7 ADVANCED CLASS FEATURE

Objectives

- Describe static variables, methods, and initializers
- Describe final classes, methods, and variables
- Explain how and when to use abstract classes and methods
- Explain how and when to use inner classes
- Distinguish between static and non-static inner classes
- Explain how and when to use an interface

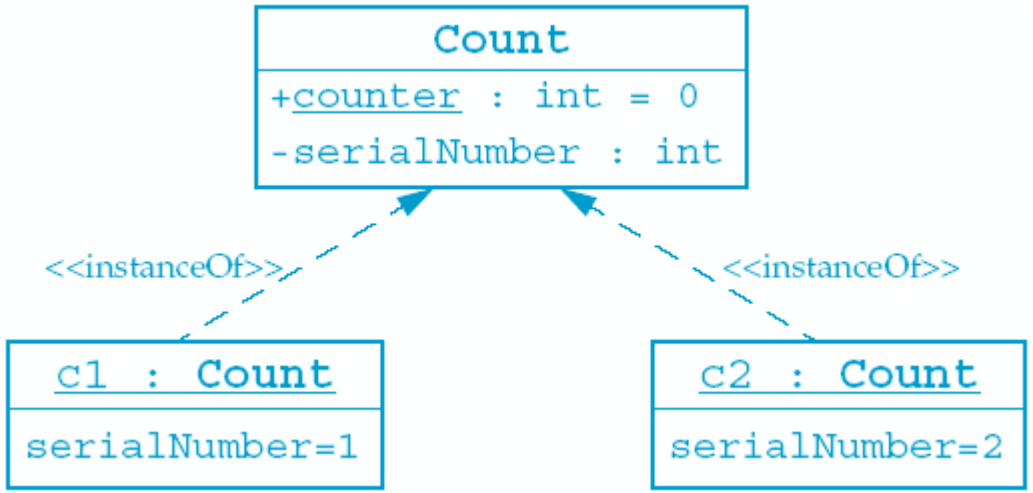
The static Keyword

- The `static` keyword is used as a modifier on variables, methods, and inner classes.
- The `static` keyword declares the attribute or method is associated with the class as a whole rather than any particular instance of that class.
- Thus `static` members are often called "class members", such as "class attributes" or "class methods".

Class Attributes

- Are shared among all instances of a class

```
1 public class Count {
2    private int serialNumber;
3    public static int counter = 0;
4
5    public Count() {
6       counter++;
7       serialNumber = counter;
8    }
9 }
```



Class Attributes

- Can be accessed from outside the class if marked as public without an instance of the class

```
1 public class OtherClass {
2       public void incrementNumber() {
3             Count.counter++;
4       }
5 }
```

Class Methods

- You can invoke static method without any instance of the class to which it belongs.

```
1 public class Count {
2 private int serialNumber;
3 private static int counter = 0;
4
5 public static int getTotalCount() {
6 return counter;
7 }
8
9 public Count() {
10 counter++;
11 serialNumber = counter;
12 }
13 }
```

```
1 public class TestCounter {
2 public static void main(String[] args) {
3 System.out.println("Number of counter is "
4 + Count.getTotalCount());
5 Count count1 = new Count();
6 System.out.println("Number of counter is "
7 + Count.getTotalCount());
8 }
9 }
```

The output of the TestCounter program is:

Number of counter is 0

Number of counter is 1

```
public class Circle {
 static int num_circles = 0;
 public double x, y, r;
  public Circle(double x, double y, double r) {
      this.x = x; this.y = y; this.r = r;
      num_circles++;
    }
    public Circle(double r) { this(0.0, 0.0, r); }
    public Circle(Circle c) { this(c.x, c.y, c.r); }
    public Circle() { this(0.0, 0.0, 1.0); }
    public double circumference() { return 2 * 3.14159 * r; }
    public double area() { return 3.14159 * r*r; }
}
```

```
public class Circle {
    public static final double PI = 3.14159265358979323846;
    public double x, y, r;
    // ... etc....
}

public double circumference() { return 2 * Circle.PI * r; }
```

derleyici

```
public class Circle {
  double x, y, r;
  public boolean isInside(double a, double b) {
      double dx = a - x;
      double dy = b - y;
      double distance = Math.sqrt(dx*dx + dy*dy);
      if (distance < r) return true;
      else return false;
   }
  // Constructor and other methods omitted.
}
```

```
public class Circle {
    public double x, y, r;
    // An instance method.  Returns the bigger of two circles.
    public Circle bigger(Circle c) {
        if (c.r > r) return c; else return this;
    }
    // A class method.  Returns the bigger of two circles.
    public static Circle bigger(Circle a, Circle b) {
        if (a.r > b.r) return a; else return b;
    }
   // Other methods omitted here.
}
```

