COMP 250

Lecture 31

inheritance (cont.)
interfaces
abstract classes

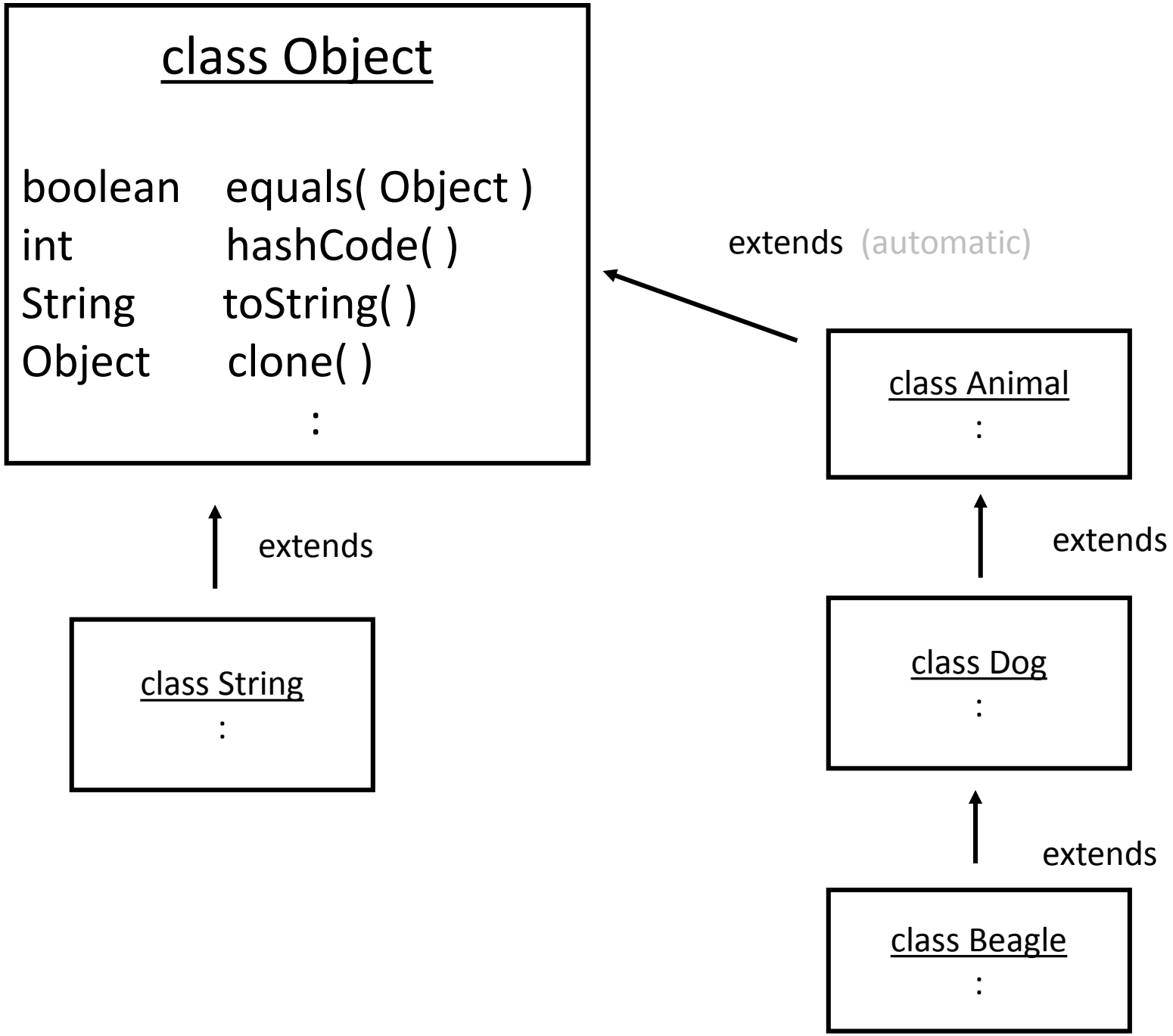
Nov. 20, 2017

CSUS Helpdesk

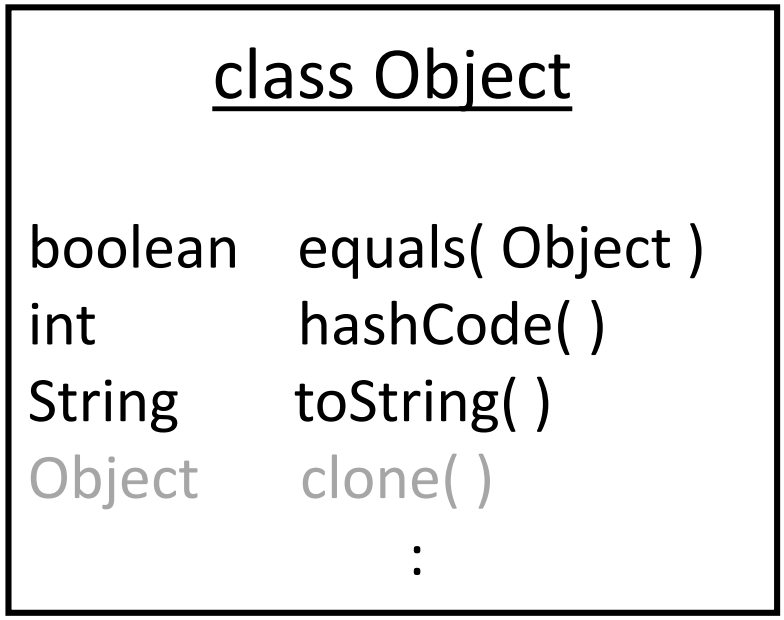
Have you used the CSUS Helpdesk? We would like to hear your thoughts about Helpdesk!

Link to survey:

https://goo.gl/forms/yMQDaEiLT7vxpnzS2


class Object
boolean equals(Object)
int hashCode( )
String toString( )
Object clone( )
:
extends (automatic)
class Animal
:
extends
class String
:
extends
class Dog
:
extends
class Beagle
:

A few more details about …



[ASIDE: slide also added to last lecture]

Java API for Object.hashCode() recommends:

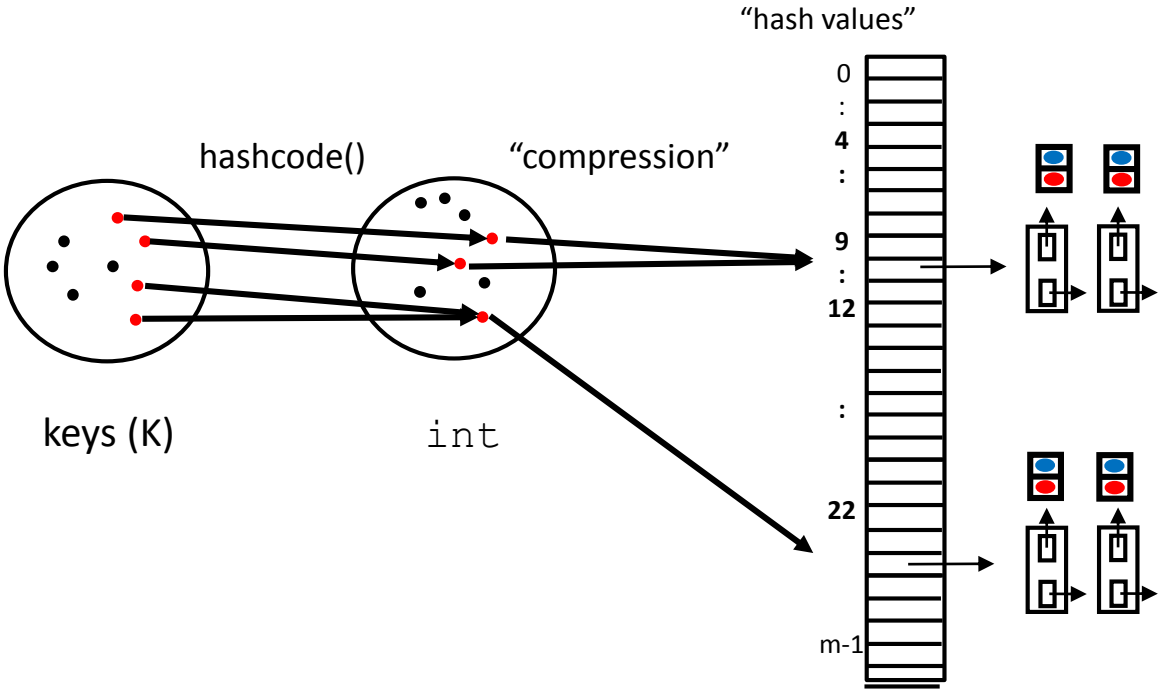
If o1.equals(o2) is true then

o1.hashCode() == o2.hashCode() should be true.

Why?

The converse need not hold. E.g. Two different Java strings might have the same hashCode().

Hash Map

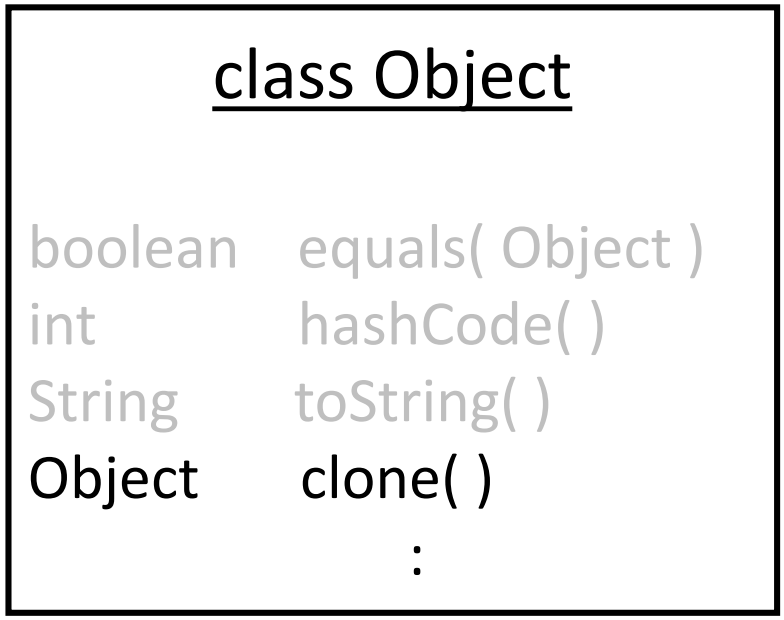


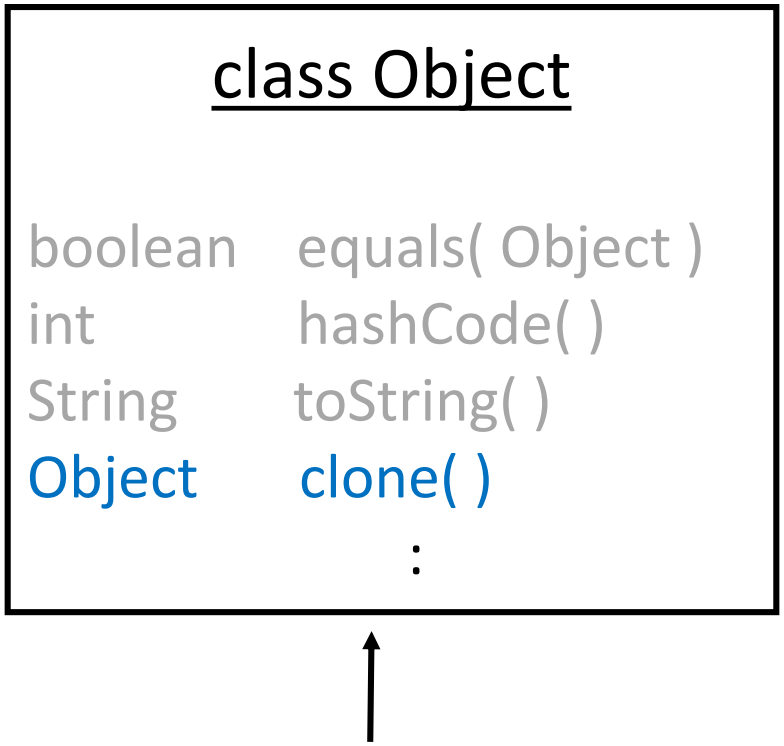
Java HashMap<K,V> *overrides* K.equals(Object) to compare keys.

If key1.equals(key2) is true,
then we want key1.hashCode() == key2.hashCode() to be true.

Otherwise get(key1) and get(key2) might return different values.

Finishing up last lecture





Object.clone()

makes a new object.

Recall Assignment 1

NaturalNumber.clone()

This is overriding.

Object.clone() recommendation

Q: x.clone() == x should be true or false ?

Q: x.equals(x.clone()) should be true or false ?

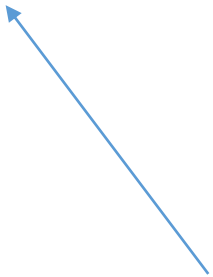
Object.clone() recommendation

Q: x.clone() == x should be true or false ?

A: false

Q: x.equals(x.clone()) should be true or false ?

A: true

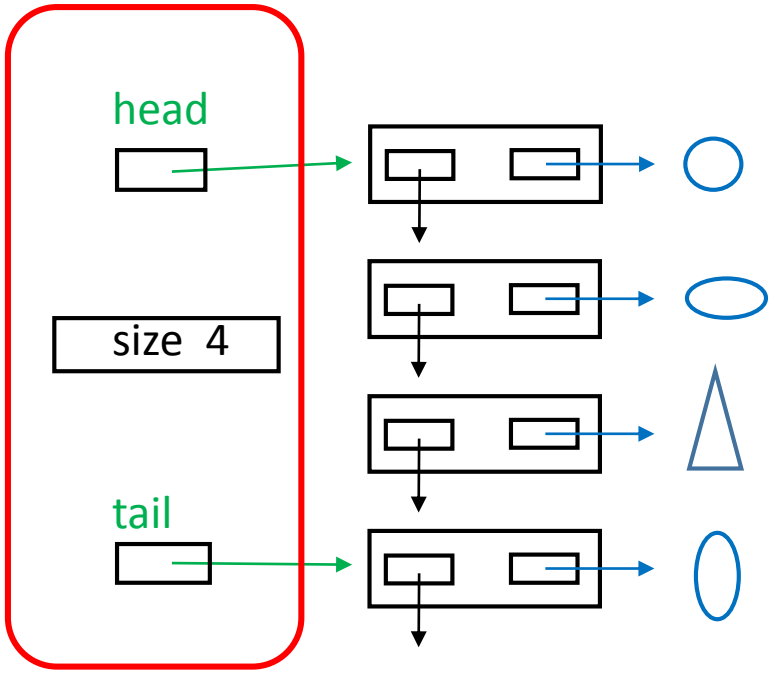


equals() needs to be carefully defined to ensure this

How to clone a list ?

