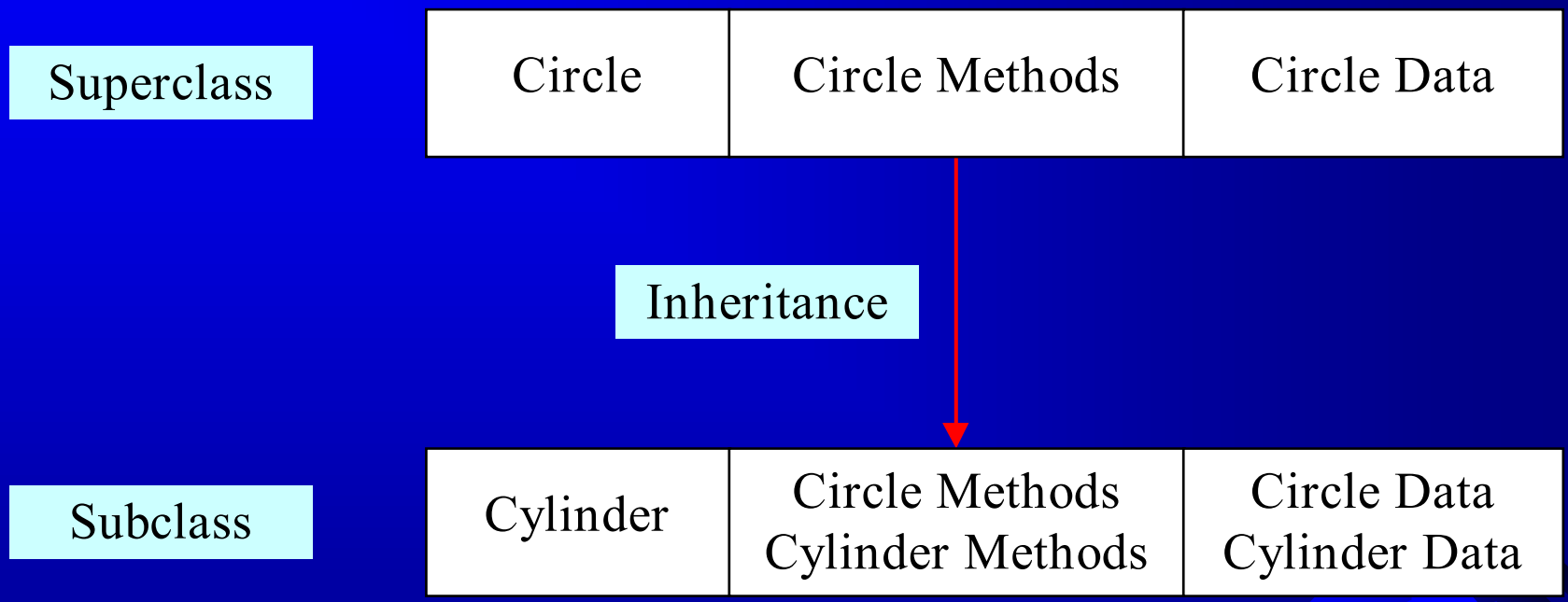
Chapter 6 Class Inheritance

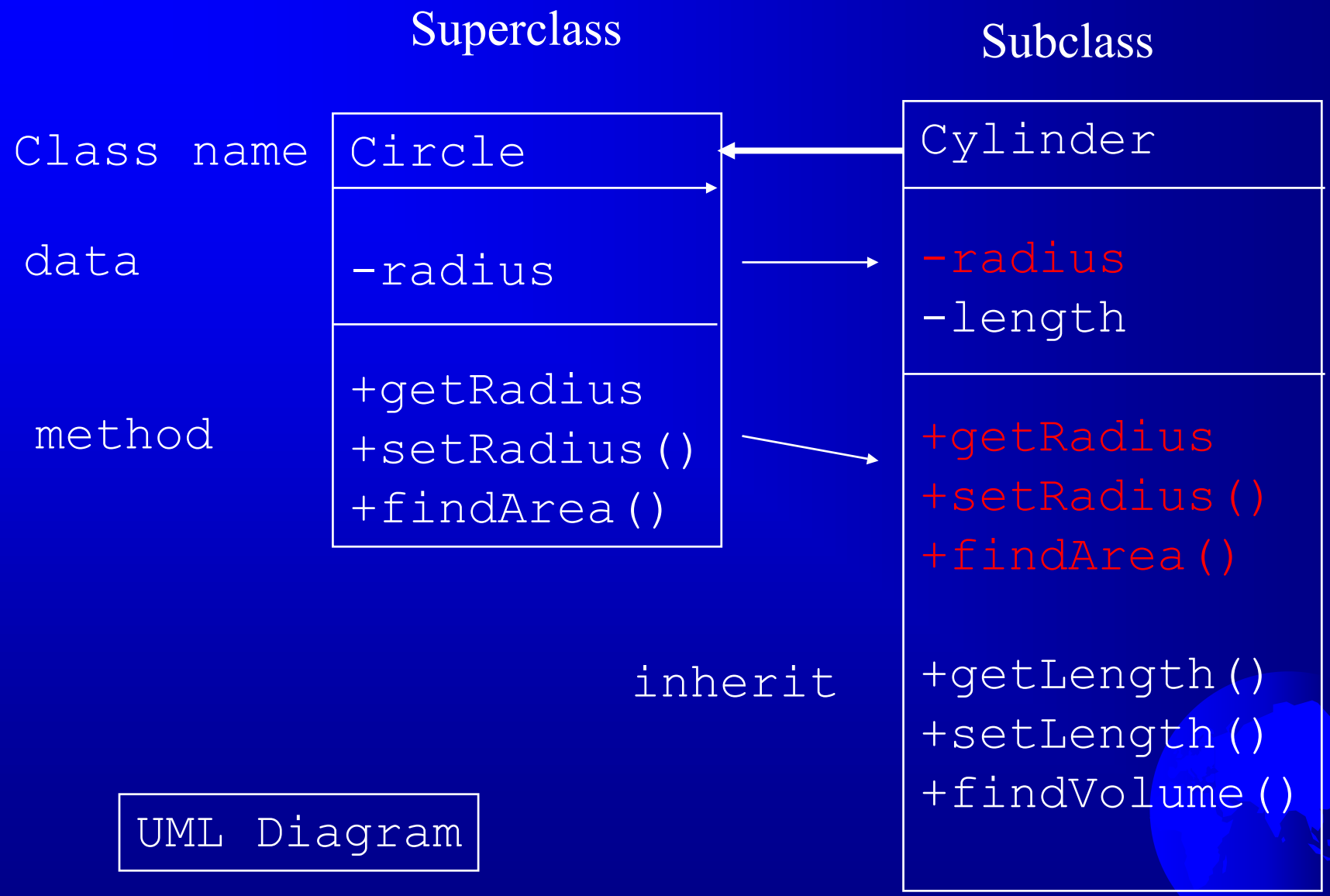
- Superclasses and Subclasses
- Keywords: `super`
- Overriding methods
- The `Object` Class
- Modifiers: `protected`, `final` and `abstract`
- Polymorphism and Object Casting
- Interfaces
- Inner Classes
- Program Development and Class Design

Chapter 6 Class Inheritance

- Objective: derive a new class from existing class
- Recognize the path of inheritance of classes in JDK
- Inherence model is the "is-a-relationship", not a "has-relationship"
- WelcomeApplet is an inherited class from Applet
- All of the classes you use or define are implicitly inherited from the top class, Object

Superclasses and Subclasses




Superclass
Subclass
Class name
Circle
Cylinder
data
-radius
-radius
-length
method
+getRadius
+setRadius()
+findArea()
+getRadius
+setRadius()
+findArea()
inherit
+getLength()
+setLength()
+findVolume()
UML Diagram

Creating a Subclass

Creating a subclass extends properties and methods from the superclass. You can also:

- Add new properties
- Add new methods
- Override the methods of the superclass
- Example: Cylinder class

Example 6.1
Testing Inheritance

- Objective: Create a `Cylinder` object and explore the relationship between the `Cylinder` and `CircleSecond` classes.

Using the Keyword `super`

The keyword `super` refers to the superclass of the class in which `super` appears. This keyword can be used in two ways:

- To call a superclass default constructor, super()
- To call a superclass general constructor, super(parameters)
- To call a superclass method, super.methodName(parameter)
- Example: super.findArea() to invoking CircleSecond.findArea() instead of findArea()

```
// Circle class second version, CircleSecond.java
Declare class Circle with private radius and accessor methods
public class CircleSecond
{
  private double radius;

  // Default constructor
  public CircleSecond()
  {
    radius = 1.0;
  }

  // Construct a circle with a specified radius
  public CircleSecond(double r)
  {
    radius = r;
  }
```

