

Object-Oriented Programming with Java **Inheritance**



Quan Thanh Tho, Ph.D.

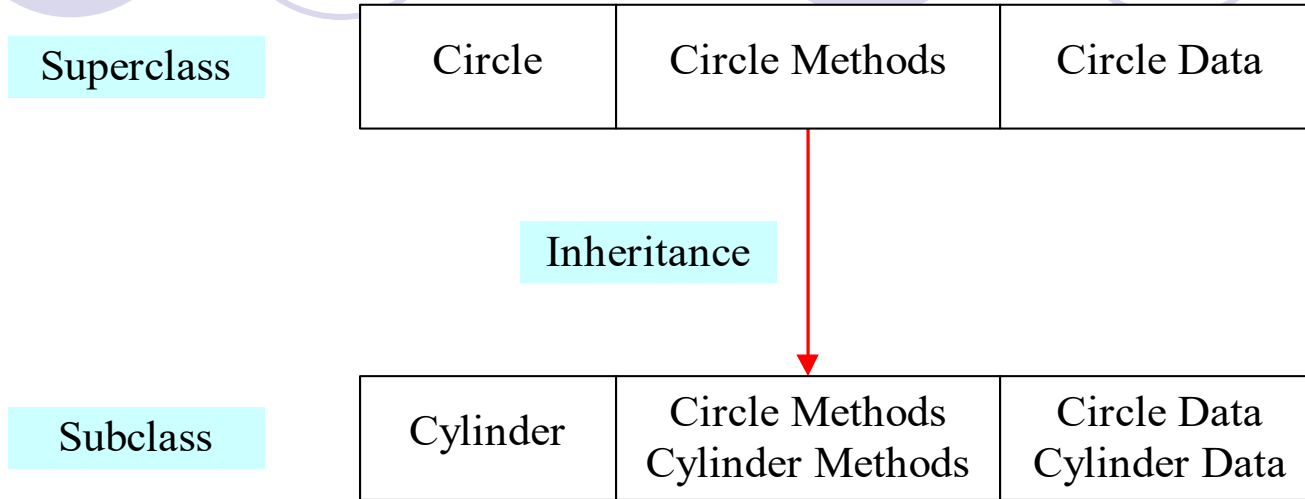
CSE – HCMUT

qttho@cse.hcmut.edu.vn

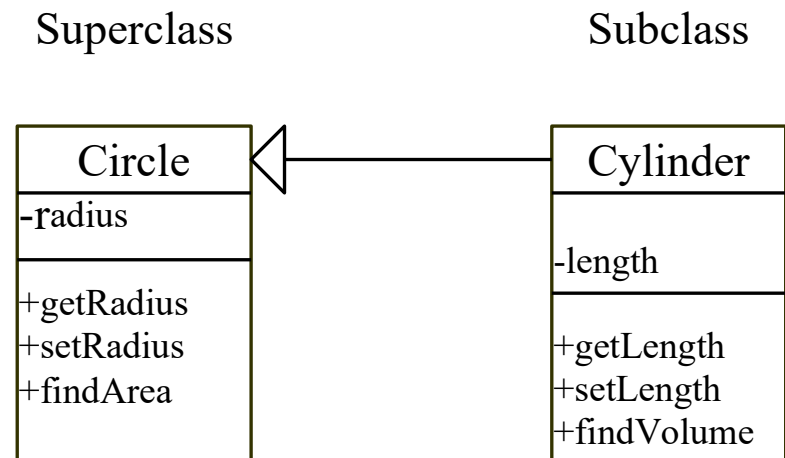
Class Inheritance and Interfaces

- ◆ Superclasses and Subclasses
- ◆ Keywords: `super`
- ◆ Overriding methods
- ◆ The `Object` Class
- ◆ Modifiers: `protected`, `final` and `abstract`
- ◆ Abstract classes
- ◆ Polymorphism and Object Casting
- ◆ Interfaces
- ◆ Inner Classes

Superclasses and Subclasses



UML Diagram



Creating a Subclass



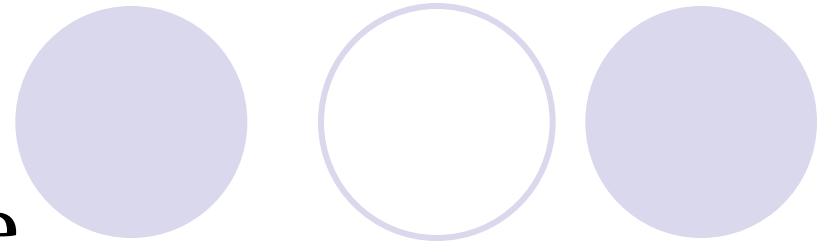
Creating a subclass extends properties and methods from the superclass. You can also:

- ◆ Add new properties
- ◆ Add new methods
- ◆ Override the methods of the superclass

Cylinder Class

Example 8.1

Testing Inheritance



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

[TestCylinder](#)

Run

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 constructor
- To call a superclass method



Example 8.2

Overriding Methods in the Superclass

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

[Cylinder](#)

[TestOverrideMethods](#)

Run

The Object Class



- The `Object` class is the root of all Java classes.
- The `equals()` method compares the contents of two objects.
- The `toString()` method returns a string representation of the object.
- The `clone()` method copy objects

The Object Class, cont.



The `equals()` method compares the contents of two objects. The default implementation of the `equals` method in the `Object` class is as follows:

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

The Object Class, cont.

- The `toString()` method returns a string representation of the object. The default implementation returns a string consisting of a class name of which the object is an instance, the at sign (`@`), and a number representing this object.

The `clone()` method copy objects

The Object Class, cont.

To create a new object with separate memory space, you need to use the `clone()` method, as follows:

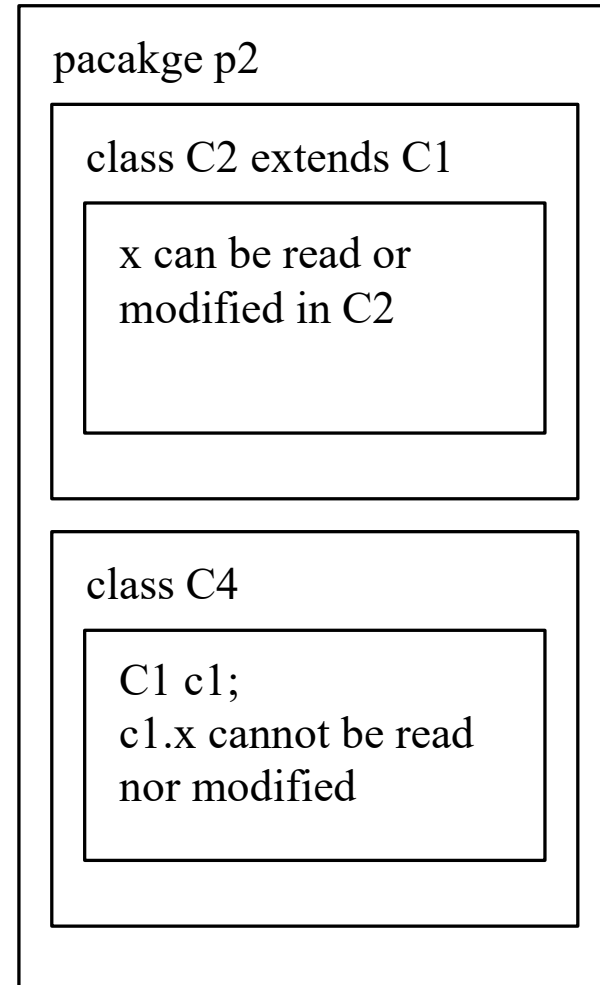
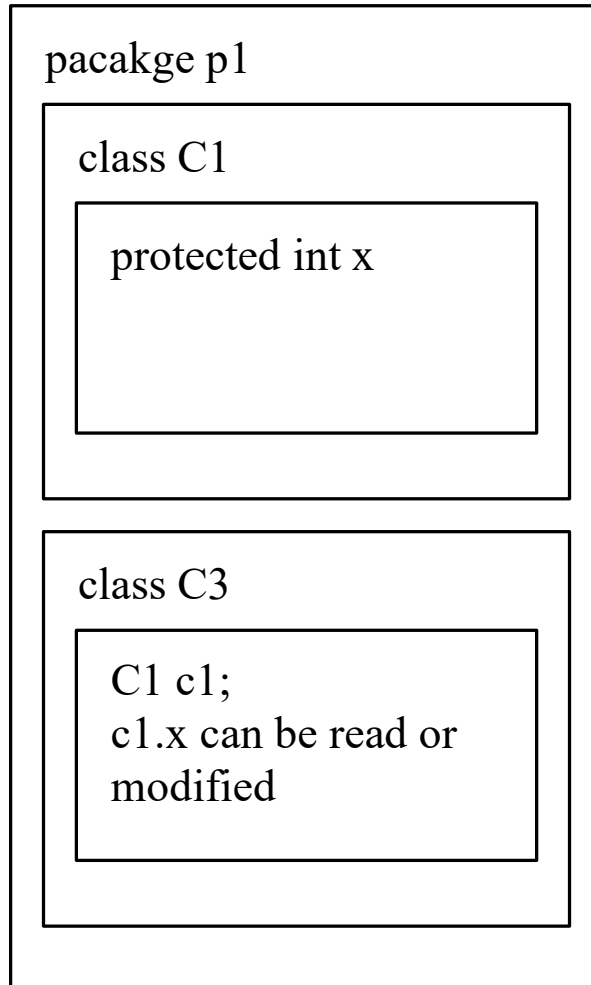
```
newObject = someObject.clone();
```