1.
```
Circle a = new Circle(2.0);
Circle b = new Circle(3.0);
Circle c = a.bigger(b);
b.bigger(a);
```

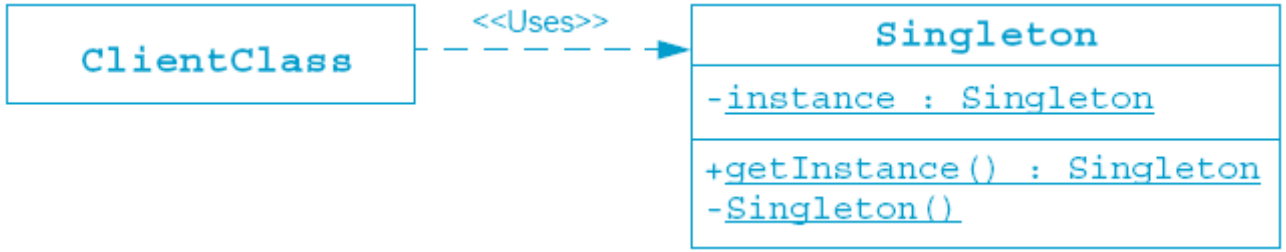
2.
```
Circle a = new Circle(2.0);
Circle b = new Circle(3.0);
Circle c = Circle.bigger(a,b);
```

