x, y ;
   public Point () {
      x = 0.0 ;
      y = 0.0 ;
   }
}
```

BAD

`Point` (with "Accessors"):

```
public class Point {
   private double x, y ;
   public Point (){
      x = 0.0 ;
      y = 0.0 ;
   }
   public double getX() {
      return x ;
   }
   public double getY() {
      return y ;
   }
   public void setX (double newX) {
      x = newX ;
   }
   public void setY (double newY) {
      y = newY ;
   }
}
```

GOOD

Let's See What Happen

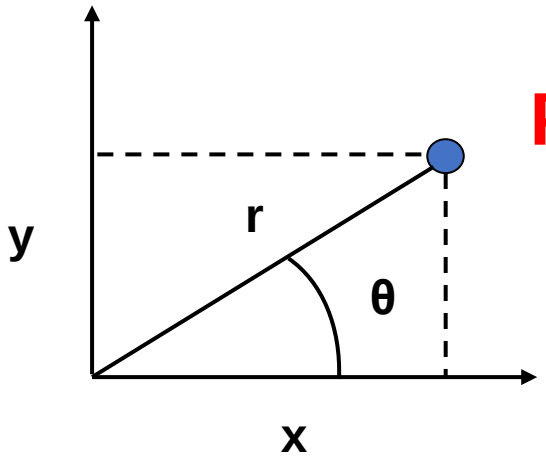
`Point` (without "Accessors"):

```
public class Point {
   public double x, y ;
   public Point () {
      x = 0.0 ;
      y = 0.0 ;
   }
}
```

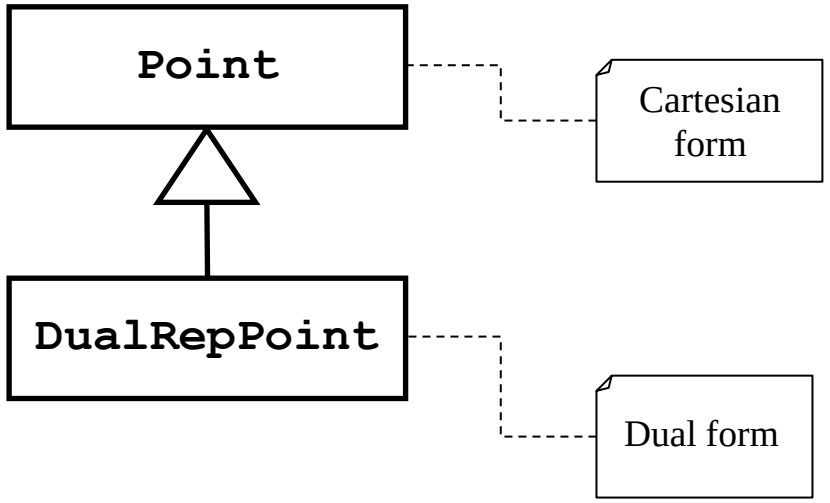
`Point` (with "Accessors"):

```
public class Point {
   private double x, y ;
   public Point (){
      x = 0.0 ;
      y = 0.0 ;
   }
   public double getX() {
      return x ;
   }
   public double getY() {
      return y ;
   }
   public void setX (double newX) {
      x = newX ;
   }
   public void setY (double newY) {
      y = newY ;
   }
}
```

Extend `Point` to `DualRepPoint`



P(x,y) = P(r,θ)



Our Target

- A class contains
  - x, y
  - r, θ
- And keep them consistent!
  - Update both of them

DualRepPoint (DRP): Ver. 1

```
public class DualRepPoint extends Point {
  public double radius, angle ;          ← Note public access

  /** Constructor: creates a default point with radius 2 at 45 degrees */
  public DualRepPoint () {
    radius = 2.0 ;
    angle = 45 ;
    updateRectangularValues();
  }

