Overriding

What we learnt so far … Overloading

- Class diagram
  - Overloading allows to use the methods with ***identical name*** and ***different parameters*** in the same class.
  - The different sequences of parameters in methods are distinguished by the ***different number of parameters*** and the sequence of ***different data types***.

Overloading and overriding

- **Overloading**
  - Implementing identically named methods that take different arguments within the same class.
- **Overriding**
  - Implementing identically named method in the derived class that overrides the identically named method in base class.
- **Overriding a base class member function with a derived member function demonstrates the concept of polymorphism. Recall that polymorphism permits the same function name to take many forms.**

Overriding: definition

- Overriding allows methods of superclasses to be redefined in subclasses of a class hierarchy. Within a class hierarchy a subclass may override methods of any of its superclasses by redefining the method. The method is redefined by using the same method name and parameters of the method that is being overridden. Now when the method is invoked for an object of that subclass the new definition of the method is called not the previous superclasses definition of that method

Overriding: advantages

- Overriding makes code extensible and maintainable since class libraries that have been written by other programmers can be used and extended without changing the functionality of the methods of the superclasses.

Overriding Example

```
class Person {
   ...
   void printname(){
      System.out.println(name);
   }
   void printaddr(){
      printname();        // Apparently, a call to method above
      System.out.println(address);
   }
}
class Attorney extends Person{
   ...
   void printname(){              // Method override
     System.out.println(name + ", Esquire");
   }
```

Overriding
Example cont.

```
class Knight extends Person{
  private int waist_line;

  Knight(String n, int y, String a, int w){
    super(n,y,a);
    waist_line = w;
  }
  void printname(){
    System.out.println("Sir " + name);
  }
}
```

compilation error: name is private in Person, not accessible in Knight!

(same problem in Attorney)

Must make name "protected"

```
class Person {
  protected String name;
...
}
```

Overriding
It works!!!

```
public class InhTest{
  public static void main(String args[]){
    Person p = new Person("E.Bronte" …          // details
    Attorney a = new Attorney("A.McBeal" …      // omitted,
    Knight k = new Knight("Topham Hat" …        // see below

    Person[] A = new Person[3]; A[0] = p; A[1] = a; A[2] = k;

    for (int i=0; i<A.length; i++) {
      A[i].printaddr();
      System.out.println();
    } ...
```

Output

```
E.Bronte
The Heights
Wuthering
...
```

```
...
A.McBeal, Esquire
Howe,Dewey,Cheatham and Wynne
Litigation City
...
```

```
...
Sir Topham Hat
The Roundhouse
Sodor Island
```

Implementation of outputData() function

In ClsPoint:

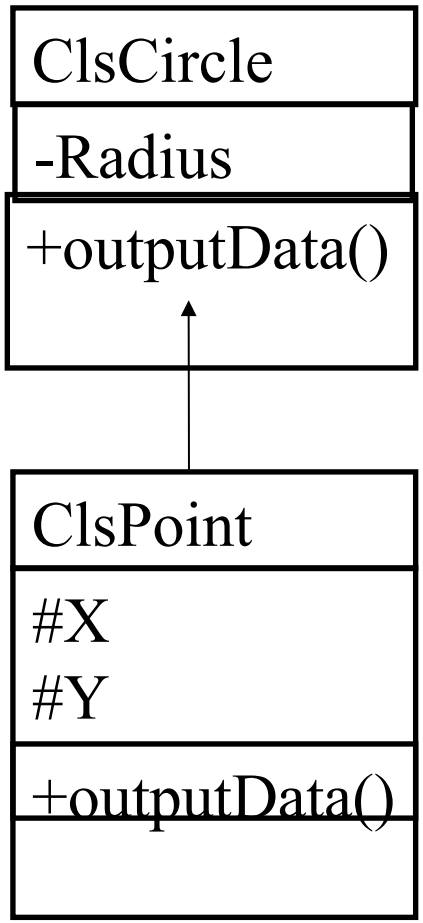
```
public void outputData(Graphics g)
{
    g.drawString("X: " + x + " Y: " + y, 10, 10);
}
```