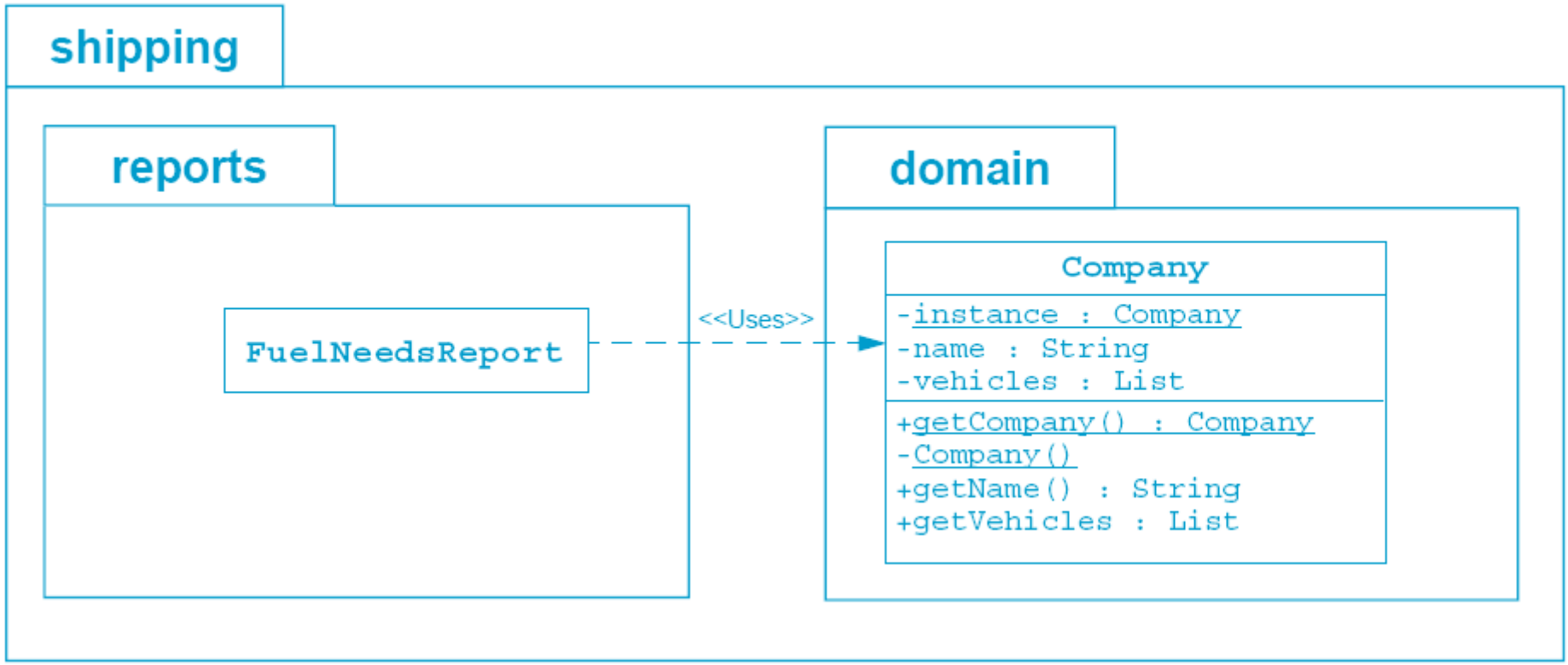
Static Initializers

- A class can contain code in a static block that does not exist within a method body.
- Static block code executes only once, when the class is loaded.
- A static block is usually used to initialize static (class) attributes

```
public class Circle {
    static private double sines[] = new double[1000];
    static private double cosines[] = new double[1000];
    static {
        double x, delta_x;
        delta_x = (Circle.PI/2)/(1000-1);
        for(int i = 0, x = 0.0; i < 1000; i++, x += delta_x) {
          sines[i] = Math.sin(x);
          cosines[i] = Math.cos(x);   }
    }
    // The rest of the class omitted.
}
```

The Singleton Design Pattern





Implementing the Singleton Design Pattern

```
1 package shipping.domain;
2
3 public class Company {
4 private static Company instance = new Company();
5 private String name;
6 private Vehicle[] fleet;
7
8 public static Company getCompany() {
9 return instance;
10 }
11
12 private Company() {...}
13
14 // more Company code ...
15 }
```

Usage Code

```
1   package shipping.reports;
2
3   import shipping.domain.*;
4
5   public class FuelNeedsReport {
6       public void generateText(PrintStream output) {
7           Company c = Company.getCompany();
8           // use Company object to retrieve the fleet vehicles
9       }
10  }
```

The `final` Keyword

- You cannot subclass a `final` class.
- You cannot override a `final` method.
- A `final` variable is a constant.
- You can set a final variable once, but that assignment can occur independently of the declaration; that is called "blank final variable."
  - A blank final instance attribute must be set in every constructor.
  - A blank final method variable must be set in the method body before being used.

`final` Classes

- The Java programming language allows you to apply the keyword final to classes. if you do this, the class cannot be inherited. For example, the class `java.lang.String` is a `final` class.
- This is done for security reasons.

`final` Methods

►You can also make individual methods as final. Methods marked final cannot be overridden. for security reasons, you should make a method final if the method has an implementation that should not be changed and is critical to the consistent state of the object.

►Methods declared final are sometimes used for optimization. The compiler can generate code that causes a direct call to the method, rather than the usual virtual method invocation that involves a runtime lookup.

`final` Variables

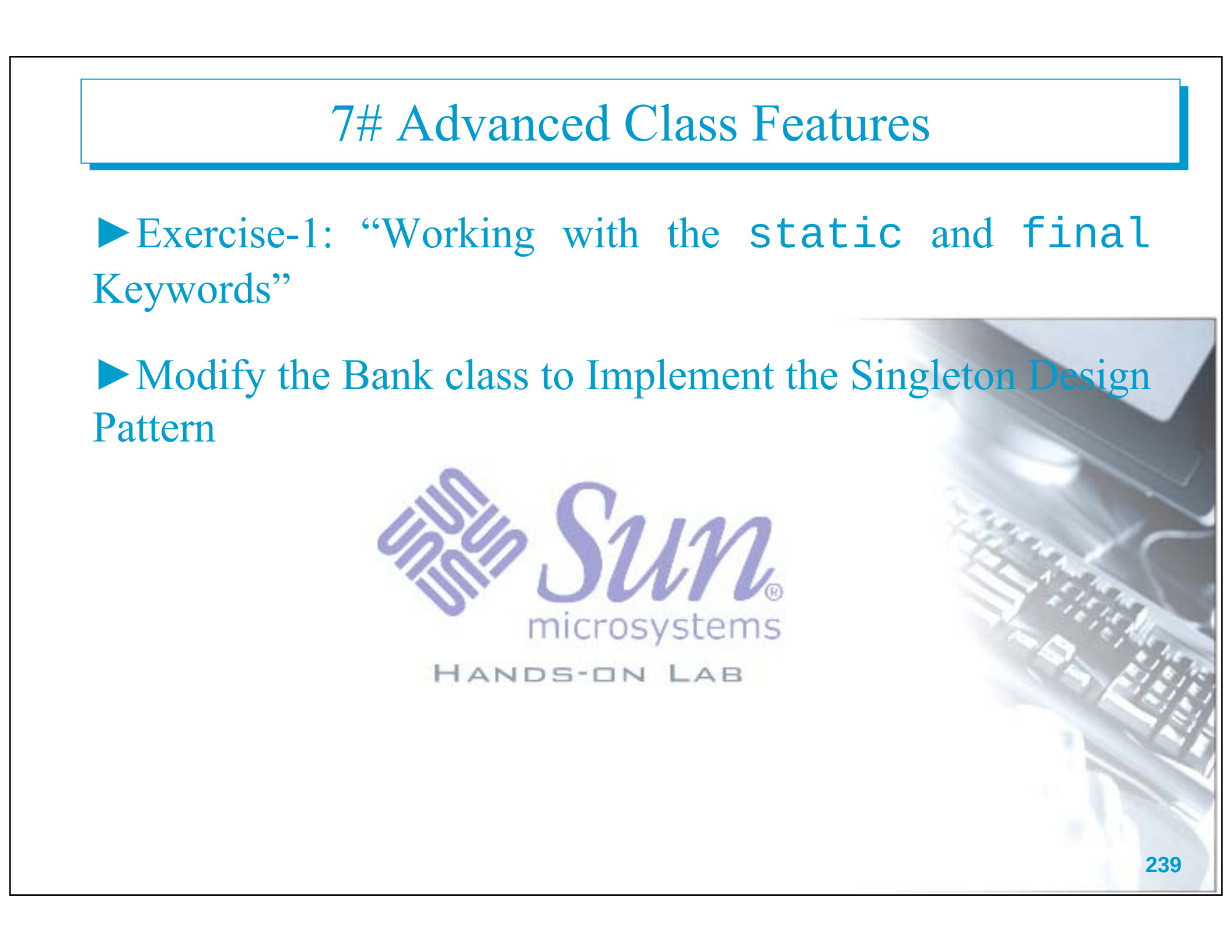
►If a variable is marked as final, the effect is to make it a constant. Any attempt to change the value of final variable causes a compiler error:

```
public class Bank{

   private static final double DEFAULT_INTEREST_RATE = 3.2 ;

  ... // more declarations

}
```