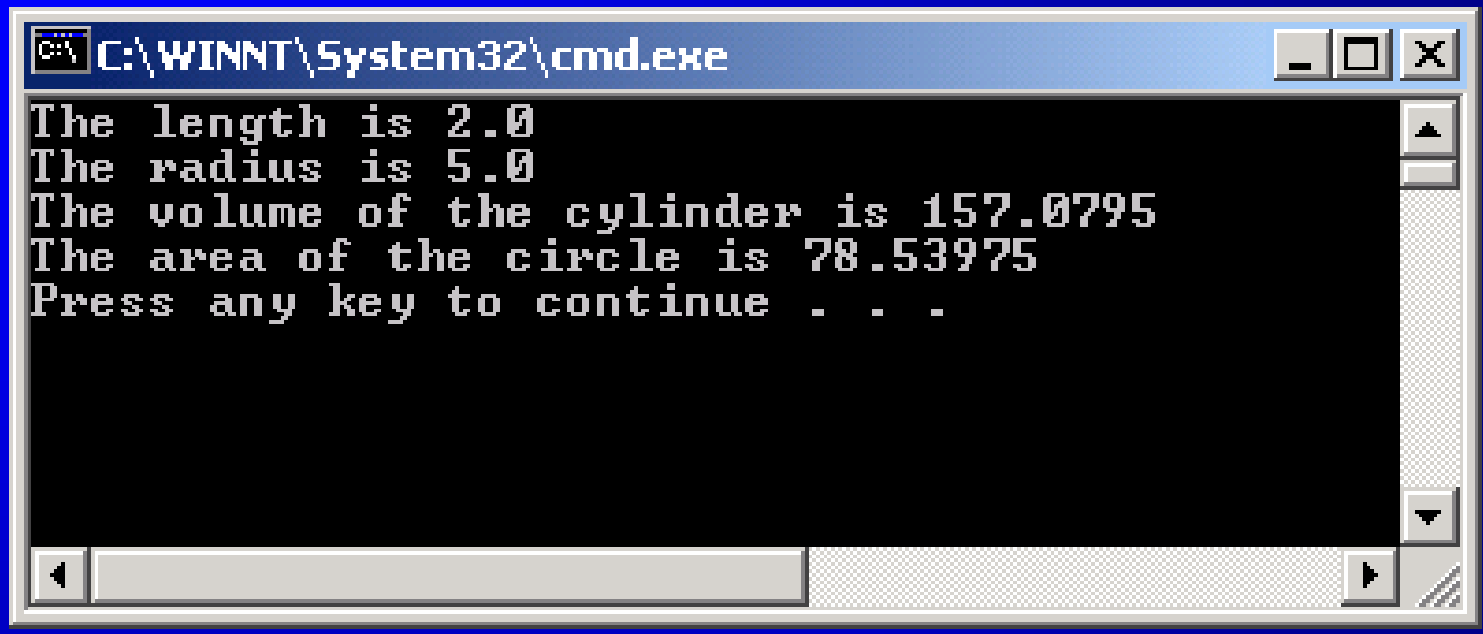
```
// Getter method for radius
 public double getRadius()
 {
  return radius;
 }

 // Setter method for radius
 public void setRadius(double newRadius)
 {
  radius = newRadius;
 }

 // Find the circle area
 public double findArea()
 {
  return radius*radius*3.14159;
 }
}
```

```
// Cylinder.java: The new cylinder class that extends the CircleSecond
// class
class Cylinder extends CircleSecond
{
  private double length;

  // Default constructor
  public Cylinder()
  {
    super();
    length = 1.0;
  }
```

```
// Construct a cylinder with specified radius, and length
 public Cylinder(double r, double l)
 {
   super(r);
   length=l;
 }
 // Getter method for length
 public double getLength()
 {
   return length;
 }
 // Setter method for length
 public void setLength(double l)
 {
   length = l;
 }
```

```
// Find cylinder volume
  public double findVolume()
  {
    return super.findArea()*length;    //   findArea()*length;
  }

}
```

```
// TestCylinder.java: Use inheritance
public class TestCylinder
{
  public static void main(String[] args)
  {
    // Create a Cylinder object and display its properties
    Cylinder myCylinder = new Cylinder(5.0, 2.0);
    System.out.println("The length is " + myCylinder.getLength());
    System.out.println("The radius is " + myCylinder.getRadius());
    System.out.println("The volume of the cylinder is " +
      myCylinder.findVolume());
    System.out.println("The area of the circle is " +
      myCylinder.findArea());
  }
}
```


C:\WINNT\System32\cmd.exe
The length is 2.0
The radius is 5.0
The volume of the cylinder is 157.0795
The area of the circle is 78.53975
Press any key to continue . . .

Example 6.2
Overriding Methods in the Superclass

- A subclass inherits methods from a superclass.
- Sometimes it is necessary for a sublcass to modify the methods defined in the superclass, or share with same method name, but have different implementation or functionality.
- For example: Implement findArea in Cylinder class is to calculate total surface area

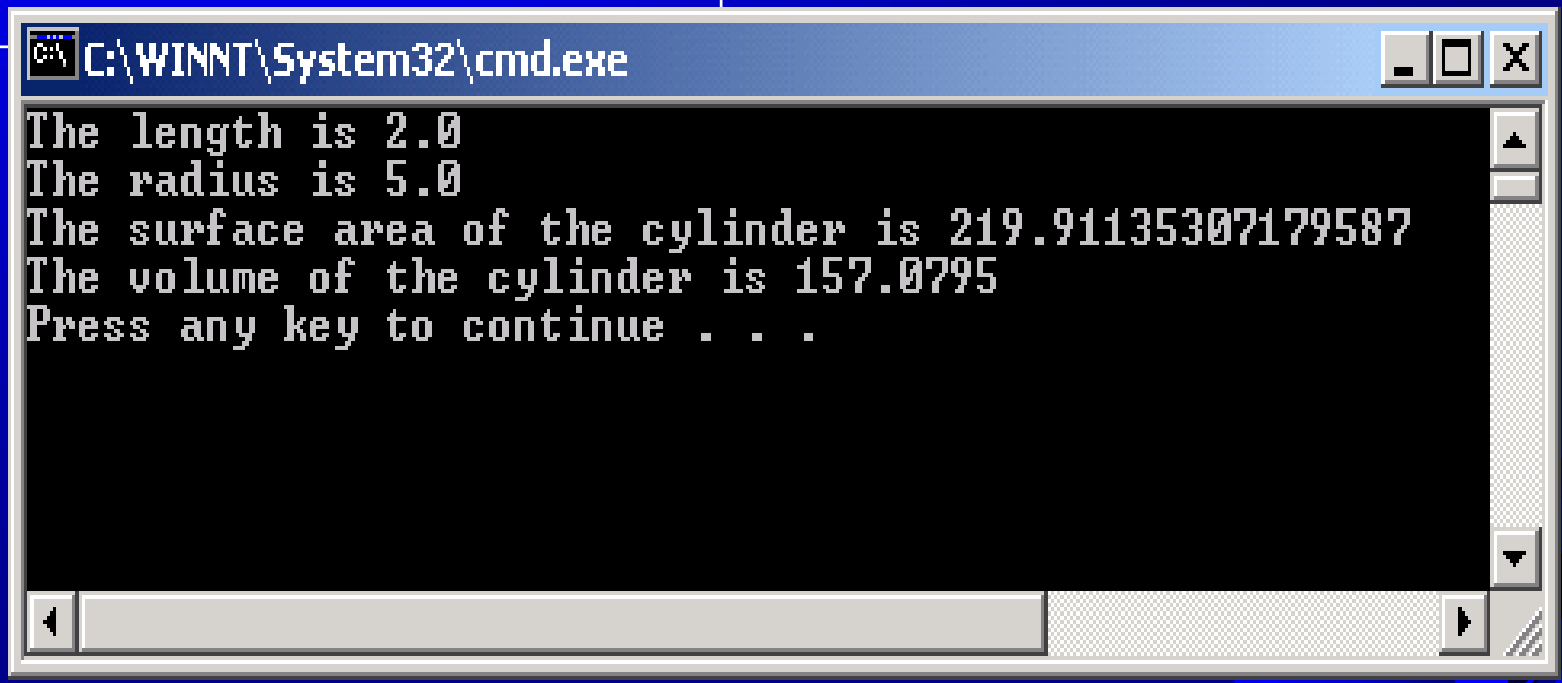
`Cylinder` class overrides the `findArea()` method defined in the `Circle` class.

```
// TestOverrideMethods.java: Test the Cylinder class that overrides
// its superclass's methods
public class TestOverrideMethods
{
  public static void main(String[] args)
  {
    Cylinder myCylinder = new Cylinder(5.0, 2.0);
    System.out.println("The length is " + myCylinder.getLength());
    System.out.println("The radius is " + myCylinder.getRadius());
    System.out.println("The surface area of the cylinder is "+
      myCylinder.findArea());
    System.out.println("The volume of the cylinder is "+
      myCylinder.findVolume());
  }
}
```

```
// New cylinder class that overrides the findArea() method defined in
// the circle class
class Cylinder extends CircleSecond
{
  private double length;

  // Default constructor
  public Cylinder()
  {
    super();
    length = 1.0;
  }
```

```
// Construct a cylinder with specified radius and length
 public Cylinder(double r, double l)
 {
   super(r);
   length = l;
 }

 // Getter method for length
 public double getLength()
 {
   return length;
 }
 // Find cylinder surface area
 public double findArea()
 {
   return 2*super.findArea()+(2*getRadius()*Math.PI)*length;
 }
```

```
// Find cylinder volume
public double findVolume()
{
  return super.findArea()*length;
}
}
```



The `Object` Class

- The `Object` class is the root of all Java classes.
- Every class implicitly inherit from java.lang.Object
- Similar with the methods provided by the Object class
- Three frequently used instant methods Object provides
  - The `equals()` method compares the contents of two objects, checking the object1 and object2 are of the same class or not.

The `Object` Class

- Equals () method is equivalent to the following implementation

```
public boolean equals(Object obj)
{
   return ( this ==obj );
}
```

The meaning of "this"

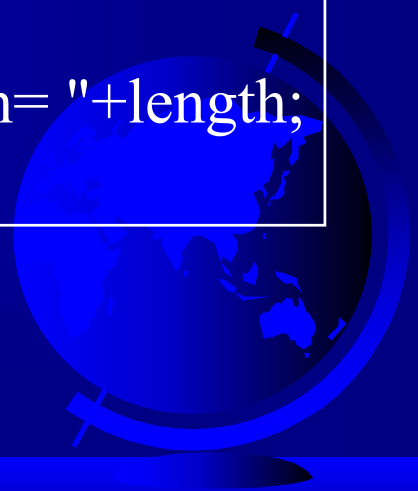
- "this" in Java is a reserved keyword
- It is used to indicate the current object or class invoked.
- For example: c1 is an object of Circle, c2 is an object, we are not sure what it belong to, we can use the previous Object instant method equals() to check out.
  - c1.equals(c2), if it return true the c1 and c2 are objects of the same class, otherwise, c2 is an object of different class as c1 is the same
- We can often use "this" to refer the current object, especially in GUI programming.