  /** Constructor: creates a point as specified by the input parameters */
  public DualRepPoint (double theRadius, double angleInDegrees) {
    radius = theRadius ;
    angle = angleInDegrees;
    updateRectangularValues();
  }

  /** Force the Cartesian values (inherited from Point) to be consistent */
  private void updateRectangularValues() {
    x = radius * Math.cos(Math.toRadians(angle));//legal assignments
    y = radius * Math.sin(Math.toRadians(angle));//(x & y are public)
  }
}
```

Client Using Public Access

```
/** This shows a "client" class that makes use of the "V. 1 DualRepPoint" class.
 *  It shows how the improper implementation of DualRepPoint (that is, use of
 *  fields with public access) leads to problems...
 */
public class SomeClientClass {

  private DualRepPoint myDRPoint ;

  // Constructor: creates a DualRepPoint with default values,
  // then changes the DualRepPoint's radius and angle values
  public SomeClientClass() {
      myDRPoint = new DualRepPoint() ;
      myDRPoint.radius = 5.0 ;     //update DualRepPoint's values
      myDRPoint.angle = 90.0 ;
  }
  ...
}
```

Anything wrong?

Problem

- Although r and θ are updated

- `updateRectangularValues()` is not called!
  - x, y are not updated

- x,y ←→ r, θ are not consistent now

DualRepPoint: Ver. 2

```
/** This class maintains a point representation in both Polar and Rectangular
 *  form and protects against inconsistent changes in the local fields */
public class DualRepPoint extends Point {

  private double radius, angle ;

  // constructors as before (not shown) ...
  public double getRadius() { return radius ; }
  public double getAngle() { return angle ; }

  public void setRadius(double theRadius) {
    radius = theRadius ;
    updateRectangularValues();
  }

  public void setAngle(double angleInDegrees) {
    angle = angleInDegrees;
    updateRectangularValues();
  }

  // force the Cartesian values (inherited from Point) to be consistent
  private void updateRectangularValues() {
    x = radius * Math.cos(Math.toRadians(angle));
    y = radius * Math.sin(Math.toRadians(angle));
  }
}
```

← New: private access

New: public accessors

Client Using DRP Accessors

```
/** This new version of the client code shows how requiring the use of accessors
 *  when manipulating the DualRepPoint radius & angle fields fixes (one) problem ...
 */

public class SomeClientClass {
  private DualRepPoint myDRPoint ;
  public SomeClientClass() {          // client constructor
    myDRPoint = new DualRepPoint(); // create a private DualRepPoint
    myDRPoint.setRadius(5.0) ; // alter DualRepPoint's values (safely): client has
    myDRPoint.setAngle(90.0) ; // no way to access radius/angle directly
  }
  .... etc.
}
```

Problem solved?

See next example

Accessing Other DRP Fields

```
/** This newer version of the client code shows how requiring the use of accessors
 *  when manipulating the DualRepPoint radius & angle fields fixes (one) problem
 *  ... but not all problems...
 */
public class SomeClientClass {

  private DualRepPoint myDRPoint ;

  public SomeClientClass() {       // client constructor as before
       myDRPoint = new DualRepPoint();
       myDRPoint.setRadius(5.0) ;
       myDRPoint.setAngle(90.0) ;
  }

  //a new client method which manipulates the portion inherited from Point
  public void someMethod() {
       myDRPoint.x = 2.2 ;
       myDRPoint.y = 7.7 ;
       ...
  }
}
```

See anything wrong?

Point Class

The x,y is still public!

- Client can modify it without updating r and θ.
- Inconsistent again

- That is why always better to use “Accessors”
  - Never know if someone will extend you code
  - Even if it is too simple

Public Fields *Break Code*

- **`Point` (without "Accessors"):**

```
public class Point {
    public double x, y ;
    public Point () {
      x = 0.0 ;
      y = 0.0 ;
    }
    ...
}
```

BAD BAD BAD

Using Accessors

- **`Point` (with "Accessors"):**