SLinkedList<Shape> list

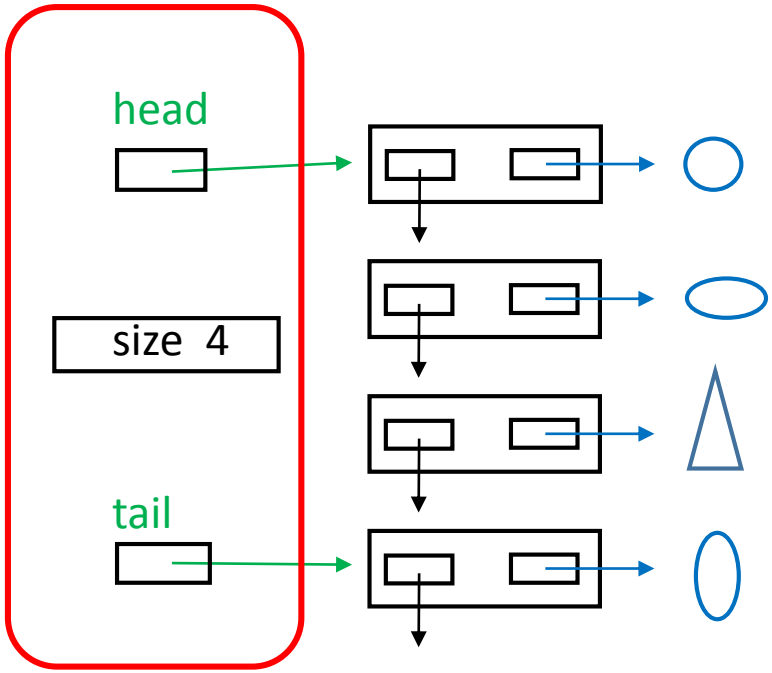
list.clone() = ?

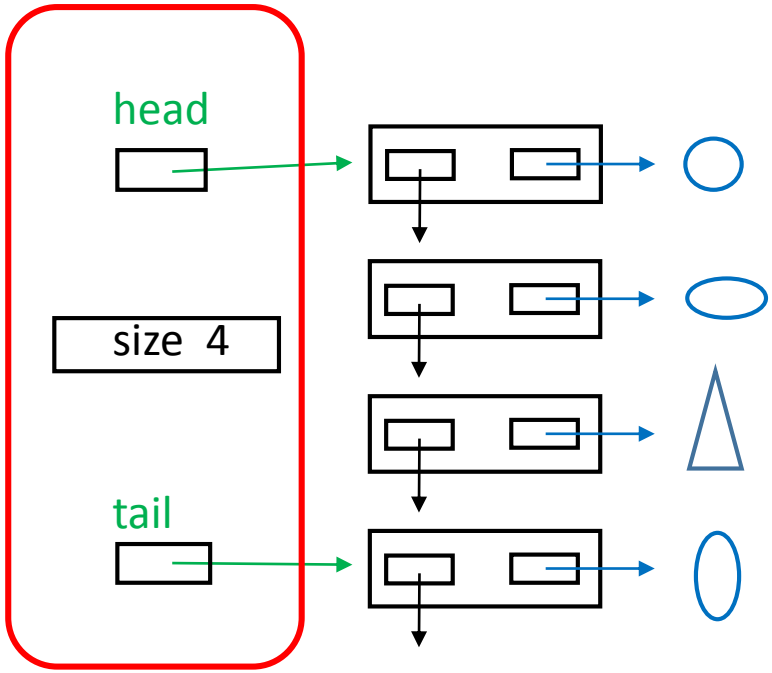


“deep copy”

SLinkedList<Shape> list

list.clone()

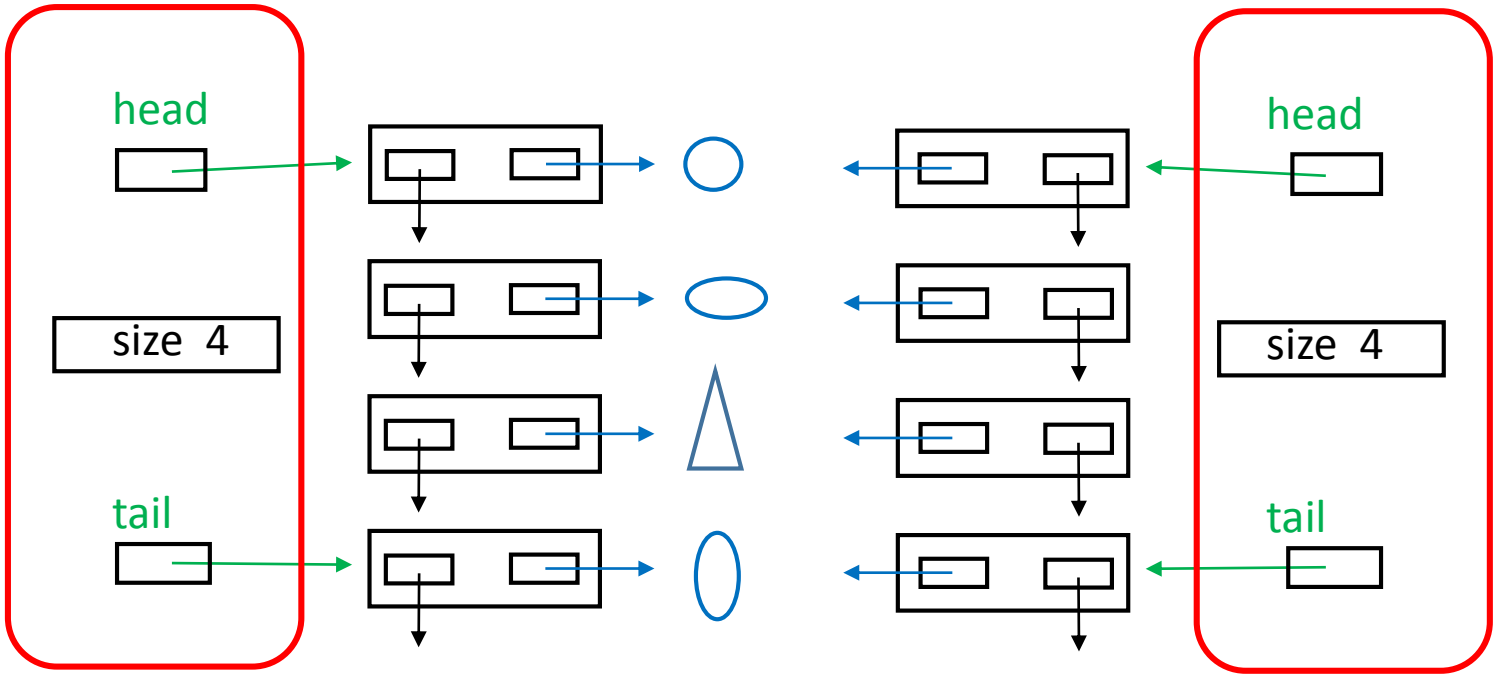




“shallow copy”

SLinkedList<Shape> list

list.clone()



https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

Java LinkedList<T>.clone() makes a shallow copy.

clone

`public Object clone()`

Returns a shallow copy of this `LinkedList`. (The elements themselves are not cloned.)