In ClsCircuit:

```
public void outputData(Graphics g)
{
    g.drawString("X: " + x + " Y: " + y + " Radius: " + radius, 10,
    10);
}
```

Overriding parent class functions (7)

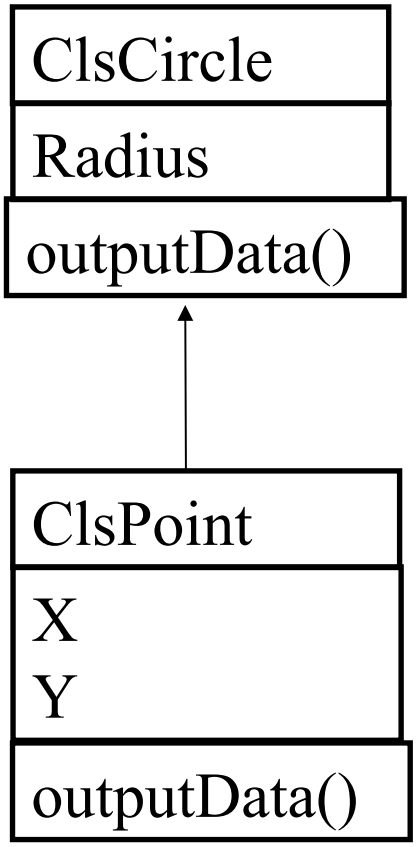
In order to implement the method mentioned above, the following code should be implemented.




```
public void test(void)
{
    ClsPoint aPoint;
    aPoint.setFields(2, 6);
    aPoint.outputData();
    ClsCircle aCircle;
    aCircle.setFields(2, 6, 1);
    aCircle.outputData();
}
```

The ClsPoint and ClsCircle class versions of outputData()

```
public void test(void)
{
    ClsPoint aPoint;
    aPoint.setFields(2, 6);
    aPoint.outputData();
    ClsCircle aCircle;
    aCircle.setFields(2, 6, 1);
    aCircle.outputData();
}
```



Overriding parent class functions (7)

```
public void test(void)
{
    ClsPoint aPoint;
    aPoint.setFields(2, 6);
    aPoint.outputData();
    ClsCircle aCircle;
    aCircle.setFields(2, 6, 1);
    aCircle.outputData();
}
```

Output of the program is:

X: 2 Y: 6

X: 23 Y: 16

Overriding and overloading parent class functions (7)

COMMENTS:

The output shows that even though you set fields for a ClsPoint and a ClsCircle by using separate functions that require separate argument lists, you use the same outputData() function that exists within the parent class for both a ClsPoint and a ClsCircle. If you want the ClsCircle class to contain its own outputData() function, you can override the parent version of that function as well.

Output of the program is:

X: 2 Y: 6

X: 23 Y: 16

The ClsPoint and ClsCircle class versions of outputData() (1)- Java

In ClsPoint:

```
public void outputData(Graphics g)
{
    g.drawString("X: " + x + " Y: " + y, 10, 10);
}
```

In ClsCircle:

```
public void outputData()
{
    super.outputData();
    g.drawString(" Radius: " + radius, 25, 10);
}
```

The ClsPoint and ClsCircle class versions of outputData() (2)- C++

Let us add the ClsCircle::outputData() function to the class (and add a prototype for the function in the class definition), then when you run the same main() function

```
void main()
{
    ClsPoint aPoint;
    aPoint.setFields(2, 6);
    aPoint.outputData();
    cout << endl<<endl;
                // double space
    ClsCircle aCircle;
    aCircle.setFields(23, 16, 1);
    aCircle.outputData();
}
```