```
public class Point {
   private double x, y ;
   public Point (){
     x = 0.0 ;
     y = 0.0 ;
   }
   public double getX() { return x ; }
   public double getY() { return y ; }
   public void setX (double newX) {
     x = newX ;
   }
   public void setY (double newY) {
     y = newY ;
   }
}
```

Good !

Good !

Good !

Good !

Accessors Don’t Solve All Problems

```
/** This new version of the client code shows how requiring the use of accessors
 *  in ALL classes may have fixed ONE problem ... but another still exists
 */

public class SomeClientClass {

  private DualRepPoint myDRPoint ;

  public SomeClientClass() {   // client constructor
      myDRPoint = new DualRepPoint(); // create a private DualRepPoint
      myDRPoint.setX(2.2) ;// alter DualRepPoint’s inherited X,Y values
      myDRPoint.setY(7.7) ;//   using inherited accessors
  }
}
```

- **Problem still exists!**

Problem

We can use setX(), setY() now

But the r and θ are not updated in setX(), and setY()

Solution?

DualRepPoint: Correct Version

```
public class DualRepPoint extends Point  {    //uses "Good" Point with accessors

  private double radius, angle ;

  //...constructors and accessors for radius and angle here as before ...

  // Override inherited accessors

  public void setX (double xVal) {   //note that overriding the parent accessors
    super.setX(xVal) ;               // makes it impossible for a client to put
    updatePolarValues() ;            // put a DualRepPoint into an inconsistent state
  }

  public void setY (double yVal) {
    super.setY(yVal) ;
    updatePolarValues() ;
  }

  private void updateRectangularValues() {
    super.setX(radius * Math.cos(Math.toRadians(angle))) ;
    super.setY(radius * Math.sin(Math.toRadians(angle))) ;
  }

  //new private method to maintain consistent state
  private void updatePolarValues() {
    double x = super.getX() ;      // note: some people would use protected to
    double y = super.getY() ;      // allow direct subclass access to X & y
    radius = Math.sqrt (x*x + y*y) ;
    angle = MathUtil.atan2 (y,x) ;// in CN1, atan2() is a member of MathUtil class
  }
}
```

Small things for Java

Notes

The following things

- Not asking you to use:
- But test your understanding
- To know why do this happen if you saw them.

Constructor

Every class has a constructor

- Execute when you new an object

```
class A {
    public A() {
        System.out.print("A");
    }
}
```

Inheritance

Child inherit methods from parent

- Not include constructor

Constructor must have the same name with class

Questions

```
class A {
    public A() {
        System.out.print("A");
    }
}
```

```
A a = new A();
```

Output?

A

Questions

```
class A {
    public A() {
        System.out.print("A");
    }
}

class B extends A{
    public B() {
        System.out.print("B");
    }
}
```

```
B b = new B();
```

A or B?

AB

Explanation

- When a child class is created
  - The no argument constructor of parent class will be called.
  - Before the child class constructor

Questions

```
class A {
    public A(int a) {
        System.out.print("A");
    }
}

class B extends A{
    public B(int b) {
        System.out.print("B");
    }
}
```

```
B b = new B(1);
```

A or B?

Compile error

Explanation

Only the parent no-argument constructor will be called. But there is no no-argument constructor in the parent.

Questions

```
class A {
}
```

```
B b = new B(1);
```

```
class B extends A{
    public B(int b) {
        System.out.print("B");
    }
}
```

A or B?

B

Explanation

When there is no constructor in a class, a no-argument constructor is automatically generated

Questions

```
class A {
    public A(int a) {
        System.out.print("A");
    }
}

class B extends A{
    public B(int b) {
        super(1);
        System.out.print("B");
    }
}
```

```
B b = new B(1);
```

A or B?

AB

Explanation

As long as parent constructor is called, the no-argument will not be call anymore.

Questions