The `Object` Class

- The `toString()` method returns a string representation of the object.
- For example, one can provide an overriding method toString() to describe the object of Cylinder class

```
public String toString()
{
retunr "Cylinder radius= "+radius+"\n and length= "+length;
}
```

```
System.out.println(myCylinder.toString());
```



The `Object` Class

- The `clone()` method copy objects
  - It is used to make a copy of an object.
- What is the different between

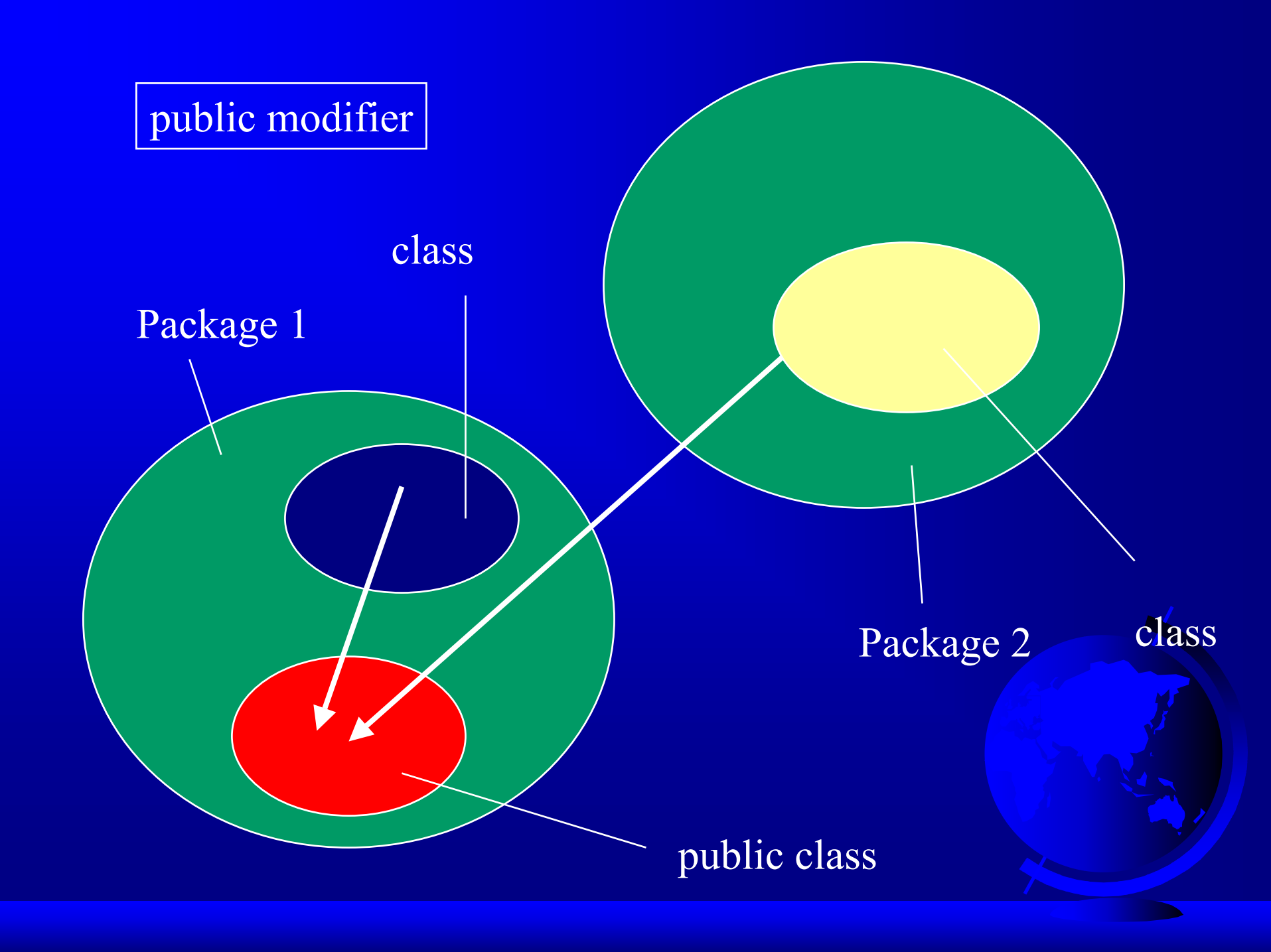
```
newObject=someObject;

and

newObject=someObject.clone();
```

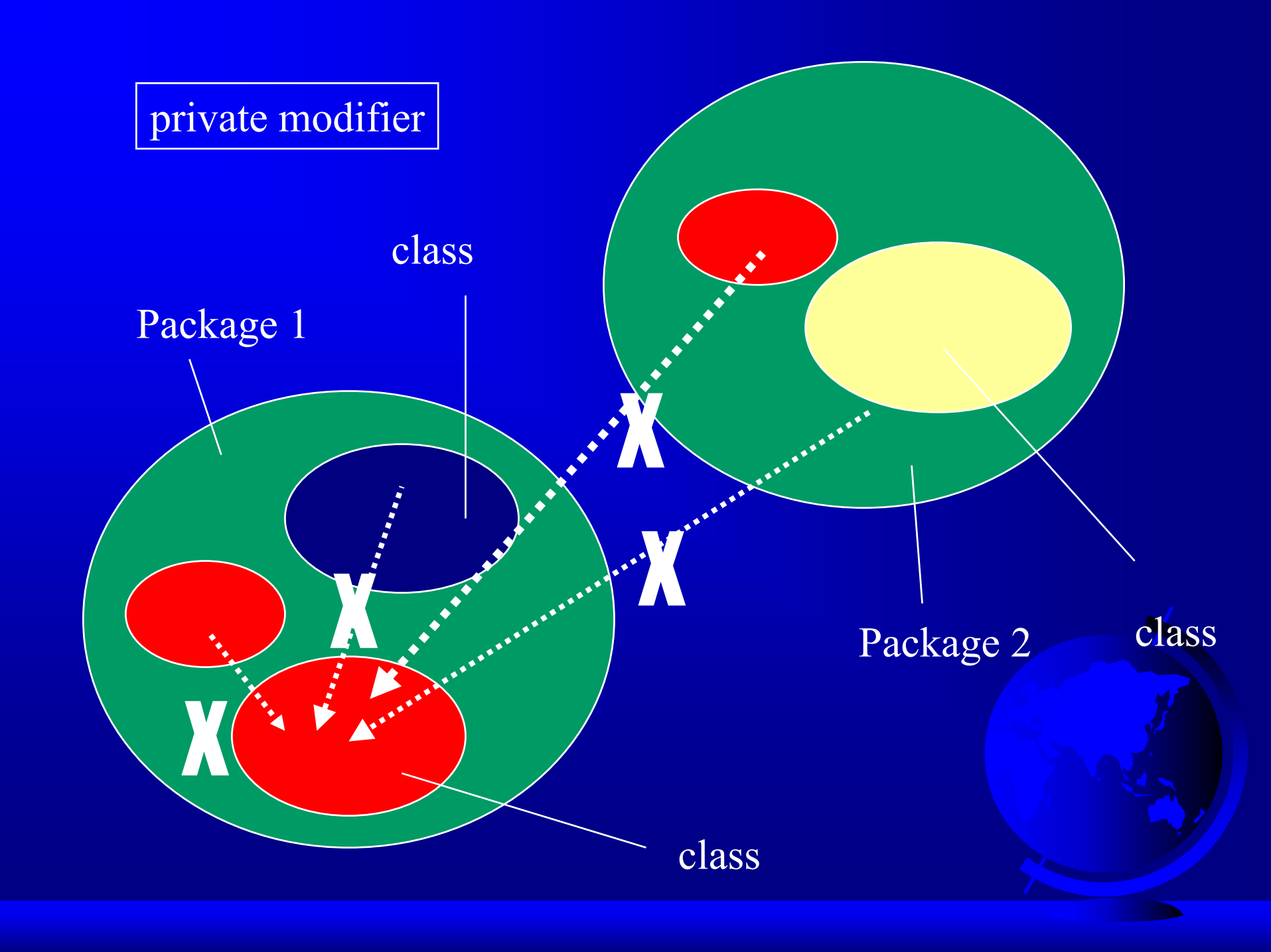
Assign the reference of someObject to newObject, not dublicate object

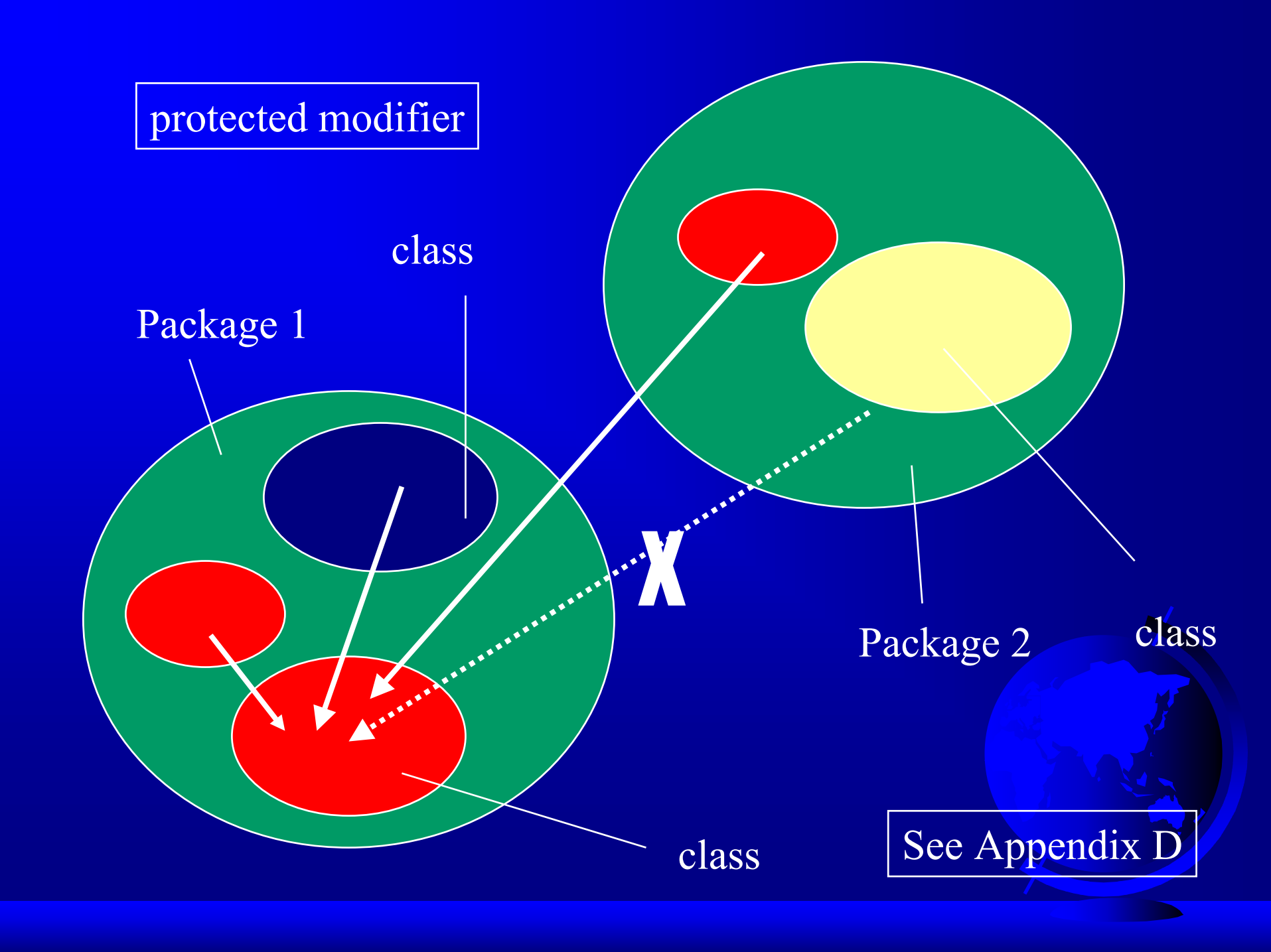
Creates a newObject with separate memory spcae and assigns the reference of the someObject to the newObject