NOTE: Not all objects can be cloned. For an object to be cloneable, its class must implement the `java.lang.Cloneable` interface. Interfaces are introduced in the section "Interfaces," in this chapter

The `protected` Modifier

- 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.

The protected Modifier, cont.



The final Modifier

- The final class cannot be extended:

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

- The final variable is a constant:

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

- The final method cannot be modified by its subclasses.



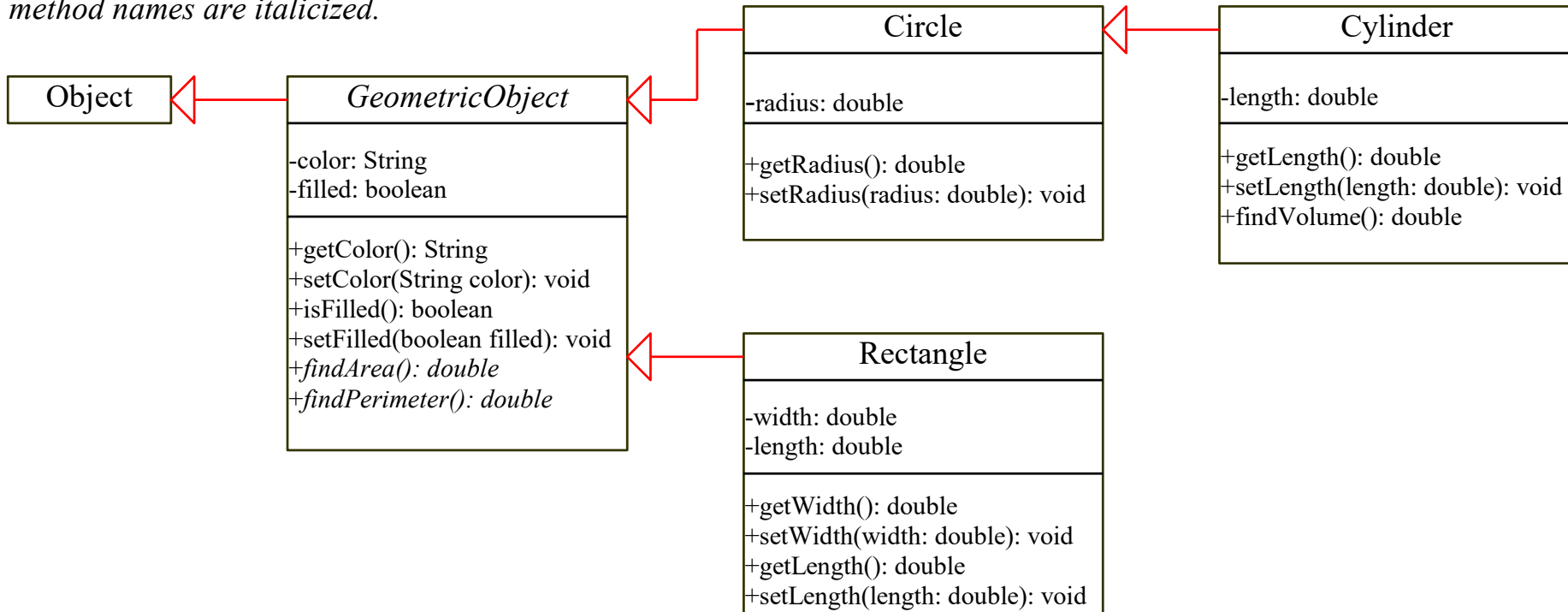
The abstract Modifier

- The abstract class
 - Cannot be instantiated
 - Should be extended and implemented in subclasses
- The abstract method
 - Method signature without implementation

Abstract Classes

UML Notation:

The abstract class name and the abstract method names are italicized.