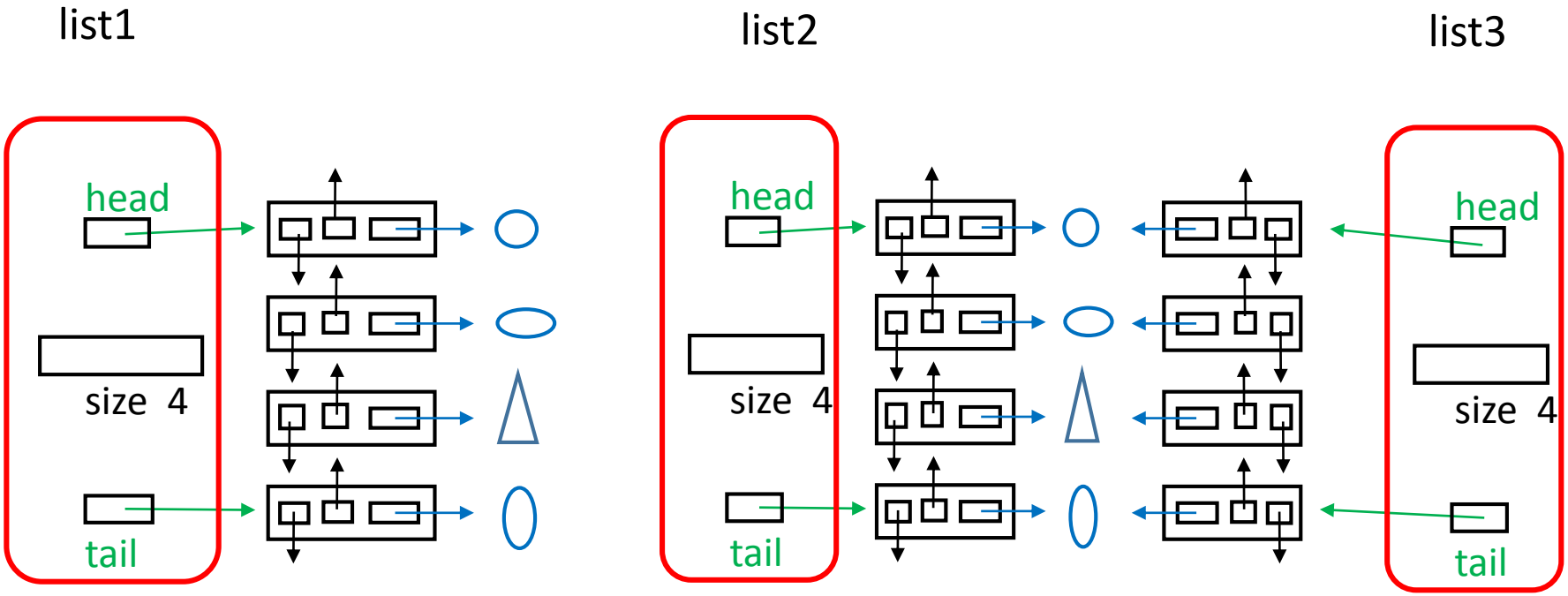
Overrides:

`clone` in class `Object`

Returns:

a shallow copy of this `LinkedList` instance

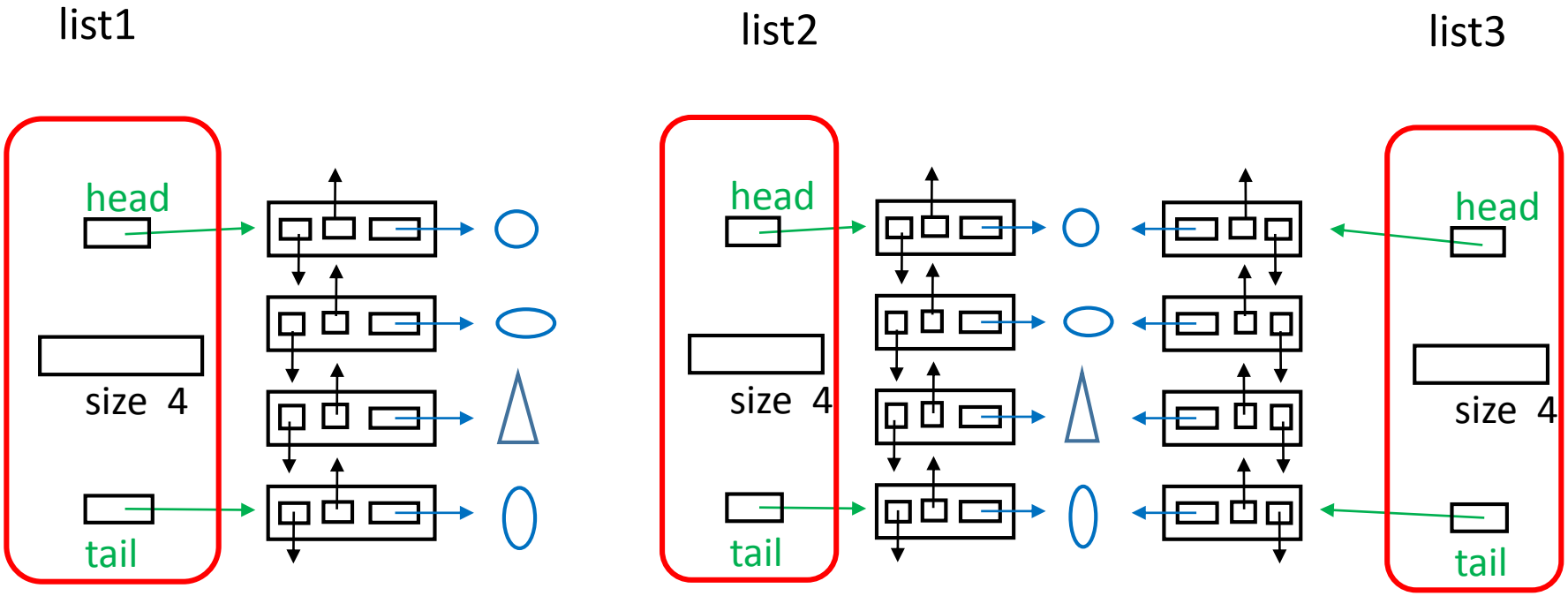
LinkedList<Shape> list1, list2, list3;



Assume Shape inherits the Object.equals(Object) method, i.e. equals() means "=="

list1.equals(list2) returns what ? list2.equals(list3) what ?

LinkedList<Shape> list1, list2, list3;



Assume Shape inherits the Object.equals(Object) method, i.e. equals() means "=="

list1.equals(list2) returns false. list2.equals(list3) returns true.

COMP 250

Lecture 31

inheritance (cont.)
interfaces
abstract classes

Nov. 20, 2017

Java `interface`

- reserved word
- like a class, but only the method signatures are defined

Example: List interface

```
interface  List<T>  {
      void          add(T)
      void          add(int,  T)
      T             remove(int)
      boolean       isEmpty()
      T             get( int )
      int           size()
                :
}
```

https://docs.oracle.com/javase/7/docs/api/java/util/List.html

```
class  ArrayList<T>  implements  List<T> {

      void        add(T)        { …. }
      void        add(int,  T)   {  ….  }
      T           remove(int) {  ….  }
      boolean     isEmpty()     {  ….  }
      T           get( int )     {  ….  }
      int         size()        {  ….  }
      void        ensureCapacity(int) { … }
      void        trimToSize( ) { … }
            :
}
```

Each of the List methods are implemented.
(In addition, other methods are implemented.)

```
class  LinkedList<T>  implements  List<T> {

      void        add(T)        { …. }
      void        add(int,  T)  {  ….  }
      T           remove(int)   {  ….  }
      boolean     isEmpty()     {  ….  }
      T           get( int )    {  ….  }
      int         size()        {  ….  }
      void        addFirst(T)   {  ….  }
      void        addLast(T)    {  ….  }
            :
}
```

Each of the List methods are implemented.
(In addition, other methods are implemented.)