The `protected` Modifier

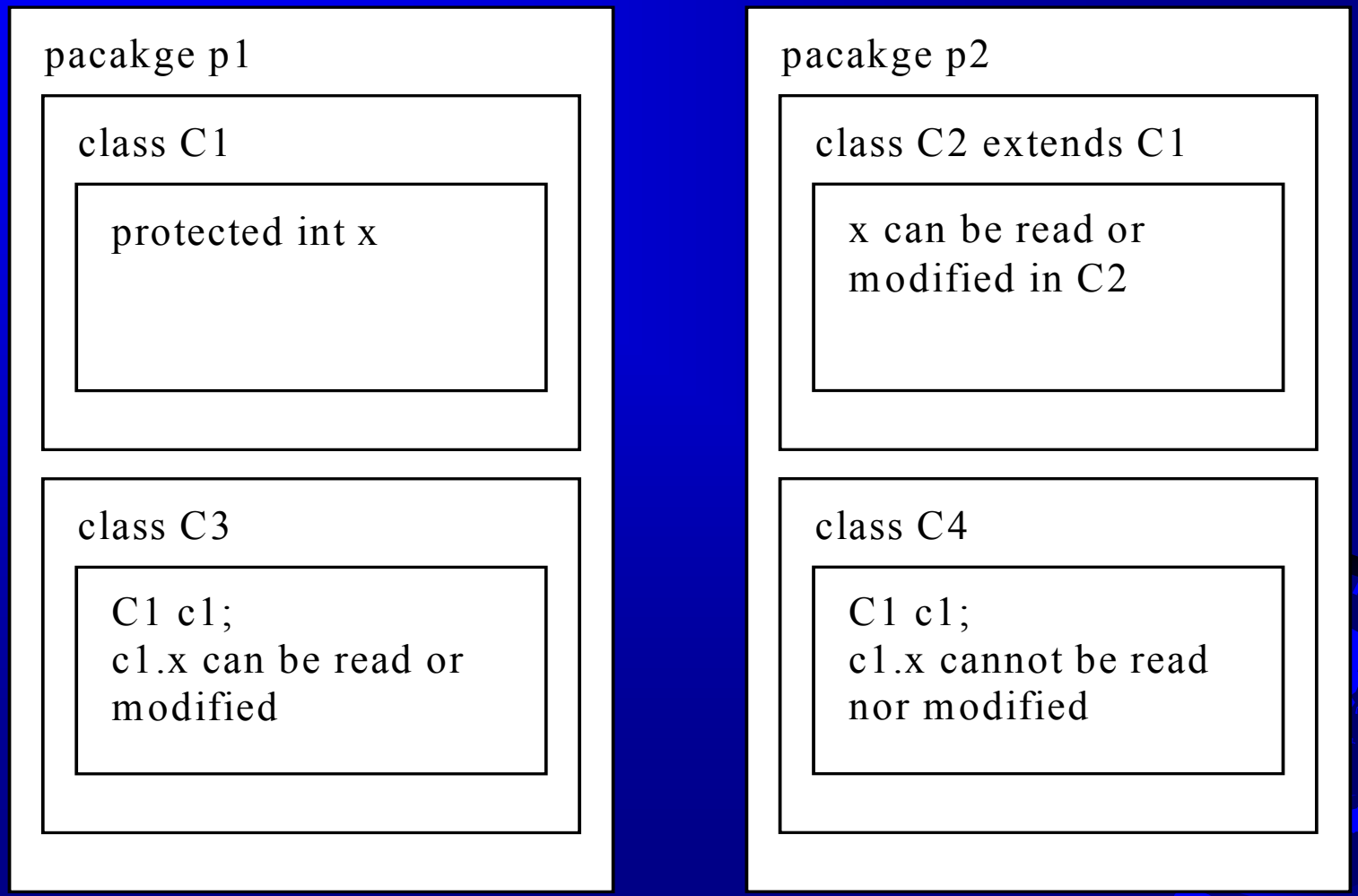
- Three additional modifiers: protected, final, abstract
- The `protected` modifier can be applied on data and methods in a class. A protected data or a protected method in a public class can be accessed by any class in the same package or its subclasses, even if the subclasses are in a different package.


public modifier
class
Package 1
Package 2
class
public class


private modifier
class
Package 1
Package 2
class
class


protected modifier
class
Package 1
X
Package 2
class
class
See Appendix D

The `protected` Modifier, cont.



Example of public, protected and private modifiers

Rectangle class

TestRectangle class

The `final` Modifier

- The `final` class cannot be extended:

```
final class Math
{...}
```

Math class can not be inherited

- The `final` variable is a constant:

```
final static double PI = 3.14159;
```

The value of PI can not be changed.

- The `final` method cannot be modified by its subclasses.

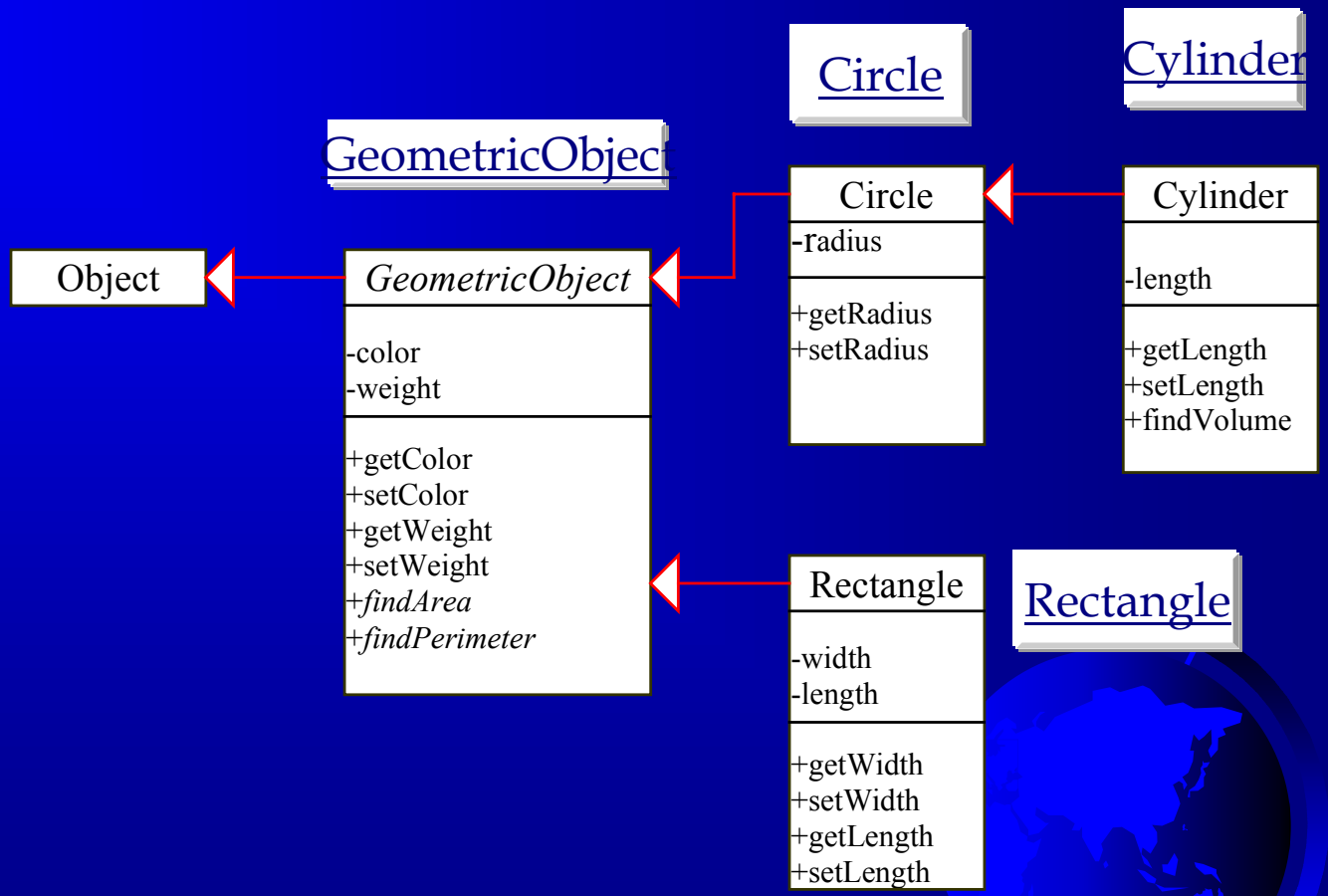
Can not be modified or overrided.

The `abstract` Modifier

- The `abstract` class
  - Cannot be instantiated
  - Should be extended and implemented in subclasses
- The `abstract` method
  - Method signature without implementation.
  - The implementation is dependent on the specific type of geometric object

Abstract Classes

Notation:
The abstract class name and the abstract method names are italicized in the UML.



Example: GeometricObject

```
// GeometricObject.java: The abstract GeometricObject class
public abstract class GeometricObject
{
  protected String color;
  protected double weight;
  // Default construct
  protected GeometricObject()
  {
    color = "white";
    weight = 1.0;
  }
  // Construct a geometric object
  protected GeometricObject(String color, double weight)
  {
    this.color = color;
    this.weight = weight;
  }
```

```
// Getter method for color
 public String getColor()
 {
  return color;
 }

 // Setter method for color
 public void setColor(String color)
 {
  this.color = color;
 }

 // Getter method for weight
 public double getWeight()
 {
  return weight;
 }
```

```
  // Setter method for weight
  public void setWeight(double weight)
  {
    this.weight = weight;
  }

  // Abstract method
  public abstract double findArea();

  // Abstract method
  public abstract double findPerimeter();
}
```

Abstract Classes

- Abstract class can not be used to create instant or object using "new"
- Abstract is designed to be inherited for deriving subclass
- An abstract class always contains at least one abstract method
- The abstract methods are implemented in the subclasses

```
// Circle.java: The circle class that extends GeometricObject
public class Circle extends GeometricObject
{
  protected double radius;

  // Default constructor
  public Circle()
  {
    this(1.0, "white", 1.0);
  }
  // Construct circle with specified radius
  public Circle(double radius)
  {
    super("white", 1.0);
    this.radius = radius;
  }
```

```
// Construct a circle with specified radius, weight, and color
 public Circle(double radius, String color, double weight)
 {
   super(color, weight);
   this.radius = radius;
 }

 // Getter method for radius
 public double getRadius()
 {
   return radius;
 }
 // Setter method for radius
 public void setRadius(double radius)
 {
   this.radius = radius;
 }
```

```
// Implement the findArea method defined in GeometricObject
  public double findArea()
  {
    return radius*radius*Math.PI;
  }

  // Implement the findPerimeter method defined in GeometricObject
  public double findPerimeter()
  {
    return 2*radius*Math.PI;
  }

  // Override the equals() method defined in the Object class
  public boolean equals(Circle circle)
  {
    return this.radius == circle.getRadius();
  }
```

```
// Override the toString() method defined in the Object class
  public String toString()
  {
    return "[Circle] radius = " + radius;
  }
}
```

```
// Rectangle.java: The Rectangle class that extends GeometricObject
public class Rectangle extends GeometricObject
{
  protected double width;
  protected double height;

  // Default constructor
  public Rectangle()
  {
    this(1.0, 1.0, "white", 1.0);
  }
  // Construct a rectangle with specified width and height
  public Rectangle(double width, double height)
  {
    this.width = width;
    this.height = height;
  }
```