[GeometricObject](#)

[Circle](#)

[Cylinder](#)

[Rectangle](#)

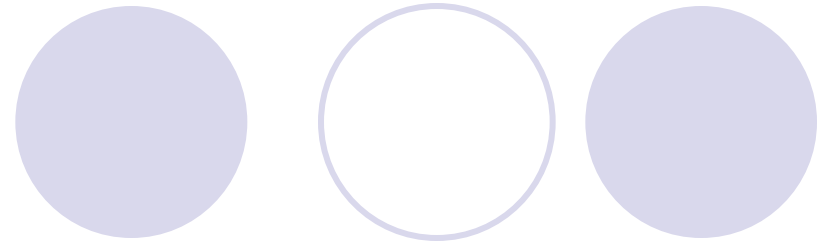


Polymorphism and Dynamic Binding

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 8.3

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.

[TestPolymorphism](#)

Run

Casting Objects

It is always possible to convert a subclass to a superclass. For this reason, explicit casting can be omitted. For example,

```
Circle myCircle = myCylinder
```

is equivalent to

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

Casting from Superclass to Subclass

Five circles are arranged horizontally at the top of the slide. From left to right, they are: a solid light purple circle, an outlined light purple circle, a solid light purple circle, an outlined light purple circle, and a solid light purple circle.

Explicit casting must be used when casting an object from a superclass to a subclass. This type of casting may not always succeed.

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

The instanceof Operator

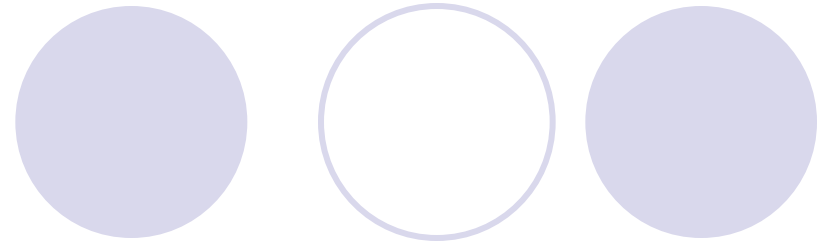
Use the instanceof operator to test whether an object is an instance of a class:

```
Circle myCircle = new Circle();

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

Example 8.4

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](#)

Run

Interfaces

A decorative header element consisting of six circles arranged in a horizontal row. The first two circles are partially overlapping the title 'Interfaces'. The circles alternate in style: solid light purple, hollow light purple outline, solid light purple, hollow light purple outline, solid light purple, and hollow light purple outline.

- What Is an Interface?
- Creating an Interface
- Implementing an Interface
- What is Marker Interface?

Creating an Interface

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


Example of Creating an Interface

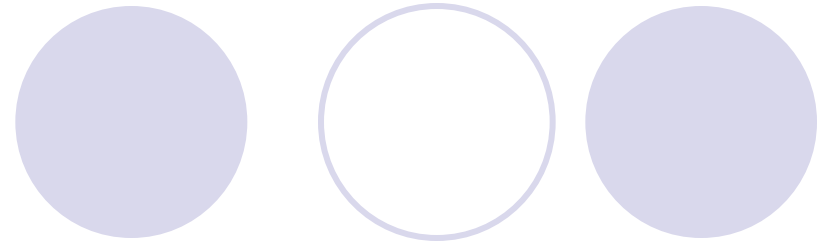
```
// This interface is defined in  
// java.lang package  
public interface Comparable  
{  
    public int compareTo(Object o);  
}
```

Generic max Method