```
class A {
    public A(int a) {
        System.out.print("A");
    }
}

class B extends A{
    public B(int b) {
        System.out.print("B");
        super(1);
    }
}
```

```
B b = new B(1);
```

A or B?

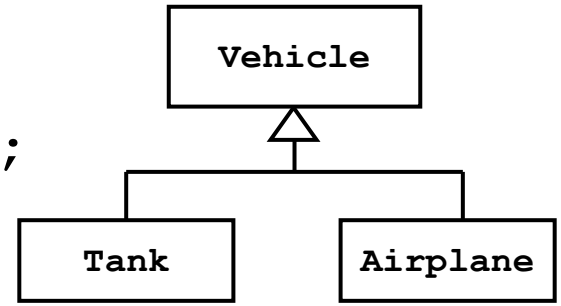
Compile error

Explanation

Parent must be constructed before child

Upcasting and Downcasting

```
Vehicle v ;
Airplane a = new Airplane();
Tank t = new Tank();
```



- "Upcasting" allowed in assignments:

```
v = t ;      // a tank IS-A Vehicle
v = a;       // an airplane IS-A Vehicle
```

- "Downcasting" requires casting:

```
t = v ;              // compiler error - a Vehicle
                        isn't a Tank
t = (Tank) v ;  // legal, but dangerous
```

Questions

```
class A {
    void go() {
        System.out.print("A");
    }
}

class B extends A{
    void go() {
        System.out.print("B");
    }
}
```

```
A a = new A();
a.go();
```

A or B?

A

Questions

```
class A {
    void go() {
        System.out.print("A");
    }
}

class B extends A{
    void go() {
        System.out.print("B");
    }
}
```

```
B b = new B();
b.go();
```

A or B?

B

Questions

```
class A {
    void go() {
        System.out.print("A");
    }
}

class B extends A{
    void go() {
        System.out.print("B");
    }
}
```

```
A a = new B();
a.go();
```

A or B?

B

Questions

```
class A {
    void go() {
        System.out.print("A");
    }
}

class B extends A{
    void go() {
        System.out.print("B");
    }
}
```

```
B b = new A();
b.go();
```

A or B?
Compile error

Explanation

- A is not a B as A is a superclass
  - Cannot put in the B type reference

Questions

```
class A {
    void go() {
        System.out.print("A");
    }
}

class B extends A{
    void go() {
        System.out.print("B");
    }
}
```

```
B b = (B)new A();
b.go();
```

A or B?
Exception

Explanation

- The actual type of the object is A
  - Cannot downcast to this subtype

Questions

```
class A {
    public char val = 'A';
    void go() {
        System.out.print(val);
    }
}
class B extends A{
    public char val = 'B';
    void go() {
        System.out.print(val);
    }
}
```

```
B b = new B();
b.go();
```

A or B?

B

Questions

```
class A {
    public char val = 'A';
    void go() {
        System.out.print(val);
    }
}

class B extends A{
    public char val = 'B';
}
```

```
B b = new B();
b.go();
```

A or B?

A

Explanation

- The call are using parent method and thus using parent variable.

Questions

```
class A {
    public char val = 'A';
    public char get() { return val; }
    void go() { System.out.print(get()); }
}

class B extends A{
    public char val = 'B';
    public char get() { return val; }
}
```

```
B b = new B();
b.go();
```

A or B?

B

Explanation

- The call are using parent method but then go back to child method due to the get() and thus using child variable.

Questions

```
class A {
    public char val = 'A';
}

interface B {
    public char val = 'B';
}

class C extends A implements B {
    public void go(){
        System.out.print(val);
    }
}
```

```
C c = new C();
c.go();
```

A or B?

Compile error

Explanation

There are two variables with the same name in the class C. Compiler do not know which one is calling.

Solution

How can we call the variable if the parent class and interface has variables with same name?

To call parent class:

- `super.val`

To call the interface one:

- `B.val`

Questions