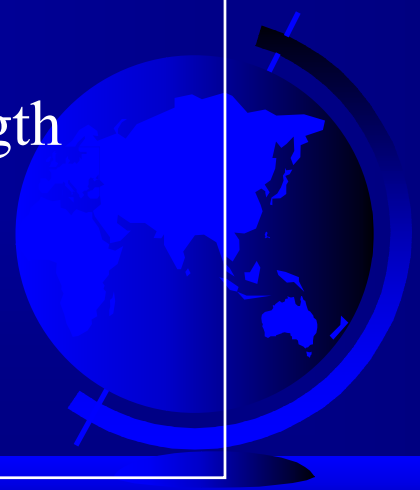
```
// Construct a rectangle with specified width,
// height, weight, and color
  public Rectangle(double width, double height,
    String color, double weight)
  {
    super(color, weight);
    this.width = width;
    this.height = height;
  }

  // Getter method for width
  public double getWidth()
  {
    return width;
  }
```

```
// Setter method for width
 public void setWidth(double width)
 {
  this.width = width;
 }

 // Getter method for height
 public double getHeight()
 {
  return height;
 }

 // Setter method for height
 public void setHeight(double height)
 {
  this.height = height;
 }
```

```
// Implement the findArea method in GeometricObject
public double findArea()
{
  return width*height;
}

// Implement the findPerimeter method in GeometricObject
public double findPerimeter()
{
  return 2*(width + height);
}
```

```
// Override the equals() method defined in the Object class
  public boolean equals(Rectangle rectangle)
  {
    return (width == rectangle.getWidth()) &&
      (height ==  rectangle.getHeight());
  }

  // Override the toString() method defined in the Object class
  public String toString()
  {
    return "[Rectangle] width = " + width + " and height = " + height;
  }
}
```

```
// Cylinder.java: The new cylinder class that extends the circle
// class
class Cylinder extends Circle
{
  private double length;

  // Default constructor
  public Cylinder()
  {
    super();
    length = 1.0;
  }
  // Construct a cylinder with specified radius, and length
  public Cylinder(double radius, double length)
  {
    this(radius, "white", 1.0, length);
  }
```



```
// Construct a cylinder with specified radius, weight, color, and
// length
public Cylinder(double radius,
  String color, double weight, double length)
{
  super(radius, color, weight);
  this.length = length;
}

// Getter method for length
public double getLength()
{
  return length;
}
```

```
// Setter method for length
 public void setLength(double length)
 {
  this.length = length;
 }

 // Find cylinder surface area
 public double findArea()
 {
  return 2*super.findArea()+(2*getRadius()*Math.PI)*length;
 }

 // Find cylinder volume
 public double findVolume()
 {
  return super.findArea()*length;
 }
```

```
// Override the equals() method defined in the Object class
public boolean equals(Cylinder cylinder)
{
  return (this.radius == cylinder.getRadius()) &&
    (this.length == cylinder.getLength());
}

// Override the toString() method defined in the Object class
public String toString()
{
  return "[Cylinder] radius = " + radius + " and length "
    + length;
}
}
```

Polymorphism and Dynamic Binding

Polymorphism refers to the ability to determine at runtime which code to run, given multiple methods with the same name but different operations in the same class or in different classes. This ability is also known as *dynamic binding*.

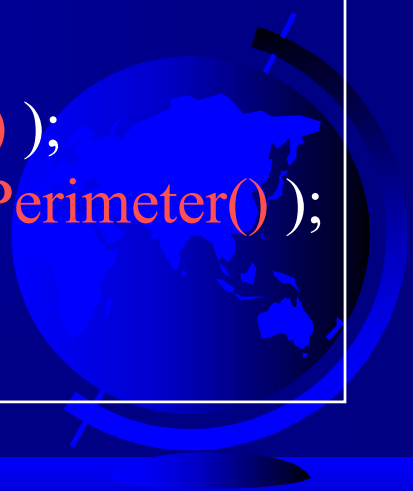
Example 6.3
Testing Polymorphism

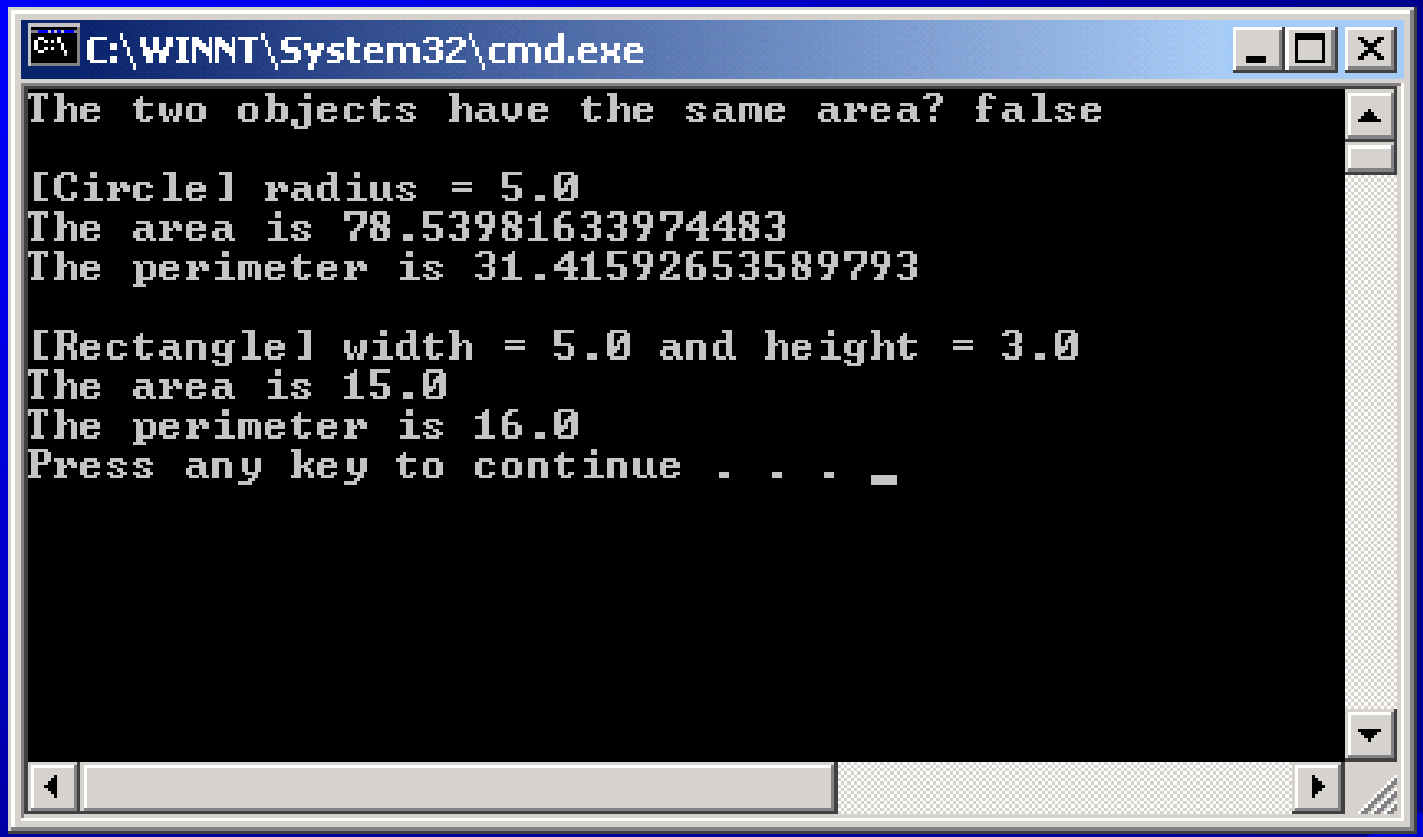
- Objective: This example creates two geometric objects: a circle, and a rectangle, invokes the equalArea method to check if the two objects have equal area, and invokes the displayGeometricObject method to display the objects.

```
// TestPolymorhism.java: Demonstrate using the max method
public class TestPolymorphism
{
  // Main method
  public static void main(String[] args)
  {
    // Declare and initialize two geometric objects
    GeometricObject geoObject1 = new Circle(5);
    GeometricObject geoObject2 = new Rectangle(5, 3);

    System.out.println("The two objects have the same area? " +
      equalArea(geoObject1, geoObject2) );
    // Display circle
    displayGeometricObject(geoObject1);
    // Display rectangle
    displayGeometricObject(geoObject2);
  }
```

Implicit casting

```
// A method for comparing the areas of two geometric objects
static boolean equalArea(GeometricObject object1,
        GeometricObject object2)
{
  return object1.findArea() == object2.findArea();
}

// A method for displaying a geometric object
static void displayGeometricObject(GeometricObject object)
{
  System.out.println();
  System.out.println(object.toString());
  System.out.println("The area is " + object.findArea() );
  System.out.println("The perimeter is " + object.findPerimeter() );
}
}
```




C:\WINNT\System32\cmd.exe
The two objects have the same area? false
[Circle] radius = 5.0
The area is 78.53981633974483
The perimeter is 31.41592653589793
[Rectangle] width = 5.0 and height = 3.0
The area is 15.0
The perimeter is 16.0
Press any key to continue . . .

Casting Objects

It is always possible to convert a subclass to a superclass. That is because an instance of a superclass is also an instance of its superclass. For this reason, explicit casting can be omitted. That is called implicit casting. For example,

```
Circle myCircle = myCylinder
```

is equivalent to

```
Circle myCircle = (Circle)myCylinder;
```

Casting from Superclass to Subclass

Explicit casting must be used when casting an object from a superclass to a subclass. This type of casting may not always succeed. Make sure the explicit casting is only valid to converting from superclass to subclass. Otherwise, a runtime exception occurs.

```
Cylinder myCylinder = (Cylinder)myCircle;
```

The `instanceof` Operator

Use the `instanceof` operator to test whether an object is an instance of a class:

```
Circle myCircle = new Circle();

if (myCircle instanceof Cylinder)
{
  Cylinder myCylinder = (Cylinder)myCircle;
  ...
}
```

Example 6.4
Casting Objects

This example creates two geometric objects: a circle, and a cylinder, invokes the displayGeometricObject method to display the objects. The displayGeometricObject displays the area and perimeter if the object is a circle, and displays area and volume if the object is a cylinder.