How are Java interface's used ?

```
List<String>       list;

list  =  new  ArrayList<String>();
list.add("hello");
          :
list  =  new  LinkedList<String>();
list.add( new String("hi") );
```

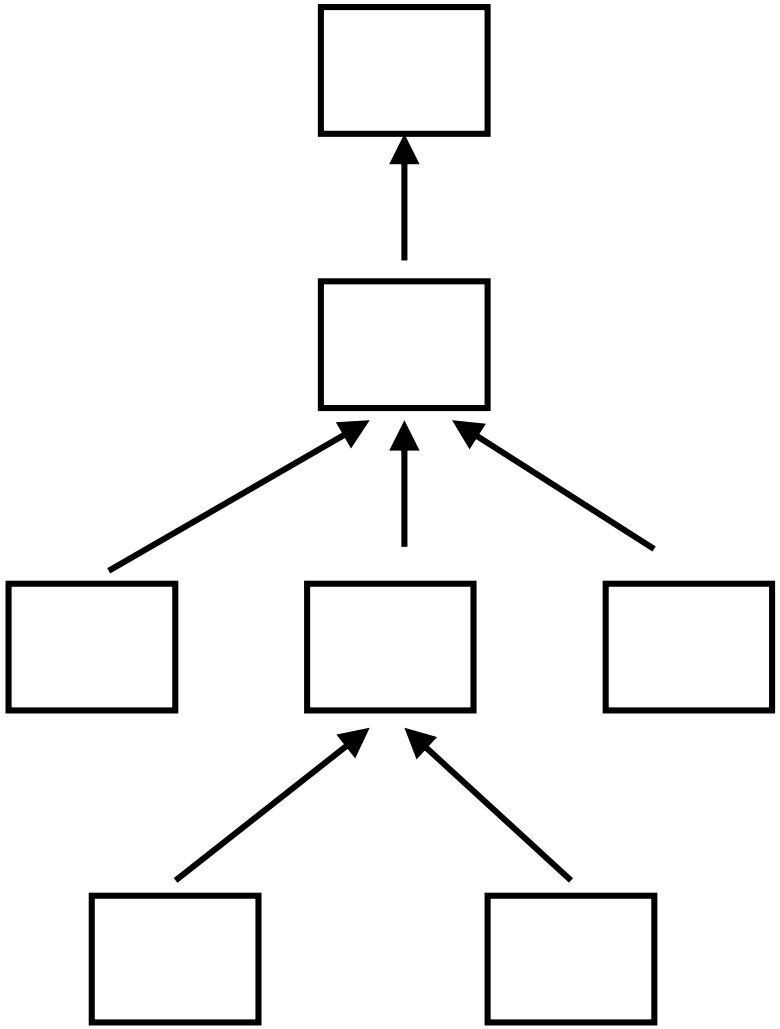
```
void  someMethod(    List<String>    list ){
      :
   list.add("hello");
      :
   list.remove( 3 );
}
```

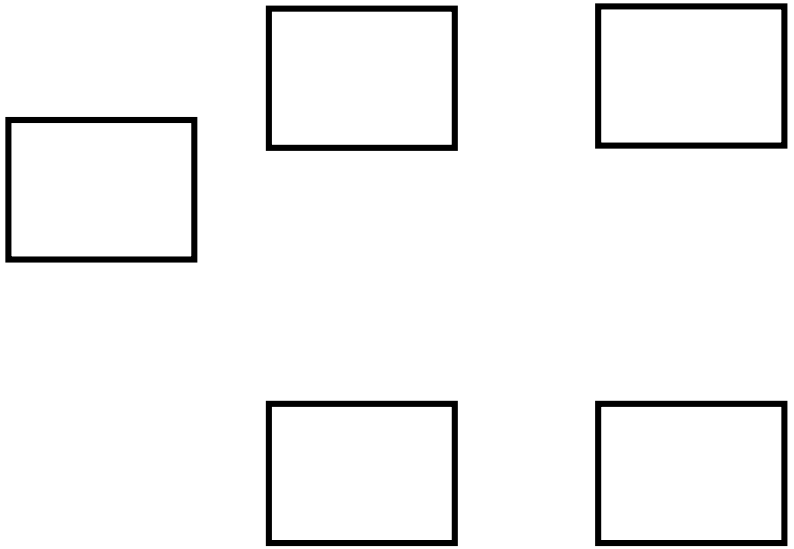
someMethod() *can be called with a LinkedList or an ArrayList as the parameter.*

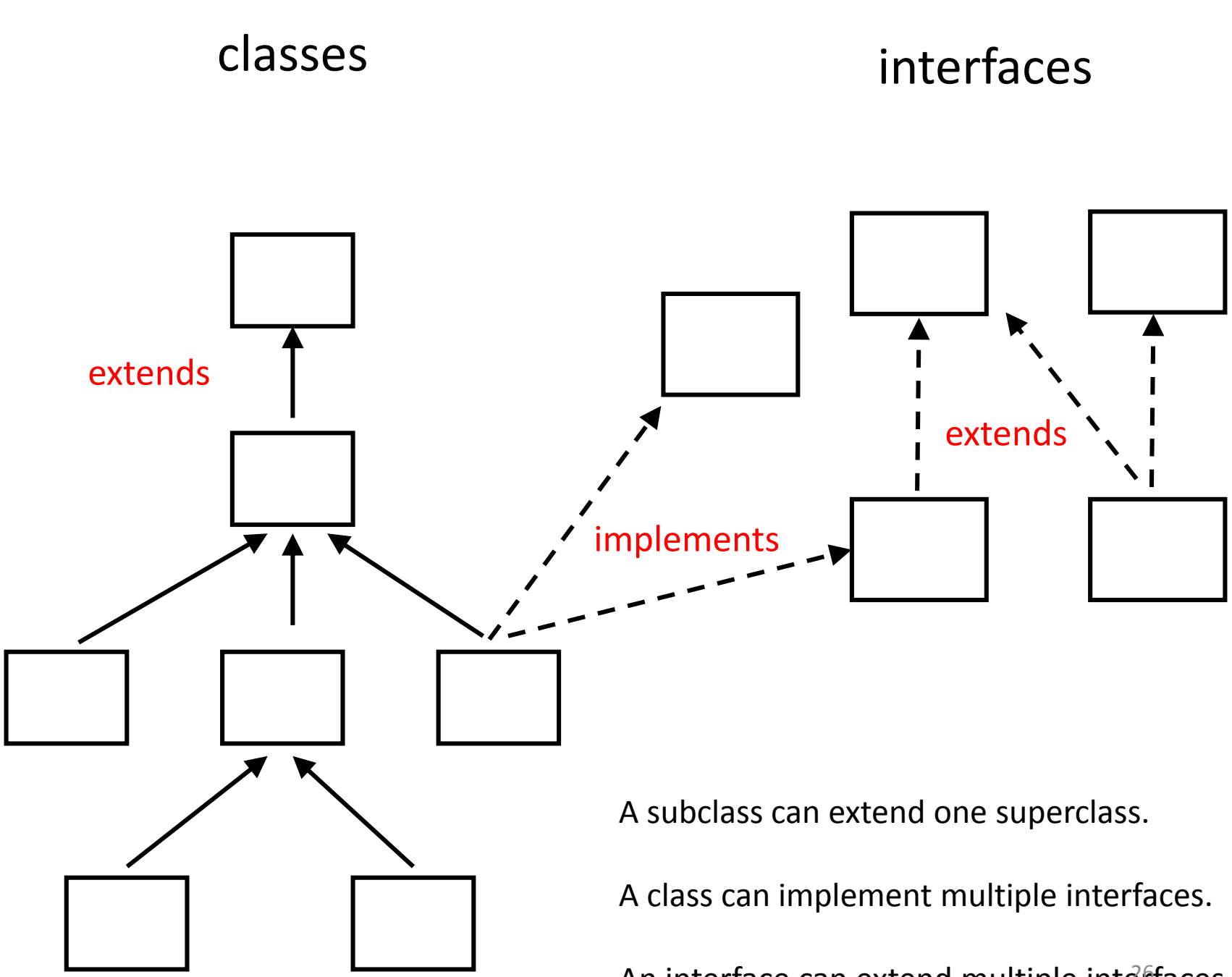
```
void  someMethod(    List<String>    list ){
      :
   list.add("hello");
      :
   list.remove( 3 );
   list.addFirst(  “have a nice day” );    //  compiler error
}
```