```
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;
    }
}
```

Example 8.5

Using Interfaces



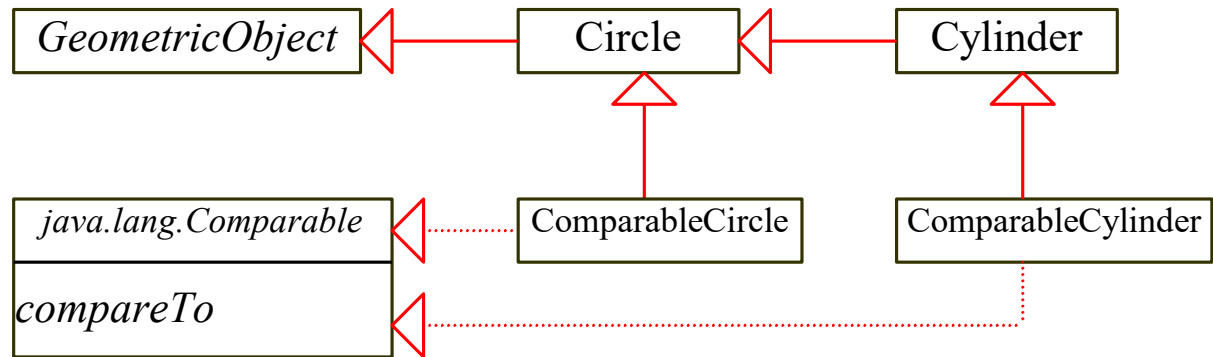
- Objective: Use the max method to find a
find the maximum circle between two
circles and a maximum cylinder between
two cylinders.

Example 8.5, cont.

Notation:

The interface name and the method names are italicized.

The dashed lines and hollow triangles are used to point to the interface.



TestInterface


Run

Interfaces vs. Abstract Classes



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

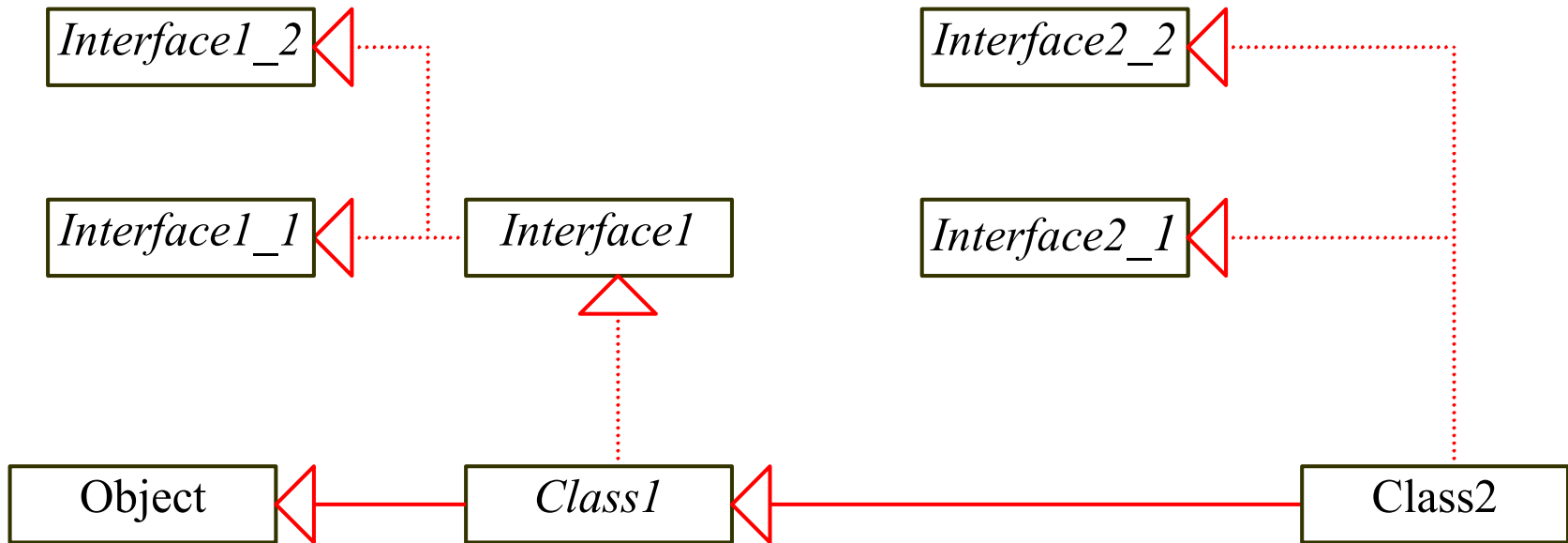
Each method in an interface has only a signature without implementation; an abstract class can have concrete methods. An abstract class must contain at least one abstract method or inherit from another abstract method.



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.

Interfaces vs. Abstract Classes, cont.



The Cloneable Interfaces

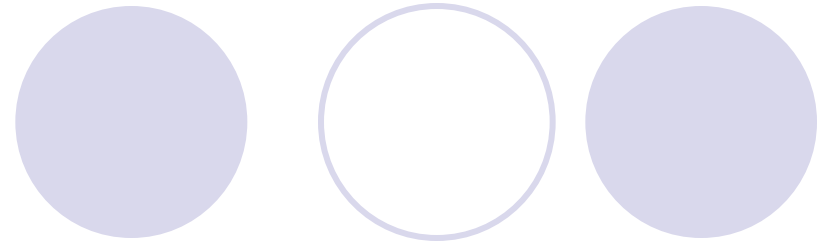
Marker Interface: An empty interface.

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
{
}
```