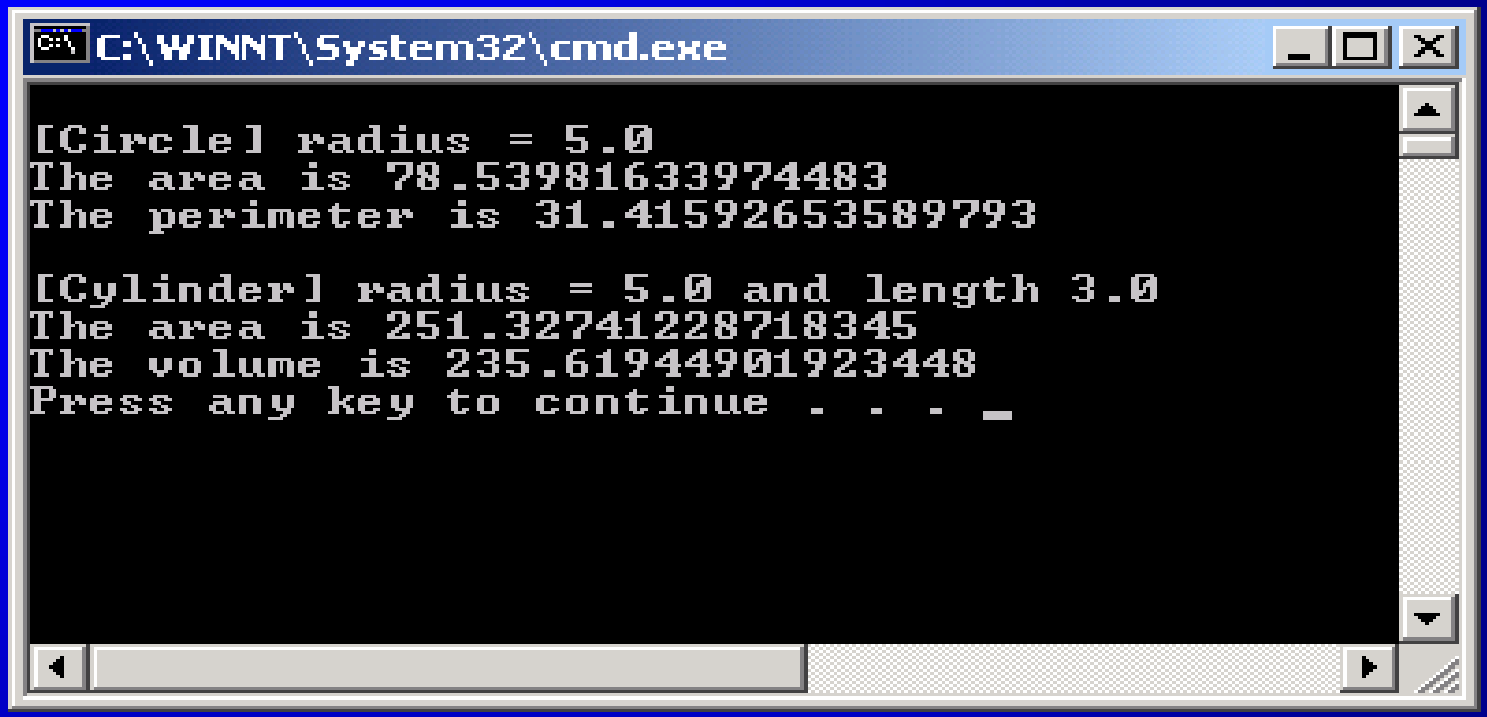
```
// TestCasting.java: Demonstrate casting objects
public class TestCasting
{
  // Main method
  public static void main(String[] args)
  {
    // Declare and initialize two geometric objects
    GeometricObject geoObject1 = new Circle(5);
    GeometricObject geoObject2 = new Cylinder(5, 3);

    // Display circle
    displayGeometricObject(geoObject1);

    // Display cylinder
    displayGeometricObject(geoObject2);
  }
```

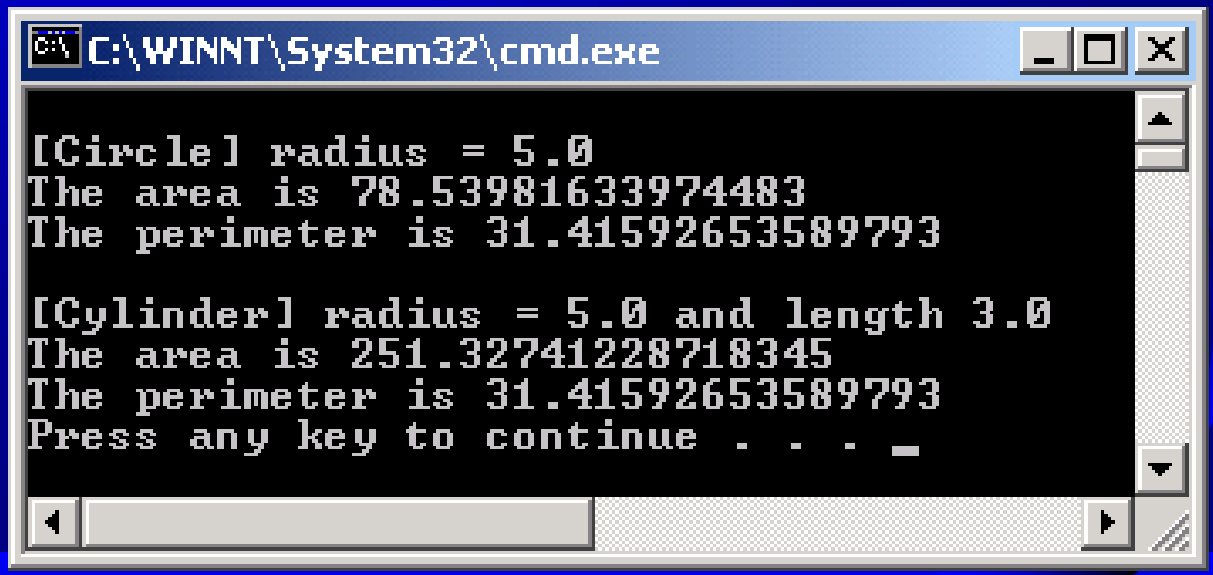
```
// A method for displaying a geometric object
 static void displayGeometricObject(GeometricObject object)
 {
   System.out.println();
   System.out.println(object.toString());
   if (object instanceof Cylinder)
   {
     System.out.println("The area is " +
       ((Cylinder)object).findArea());
     System.out.println("The volume is " +
       ((Cylinder)object).findVolume());
   }
```

```
else if (object instanceof Circle)
    {
      System.out.println("The area is " + object.findArea());
      System.out.println("The perimeter is " +
        object.findPerimeter());
    }
  }
}
```

Note:

1. No need to cast object of GeometricObject to Circle.
2. Need to cast the object of GeometricObject to Cylinder.
3. The order of if is very important.


C:\WINNT\System32\cmd.exe
[Circle] radius = 5.0
The area is 78.53981633974483
The perimeter is 31.4159265358979 3
[Cylinder] radius = 5.0 and length 3.0
The area is 251.3274122871834 5
The volume is 235.6194490192344 8
Press any key to continue . . .


C:\WINNT\System32\cmd.exe
[Circle] radius = 5.0
The area is 78.53981633974483
The perimeter is 31.4159265358979 3
[Cylinder] radius = 5.0 and length 3.0
The area is 251.3274122871834 5
The perimeter is 31.4159265358979 3
Press any key to continue . . .

Interfaces

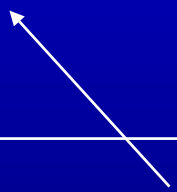
- What Is an Interface?
  - Special class, which is introduced for multiple inheritance
  - Java only allows to inherit one superclass
  - But Java can inherit many interface

  - Interface has only constants and abstract method
  - The difference between interface and abstract class is
    - Abstract class can contains constants and abstract methods, as well as variables and concrete methods

Interfaces

- Creating an Interface

```
modifier interface InterfaceName
{
//constants declarations;

//abstract methods signatures;

}
```



abstract method without implementation

Interfaces

- As with an abstract class, you can not create instance for interface using "new"
- The way to use interface is the same as to use a abstract class

- Implementing an Interface
  - Use reserved word "impliments" insetad of "extends"
- What is Marker Interface?

Creating an Interface

```
modifier interface InterfaceName
{
   constants declarations;
   methods signatures;
}
```

Example of Creating an Interface

Design a generic method to find out the maximum of two objects

```
// This interface is already defined in
// java.lang package
public interface Comparable
{
  public static int compareTo(Object o);   //abstract method
}
```

Generic `max` Method

```
// Max.java: Find a maximum object
public class Max
{
  // Return the maximum between two objects
  public static Comparable max(Comparable o1, Comparable o2)
  {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

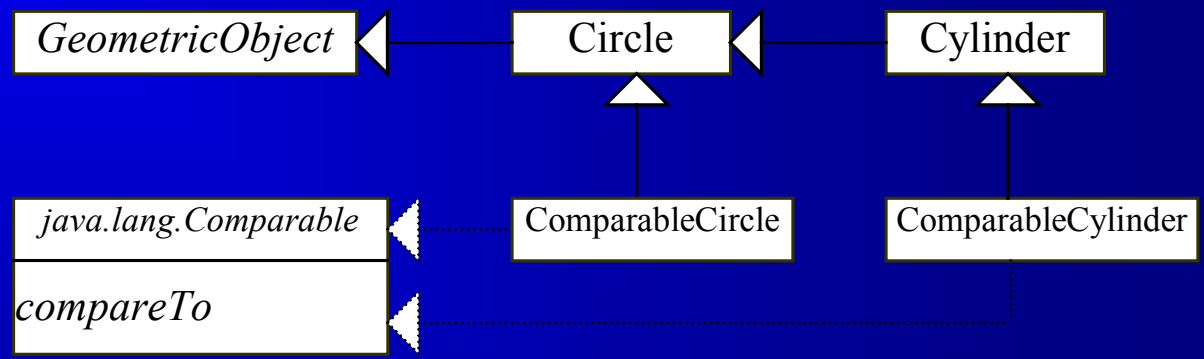
Abstract method

Example 6.5
Using Interfaces

- Objective: Use the max method to find a find the maximum circle between two circles and a maximum cylinder between two cylinders.

Example 6.5, cont.

Notation:
The interface class name and the method names are italicized. The dashed lines and dashed hollow triangles are used to point to the interface.