classes

interfaces





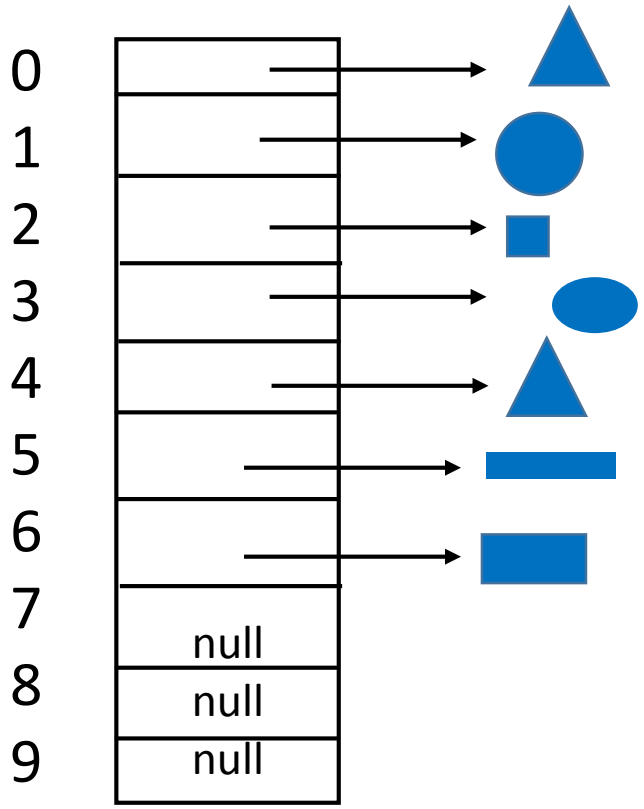


A subclass can extend one superclass.

A class can implement multiple interfaces.

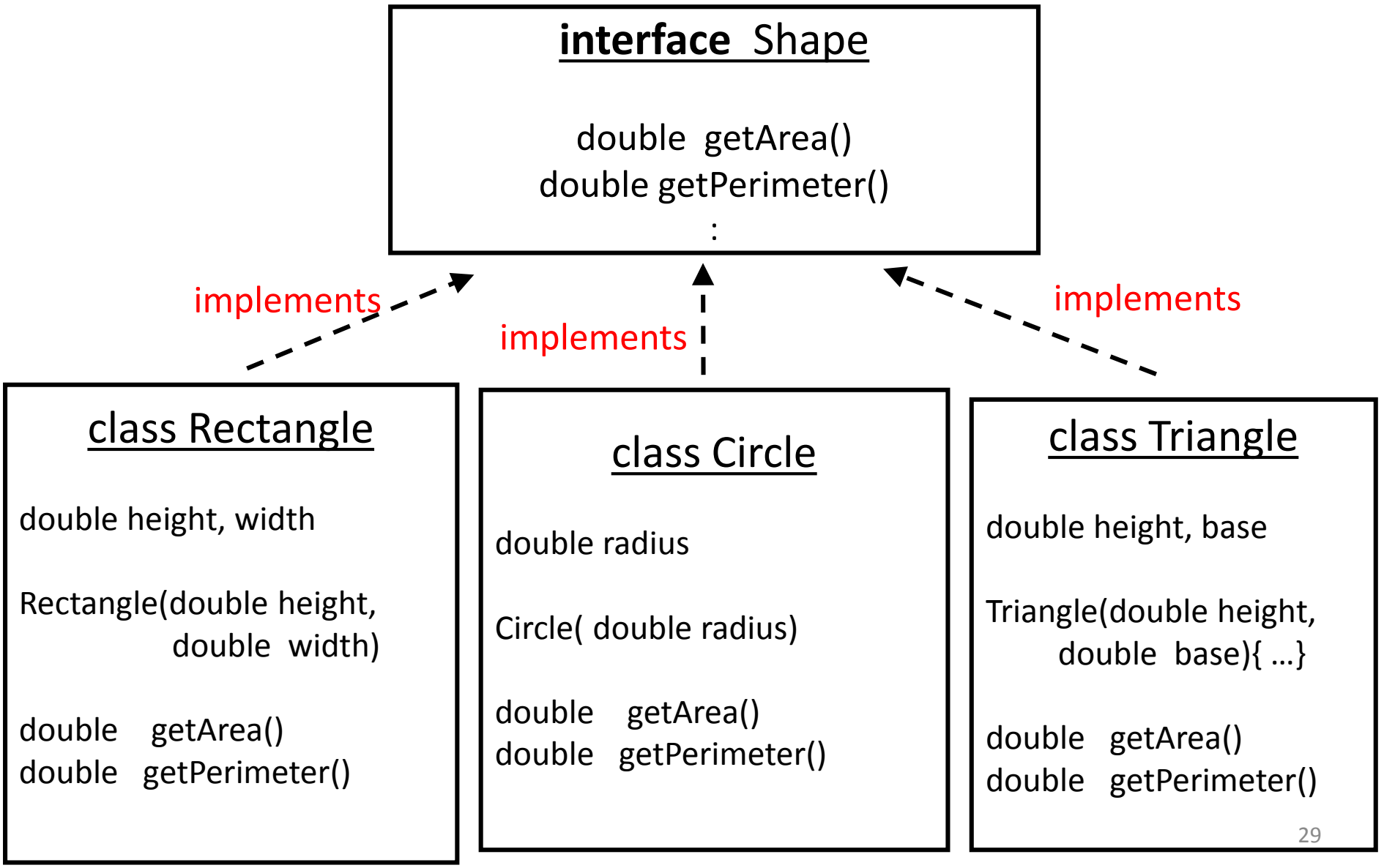
An interface can extend multiple interfaces.

Recall lecture 4 where we introduced ArrayList's and assumed a Shape class.



Would it make more sense for Shape to be an interface?

```
interface  Shape {
    double  getArea();
    double  getPerimeter();
}

class   Rectangle  implements  Shape{
        :
}

class   Circle  implements  Shape{
        :
}
```


interface Shape
double getArea()
double getPerimeter()
:
implements
implements
implements
class Rectangle
double height, width
Rectangle(double height,
double width)
double getArea()
double getPerimeter()
class Circle
double radius
Circle(double radius)
double getArea()
double getPerimeter()
class Triangle
double height, base
Triangle(double height,
double base){ …}
double getArea()
double getPerimeter()

```
class   Rectangle  implements  Shape{

    double  height,  width;

    Rectangle(  double h,  double w ){
          height = h;   weight = w;
    }

    double   getArea(){   return height * width;   }

    double   getPerimeter(){   return 2*(height + width);  }

}
```

```
class   Circle  implements  Shape{

    double  radius;

    Circle(  double r ){
          radius = r;
    }

    double   getArea(){   return   MATH.PI * radius * radius; }

    double   getPerimeter(){
                              return    2*MATH.PI * radius }

}
```

….. similarly for Triangle

```
Shape    s =  new  Rectangle( 30, 40 );
                  :
         s =  new   Circle(  2.5  );
                  :
         s =  new   Triangle(  4.5,  6.3  );
```