Example 8.6

Cloning Objects



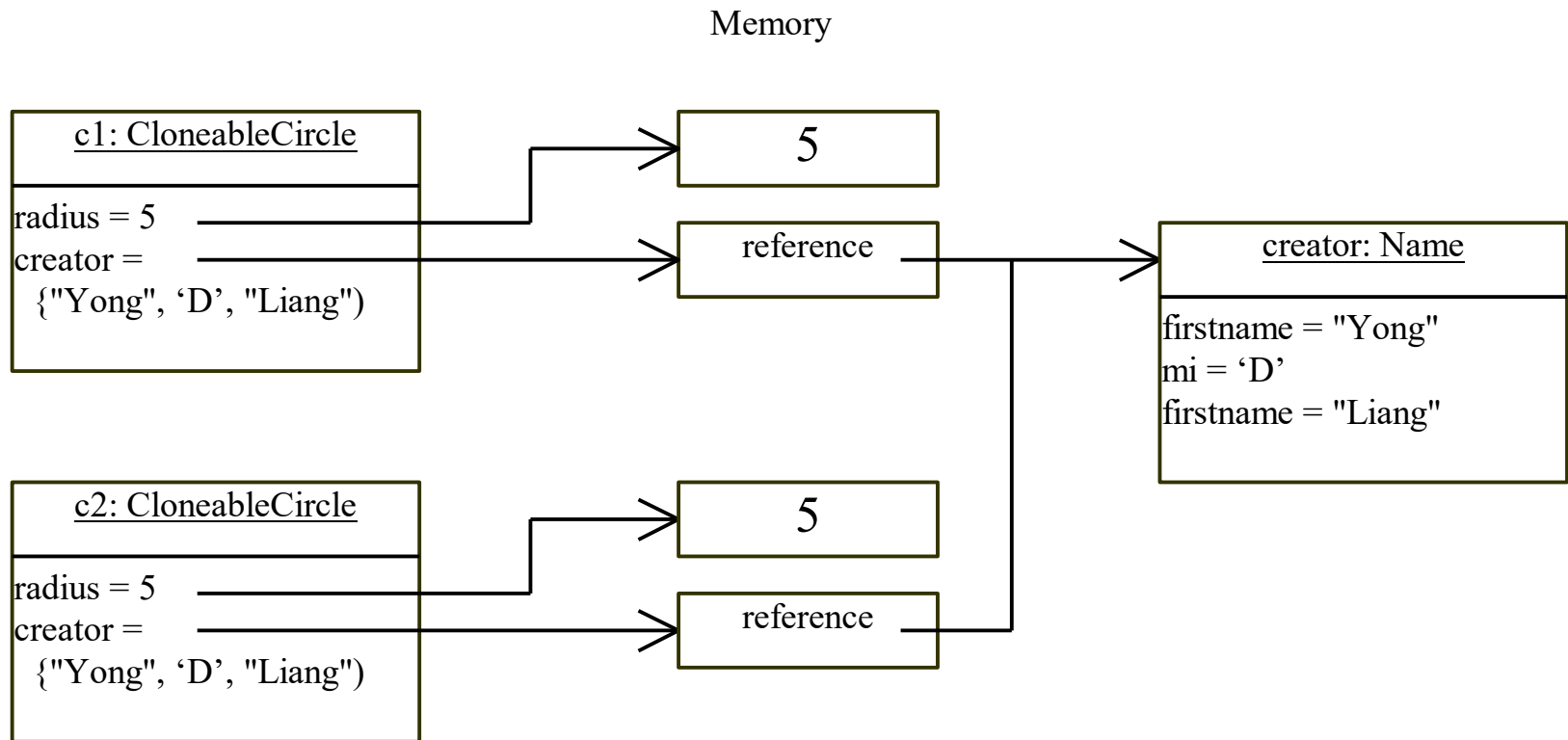
- Objective: uses the Cloneable interface to mark classes cloneable and uses the clone method to copy objects.

[TestCloneable](#)

Run

Example 8.6

Cloning Objects, cont.



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.

[ShowInnerClass](#)

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.
- An inner class is only for supporting the work of its containing outer class, and it cannot be used by other classes.