```
// TestInterface.java: Use the Comparable interface
// and the generic max method to find max objects
public class TestInterface
{
  // Main method
  public static void main(String[] args)
  {
    // Create two comarable circles
    ComparableCircle circle1 = new ComparableCircle(5);
    ComparableCircle circle2 = new ComparableCircle(4);

    // Display the max circle
    Comparable circle = Max.max(circle1, circle2);
    System.out.println("The max circle's radius is " +
      ((Circle)circle).getRadius());
    System.out.println(circle);
```

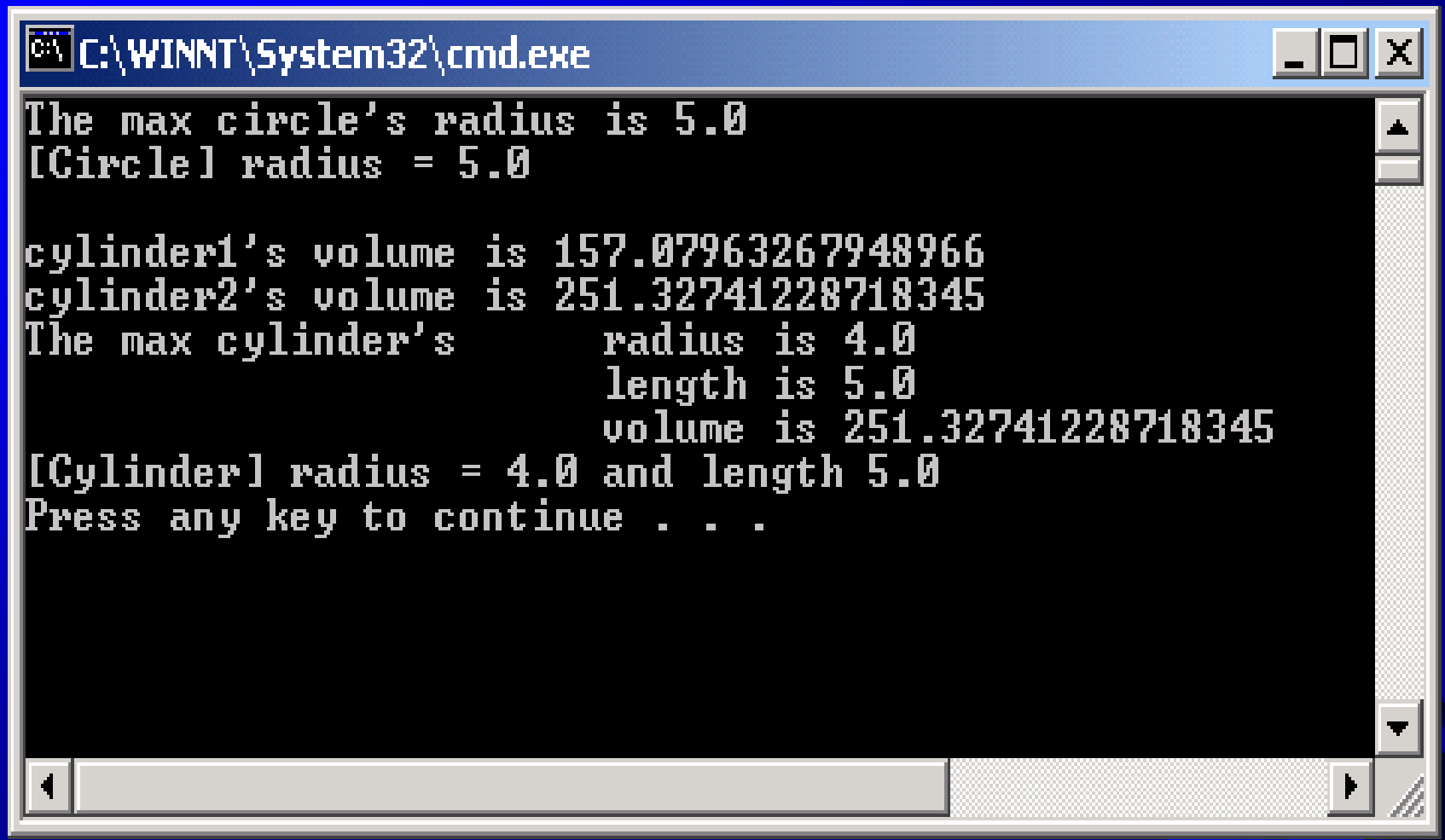
```
// Create two comarable cylinders
  ComparableCylinder cylinder1 = new ComparableCylinder(5, 2);
  ComparableCylinder cylinder2 = new ComparableCylinder(4, 5);

  // Display the max cylinder
  Comparable cylinder = Max.max(cylinder1, cylinder2);
  System.out.println();
  System.out.println("cylinder1's volume is " +
    cylinder1.findVolume());
  System.out.println("cylinder2's volume is " +
    cylinder2.findVolume());
  System.out.println("The max cylinder's \tradius is " +
    ((Cylinder)cylinder).getRadius() + "\n\t\t\tlength is " +
    ((Cylinder)cylinder).getLength() + "\n\t\t\tvolume is " +
    ((Cylinder)cylinder).findVolume());
  System.out.println(cylinder);
 }
}
```

```
// ComparableCircle is a subclass of Circle, which implements the
// Comparable interface
class ComparableCircle extends Circle implements Comparable
{
  // Construct a CompareCircle with specified radius
  public ComparableCircle(double r)
  {
    super(r);
  }
```

```
// Implement the compareTo method defined in Comparable
  public int compareTo(Object o)
  {
    if (getRadius() > ((Circle)o).getRadius())
      return 1;
    else if (getRadius() < ((Circle)o).getRadius())
      return -1;
    else
      return 0;
  }
}
```

```
// ComparableCylinder is a subclass of Cylinder,
//  which implements the CompareObject interface

class ComparableCylinder extends Cylinder implements Comparable
{
  // Construct a CompareCylinder with radius and length
  ComparableCylinder(double r, double l)
  {
    super(r, l);
  }
```

```
  // Implement the compareTo method defined in
// Comparable interface
  public int compareTo(Object o)
  {
    if (findVolume() > ((Cylinder)o).findVolume())
      return 1;
    else if (findVolume() < ((Cylinder)o).findVolume())
      return -1;
    else
      return 0;
  }
}
```


C:\WINNT\System32\cmd.exe
The max circle's radius is 5.0
[Circle] radius = 5.0
cylinder1's volume is 157.07963267948966
cylinder2's volume is 251.32741228718345
The max cylinder's radius is 4.0
length is 5.0
volume is 251.32741228718345
[Cylinder] radius = 4.0 and length 5.0
Press any key to continue . . .

Interfaces vs. Abstract Classes

In an interface, the data must be constants; an abstract class can have all types of data.

Each method in an interface has only a signature without implementation; an abstract class can have concrete methods. An abstract class must contain at least one abstract method or inherit from another abstract method.

Interfaces vs. Abstract Classes, cont.

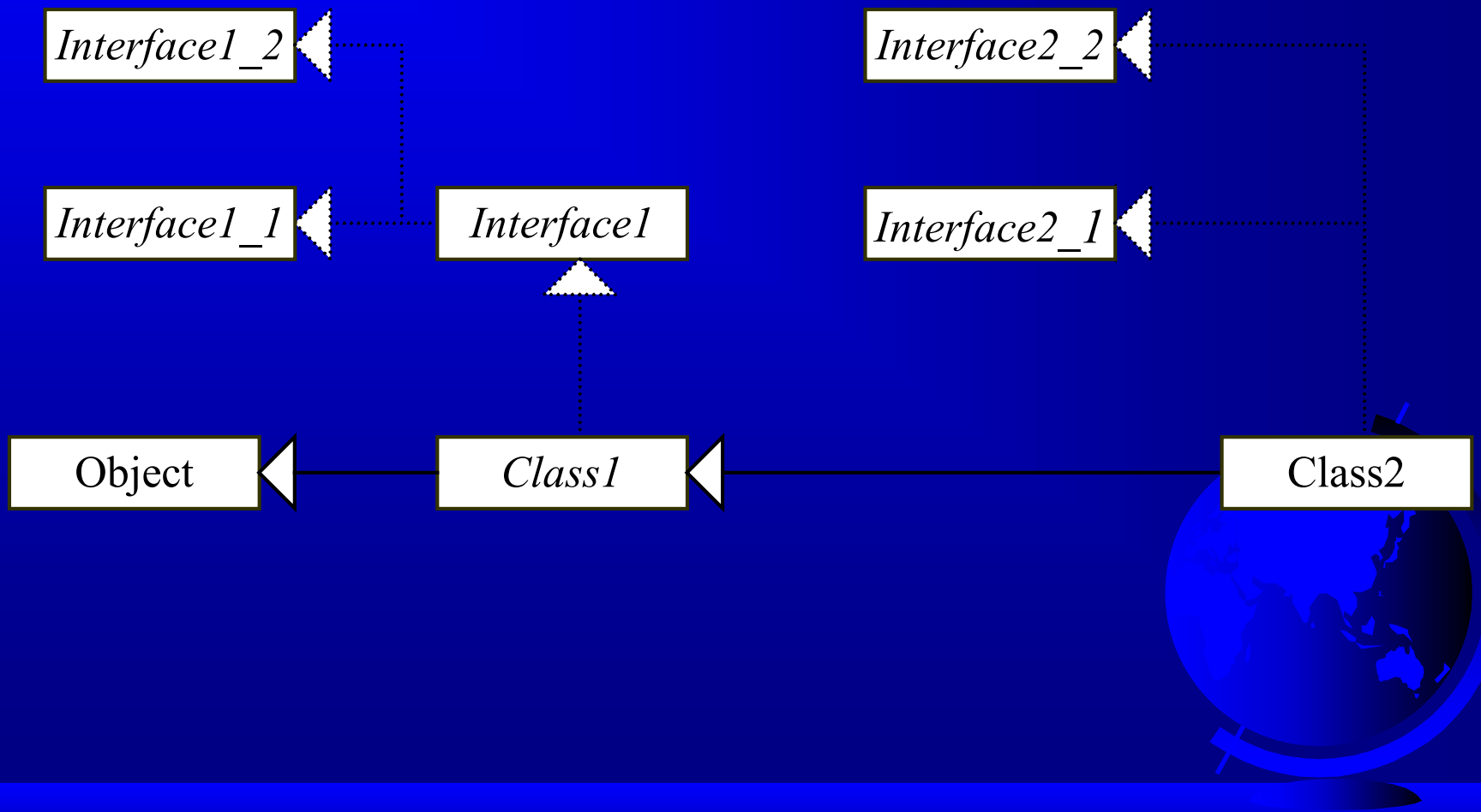
Since all the methods defined in an interface are abstract methods, Java does not require you to put the abstract modifier in the methods in an interface, but you must put the abstract modifier before an abstract method in an abstract class.

Interface can only extends another interface.

Class and extends its superclass and implments multiple interfaces

No root interface for interface.

Interfaces vs. Abstract Classes, cont.



The `Cloneable` Interfaces

Marker Interface: An empty interface.

A marker interface does not contain constants or methods, but it has a special meaning to the Java system. The Java system requires a class to implement the Cloneable interface to become cloneable.