COMP 250

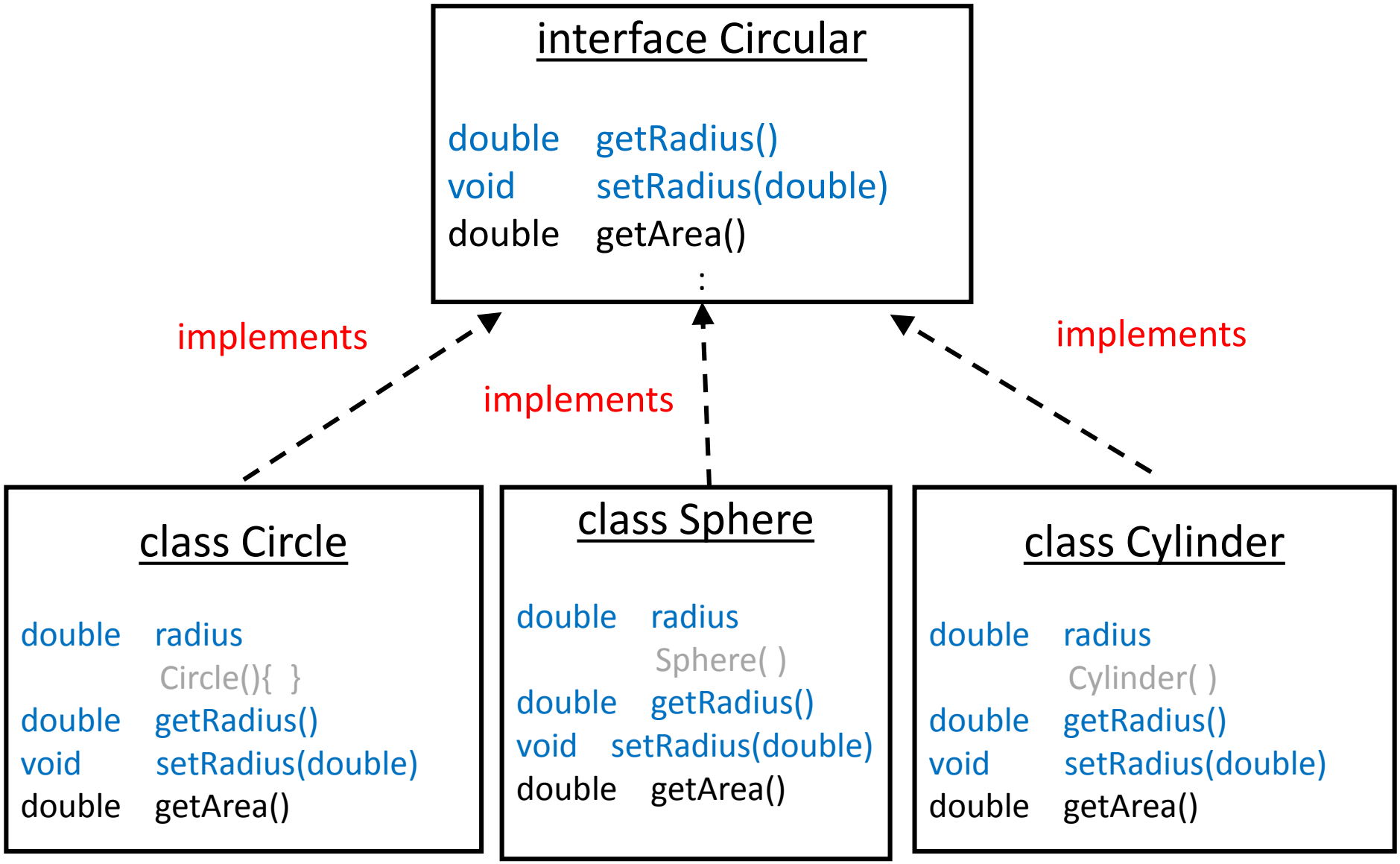
Lecture 31

inheritance (cont.)
interfaces
abstract classes

Nov. 20, 2017

Motivating Example: Circular

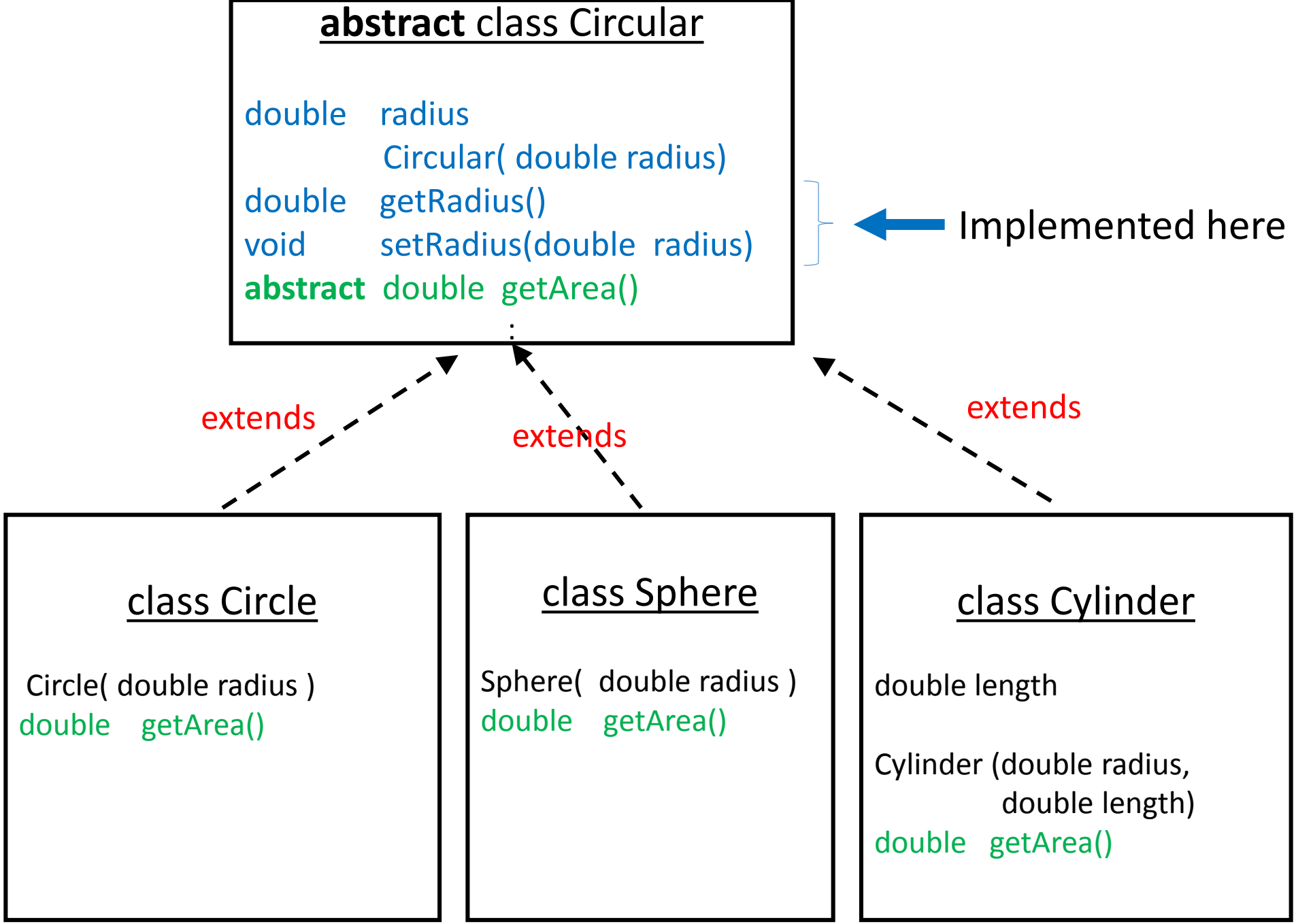
Circle Sphere Cylinder



Can we avoid repeating some of these definitions?