```
class A {
    private char val = 'A';
}

interface B {
    public char val = 'B';
}

class C extends A implements B {
    public void go(){
        System.out.println(val);
    }
}
```

```
C c = new C();
c.go();
```

A or B?

B

Questions

```
class A {
    public char val = 'A';
}

interface B {
    private char val = 'B';
}

class C extends A implements B {
    public void go(){
        System.out.println(val);
    }
}
```

```
C c = new C();
c.go();
```

A or B?

Compile error

Explanation

- Every fields in interface must be public.

Questions

```
interface A {
        void go();
}
```

Valid?

Valid

Questions

```
interface A {
        private void go(){}
}
```

Valid?

No problem

Explanation

Java allows you to declare private methods in interface. Just no one can use it.

Questions

```
interface A {
    void go();
}
```

```
interface B implements A{
    void go2();
}
```

Valid?
Compile error

Questions: Are They Valid?

- Interface `extends` interface 
- Interface `extends` class 
- Interface `implements` interface 
- Interface `implements` class 
- Class `extends` interface 
- Class `extends` class 
- Class `implements` interface 
- Class `implements` class 

Questions

```
class A {
    void go(){ ... }
}

 abstract class B extends A{
     void abstract go2();
 }

 class C extends B{
     void go2() { ... }
 }
```

Valid?

Valid

Questions

```
class A {
    abstract void go();
}

 class B extends A{
     void go() { ... }
 }
```

Valid?
Compile error

Explanation

- Class with abstract method must be abstract

Questions

```
abstract class A {
        void go(){ ... }
}

 class B extends A{
     void go() { ... }
 }
```

Valid?

Valid

Comparing to C++

Virtual Function

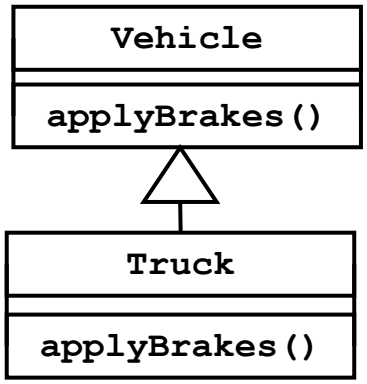
- In C++, only virtual function can be override

```
virtual void f() {...}
void f() override {...}
```

Optional

- In Java, every method is by default “Virtual”
  - Unless with **`final`** keywords

Java vs. C++ : Example



C++

```
class Vehicle {
  public:
    void applyBrakes() {
      printf ("Applying vehicle brakes...\n");
    }
};
class Truck : public Vehicle {
  public:
    void applyBrakes() {
      printf ("Applying truck brakes...\n");
    }
};
```

Java

```
class Vehicle {
   public void applyBrakes() {
      System.out.printf ("Applying vehicle brakes\n");
   }
}
class Truck extends Vehicle {
   public void applyBrakes() {
      System.out.printf("Applying truck brakes...\n");
   }
}
```

Non-Virtual

C++

```
void main (int argc, char** argv){
  Vehicle * pV ;
  Truck * pT ;
  pT = new Truck();
  pT->applyBrakes();
  pV = pT;
  pV->applyBrakes();
}
```

Java

```
public static void main (String [] args){
  Vehicle v;
  Truck t;
  t = new Truck();
  t.applyBrakes();
  v = t ;
  v.applyBrakes();
}
```

Output

```
Applying truck brakes...
Applying vehicle brakes...
```

```
Applying truck brakes...
Applying truck brakes...
```

Abstract Class

Java

- With keyword

```
Abstract
```

- Only abstract method can have no body

C++

- Use pure virtual function

```
virtual int f() = 0;
```

- Any function can have no body
  - Function declaration

Finally


NEW GAME
ASSIGNMENT ONE
Class Associations & Interfaces
In Canvas
FEB 09, 2023
Available on the
Canvas

Any Questions?