```
public interface Cloneable
{
}
```

Empty body

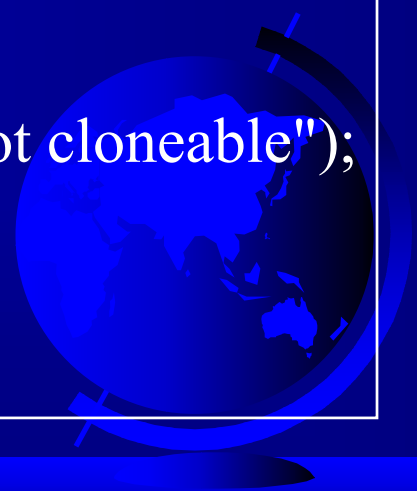
Example 6.6
Cloning Objects

- Objective: uses the Cloneable interface to mark classes cloneable and uses the clone method to copy objects.

```
// TestCloneable.java: Use the TestCloneable interface
// to enable cloning
public class TestCloneable
{
  // Main method
  public static void main(String[] args)
  {
    // Declare and create an instance of CloneableCircle
    CloneableCircle c1 = new CloneableCircle(5);
    CloneableCircle c2 = (CloneableCircle)c1.clone();

    System.out.println("After copying c1 to circl2");
```

```
// Check if two variables point to the same object
  if (c1 == c2)
    System.out.println("c1 and c2 reference to the same object");
  else
    System.out.println("c1 and c2 don't point to the same object");

  // Check if two objects are of identical contents
  if (c1.equals(c2))
    System.out.println("c1 and c2 have the same contents");
  else
    System.out.println("c1 and c2 don't have the same contents");

  // Modify c1's radius, name
  c1.setRadius(10);
  c1.getCreator().setFirstname("Michael");
  c1.getCreator().setMi("Z");
```

```
// Display c1 and c2
   System.out.println("\nAfter modifying c1");
   System.out.println("c1 " + c1);
   System.out.println("c2 " + c2);

   System.out.println();
   if (c1 instanceof Cloneable)
   {
    System.out.println("A CloneableCircle objec is cloneable");
   }
   else
   {
    System.out.println("A CloneableCircle objec is not cloneable");
   }
```



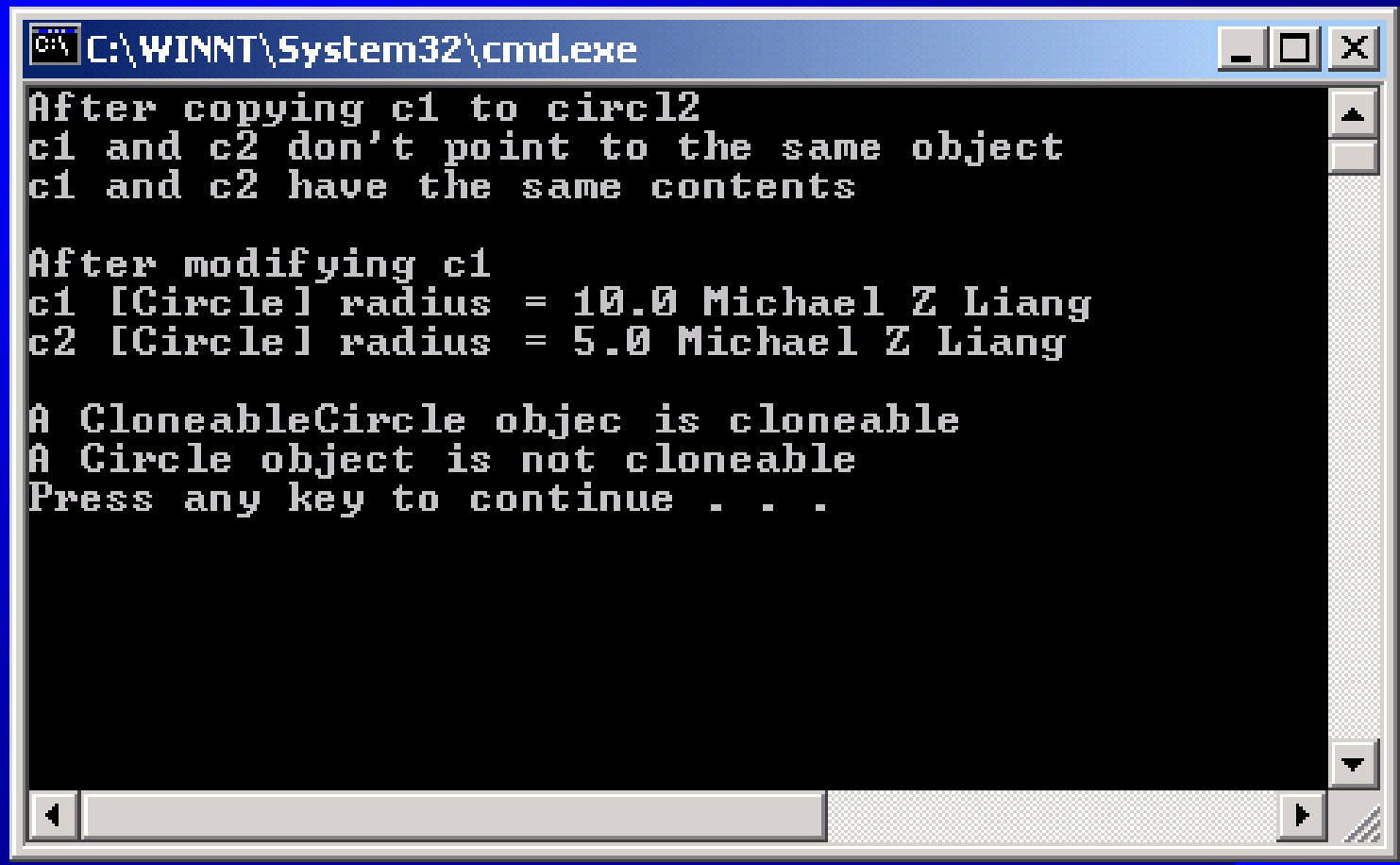
```
// Check if a Circle object is cloneable
   Circle c = new Circle();
   if (c instanceof Cloneable)
    {
     System.out.println("A Circle object is cloneable");
    }
   else
    {
     System.out.println("A Circle object is not cloneable");
    }
  }
}
```

```
// CloneableCircle is a subclass of Circle, which implements the
// Cloneable interface
class CloneableCircle extends Circle implements Cloneable
{
  // Store the creator of the object
  private Name creator = new Name("Yong", "D", "Liang");

  // Construct a CloneableCircle with specified radius
  public CloneableCircle(double radius)
  {
    super(radius);
  }
```

```
// Getter method for creator
 public Name getCreator()
 {
   return creator;
 }

 // Setter method for creator
 public void setCreator(Name name)
 {
   creator = name;
 }
```

```
// Override the protected clone method defined in the Object class
 public Object clone()
 {
  try
  {
   return super.clone();
  }
  catch (CloneNotSupportedException ex)
  {
   return null;
  }
 }
```

```
// Override the toString method defined in the Object class
  public String toString()
  {
    return super.toString() + " " + creator.getFullname();
  }
}
```


C:\WINNT\System32\cmd.exe
After copying c1 to circl2
c1 and c2 don't point to the same object
c1 and c2 have the same contents
After modifying c1
c1 [Circle] radius = 10.0 Michael Z Liang
c2 [Circle] radius = 5.0 Michael Z Liang
A CloneableCircle objec is cloneable
A Circle object is not cloneable
Press any key to continue . . .

Inner Classes

Inner class: A class is a member of another class.

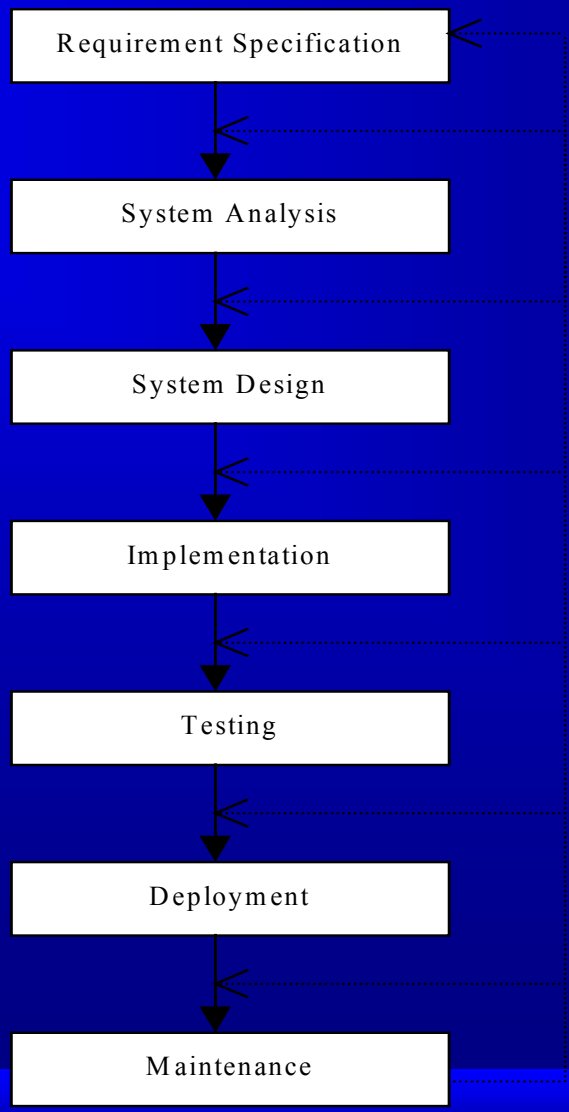
Advantages: In some applications, you can use an inner class to make programs simple.

- An inner class can reference the data and methods defined in the outer class in which it nests, so you do not need to pass the reference of the outer class to the constructor of the inner class.

Inner Classes (cont.)

- Inner classes can make programs simple and concise. As you see, the new class is shorter and leaner. Many Java development tools use inner classes to generate adapters for handling events. Event-driven programming is introduced in Chapter 8, "Getting Started with Graphics Programming."
- An inner class is only for supporting the work of its containing outer class, and it cannot be used by other classes.

Software Development Process



Class Design Guidelines

- Hide private data and private methods.
- A property that is shared by all the instances of the class should be declared as a class property.
- Provide a public default constructor and override the equals method and the toString method defined in the Object class whenever possible.

Class Design Guidelines, cont.

- Choose informative names and follow consistent styles.
- A class should describe a single entity or a set of similar operations.
- Group common data fields and operations shared by other classes.