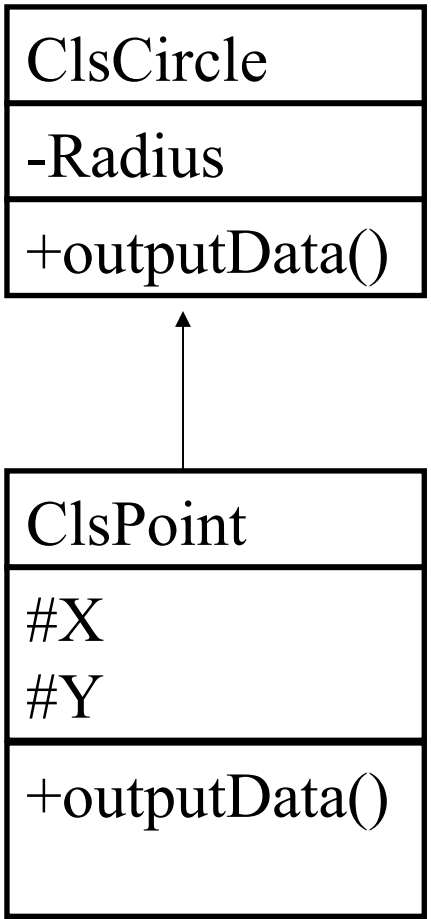
The ClsPoint and ClsCircle class versions of outputData() (2)- Java

Let us add the ClsCircle::outputData() function to the class (and add a prototype for the function in the class definition), then when you run the same main() function

```
public void test(void)
{
   ClsPoint aPoint;
    aPoint.setFields(2, 6);
    aPoint.outputData();
    ClsCircle aCircle;
    aCircle.setFields(23, 16, 1);
    aCircle.outputData();
}
```

The ClsPoint and ClsCircle class versions of outputData() (2)

```
public void test(void)
{
   ClsPoint aPoint;
    aPoint.setFields(2, 6);
    aPoint.outputData();
    ClsCircle aCircle;
    aCircle.setFields(23, 16, 1);
    aCircle.outputData();
}
```



The ClsPoint and ClsCircle class versions of outputData() (2)

```
public void test(void)
{
   ClsPoint aPoint;
    aPoint.setFields(2, 6);
    aPoint.outputData();
    ClsCircle aCircle;
    aCircle.setFields(23, 16, 1);
    aCircle.outputData();
}
```

Output of the program is:

X: 2 Y: 6

X: 23 Y: 16

Radius: 1

The ClsPoint and ClsCircle class versions of outputData() (2)

The different output formats demonstrate that the outputData() function called using the ClsPoint object differs from the outputData() function called using the ClsCircle object. Any ClsPoint object calls the ClsPoint functions. If a class derived from ClsPoint has functions with the same names as the ClsCircle class functions, the new class functions override the base class functions. The exception occurs when you use a class specifier with a function name: ClsPoint::setFields() or ClsPoint::outputData().

Output of the program is:

X: 2 Y: 6

X: 23 Y: 16

Radius: 1

The ClsPoint and ClsCircle class versions of outputData() (2)

Using the class name ClsPoint indicates precisely which class setFields() or outputData() function should be called. Thus. A child class object can use its own functions or its parent's (as long as the parent functions are not private). The opposite is not true – a parent object cannot use its child's functions. In other words, an ClsCircle is a ClsPoint and can do all the things a ClsPoint can do. However, every ClsPoint is not a ClsCircle; therefore, a ClsPoint cannot use ClsCircle functions.

Output of the program is:

X: 2 Y: 6

X: 23 Y: 16

Radius: 1

The ClsPoint and ClsCircle class versions of outputData()

In ClsCircle.java

```
public void outputData()
{
    outputData();
    cout << " Radius: " << radius
    << endl;
}
```

Output of the program is:

X: 2 Y: 6

Radius: 1

Radius: 1

Radius: 1

Radius: 1

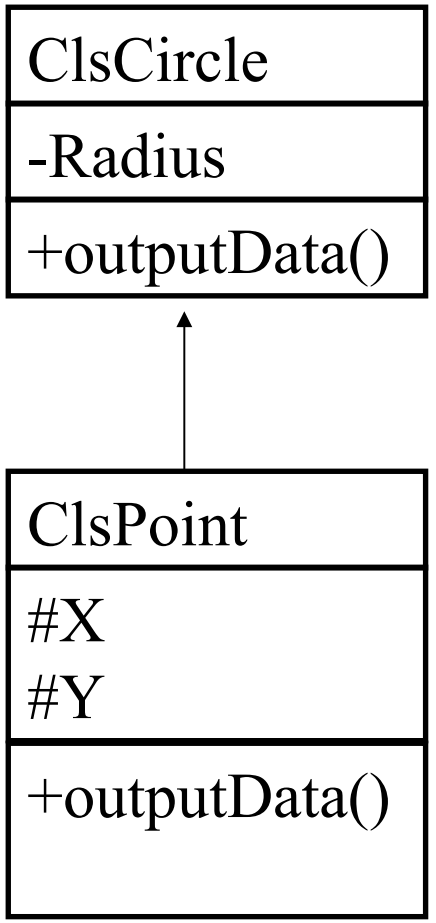
Radius: 1

Radius: 1

Radius: 1

The ClsPoint and ClsCircle class versions of outputData() (2)

The functions used by members of the ClsCircle class remain separate from those used by members of the ClsPoint class. These functions are not overloaded. Overloaded functions, you will recall, require different parameter lists, and the outputData() functions in ClsPoint and ClsCircle have identical parameter lists. Instead, ClsCicle's outputData() function ***overrides*** the outputData() function defined in ClsPoint.



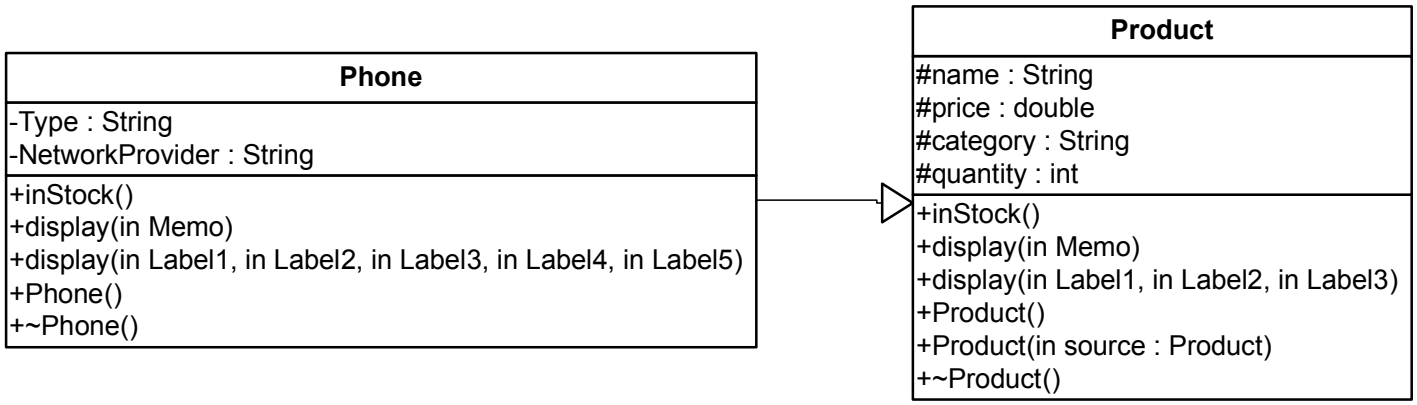
Task to do

- In the next set of steps, you create a ClsSphere class as a child of the ClsCircle. A ClsSphere has all the attributes of a ClsCircle, but they are described by using one more parameter – z.
- 1. Declare class ClsSphere and implement inheritance from the ClsCircle class.
- 2. Declare and implement setFields() and outputData() functions for the ClsSphere class.
- 3. Write the main() function based on the example given earlier.