Abstract Class

- Like a class, it can have fields and methods with bodies
- Like an interface, it can have methods with only signatures.
- It cannot be instantiated, but it has constructors which are called by the sub-classes.


abstract class Circular
double radius
Circular(double radius)
double getRadius()
void setRadius(double radius)
abstract double getArea()
Implemented here
extends
extends
extends
class Circle
Circle(double radius)
double getArea()
class Sphere
Sphere(double radius)
double getArea()
class Cylinder
double length
Cylinder (double radius,
double length)
double getArea()

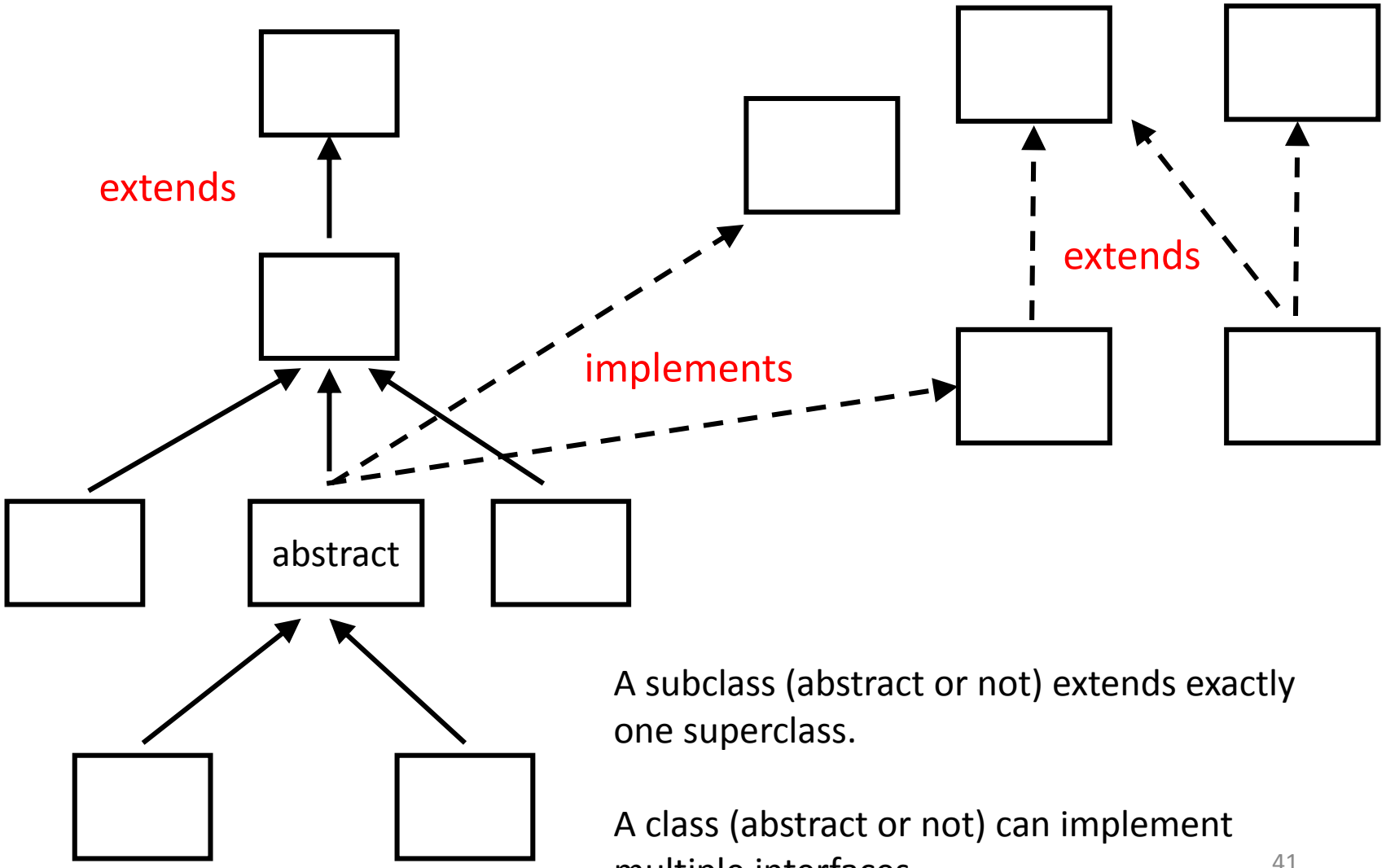
```
abstract class Circular {

        double radius;                  //  field

        Circular(double radius){        //  constructor
                this.radius = radius;
        }

        double getRadius(){             //  implemented methods
                return radius;
        }

        void setRadius(double r){
                this.radius = r;
        }

        abstract double  getArea();    //  abstract method
}
```

```
class Circle extends Circular{

        Circle(double radius){          // constructor
                super(radius);          //   superclass field
        }

        double getArea(){
                double r = this.getRadius();
                return  Math.PI * r*r;
        }
}
```

```
class Cylinder extends Circular{

        double height;

        Cylinder(double radius, double h){                  // constructor
                super(radius);
                this.height = h;
        }

        double getArea(){
                double r = this.getRadius();
                return  2 * Math.PI * radius * height;
        }
}
```

classes (abstract or not)

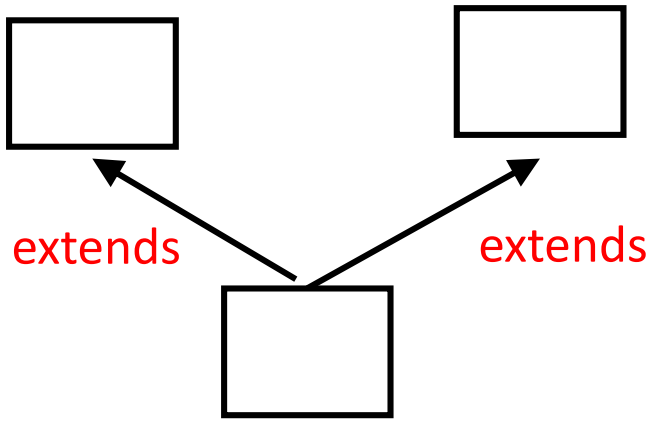
interfaces



A subclass (abstract or not) extends exactly one superclass.

A class (abstract or not) can implement multiple interfaces.

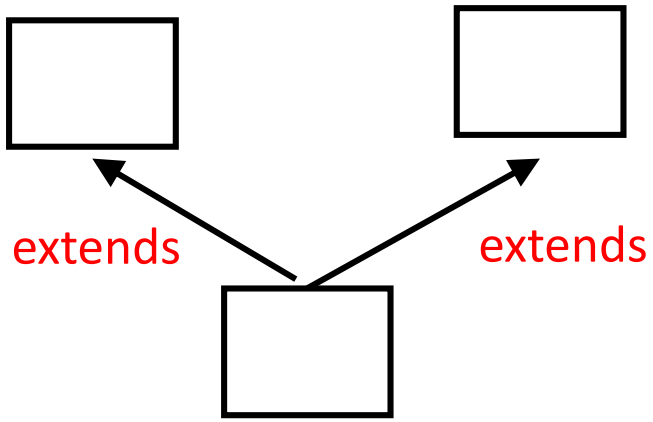
A class (abstract or not) cannot extend more than one class (abstract or not).



Not allowed!

Why not ?

A class (abstract or not) cannot extend more than one class (abstract or not).



Not allowed!

Why not ?

The problem could occur if two superclasses have implemented methods with the same signature. Which would be inherited by the subclass?