Typical exam questions

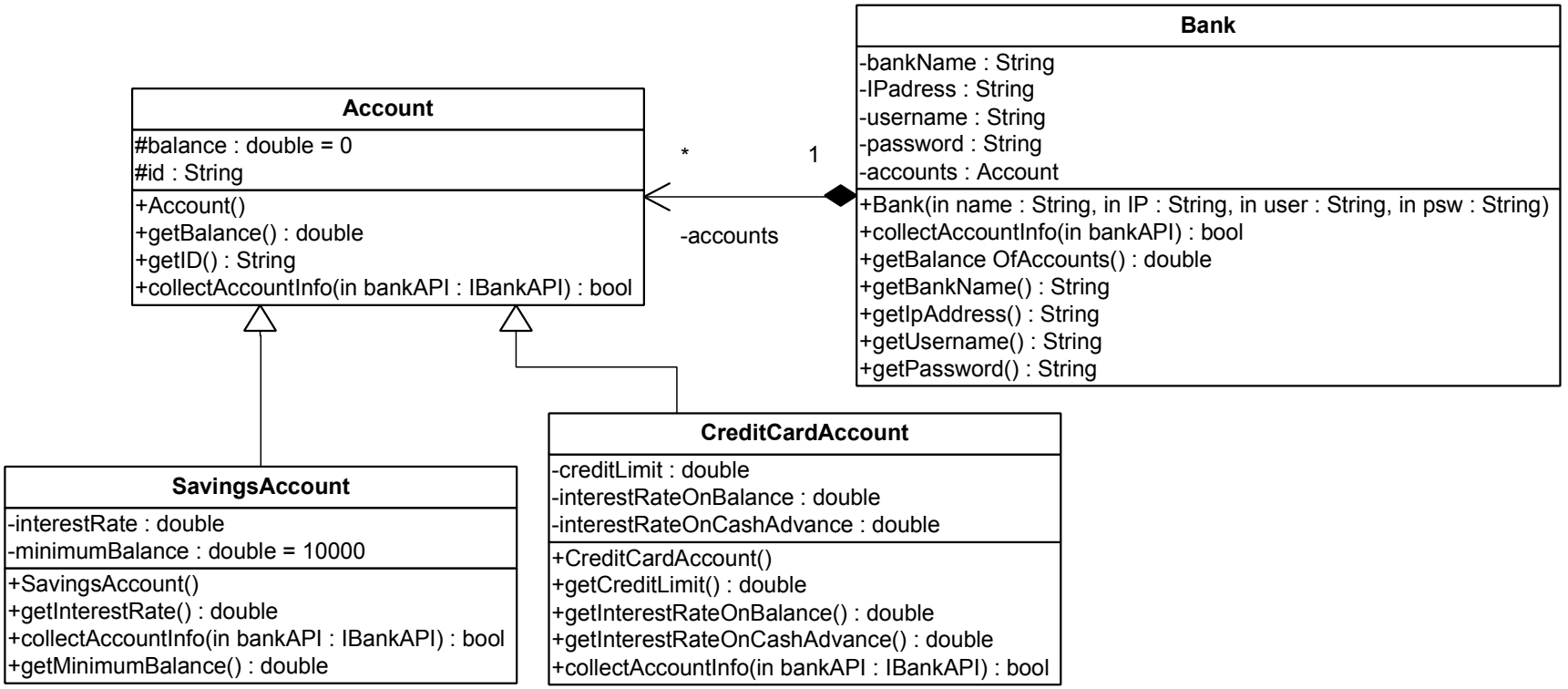
- Using either C++ or Java implement the respective ***display*** methods shown in Figure Q2b for the classes ***Phone*** and ***Product***. What are the terminologies commonly used to describe the implementation pattern/process these respective sets of methods are involved in?
  [7 marks]

Figure Q2b.



Typical exam questions

- Explain the main principles of overloading and overriding. Identify the functions involved in overriding and overloading in the class diagram given in Figure 4.2. If no function exists, modify the class diagram to demonstrate both principles.
  [6 marks]



What we learnt so far … Overriding

- Class diagram
  - Overriding allows to use the methods with ***identical name*** and ***identical parameters*** inside of inheritance.
  - In this case we will say the method in derived class ***overrides*** the method in the base class.