7# Advanced Class Features

►Exercise-1: “Working with the `static` and `final` Keywords”

►Modify the Bank class to Implement the Singleton Design Pattern



Yokedici Fonksiyonlar

```
protected void finalize() throws IOException {
        if (fd != null) close();
}
```

finalize fonksiyonu, C++'ın aksine, nesnenin erimi dışına çıkılınca derleyici tarafından değil, çöp toplayıcı süreç tarafından, yürütme zamanında çalıştırılır. Java nesnenin ne zaman bellekten kaldırılacağını garanti etmez. Bu nedenle nesne belirli bir süre daha erişilebilir durumda olacaktır.

Nesnelerin Yokedilmesi

- **Çöp toplama**
  - bir süreç kullanılmayan bellek alanlarını saptar : Java Grabage Collector : düşük-seviyeli süreç
  - delete ?

```
String processString(String s)
{
    StringBuffer b = new StringBuffer(s);
    return b.toString();
}
```

Kullanılan Bellek Alanlarının Çöplüğe Atılması


```
public static void main(String argv[])
{
    int big_array[] = new int[100000];
    int result = compute(big_array);
    big_array = null;
    for(;;) handle_input();
}
```

Yokedici Fonksiyon Zinciri

```
public class Point {
    ...
    protected void finalize() {
        ...
        super.finalize() ;
    }
}
public class Circle extends Point {
    ...
    protected void finalize() {
        ...
        super.finalize() ;
    }
}
```

Soyut Sınıflar (=abstract classes)

Soyut sınıflar, sadece metod isimlerini içeren ancak metodların gövdelerinin tanımlanmadığı sınıflardır. Soyut sınıftan bir nesne yaratılamaz:

```
public abstract class Shape {
    public abstract double area();
    public abstract double circumference();
}
```

Soyut sınıflar C++'daki **pure virtual** sınıflara karşılık düşmektedir:

```
class Shape {
    public :
      double area()=0;
      double circumference()=0;
}
```

```
class Circle extends Shape {
    protected double r;
    protected static final double PI = 3.14159265358979323846;
    public Circle() { r = 1.0; }
    public Circle(double r) { this.r = r; }
    public double area() { return PI * r * r; }
    public double circumference() { return 2 * PI * r; }
    public double getRadius() { return r; }
}
```

```
class Rectangle extends Shape {
    protected double w, h;
    public Rectangle() { w = 0.0; h = 0.0; }
    public Rectangle(double w, double h){ this.w=w;this.h=h;}
    public double area() { return w * h; }
    public double circumference() { return 2 * (w + h); }
    public double getWidth() { return w; }
    public double getHeight() { return h; }
}
```

```
Shape[] shapes = new Shape[3];
shapes[0] = new Circle(2.0);
shapes[1] = new Rectangle(1.0, 3.0);
shapes[2] = new Rectangle(4.0, 2.0);
double total_area = 0;
for(int i = 0; i < shapes.length; i++)
    total_area += shapes[i].area();
```

interface

Java'nın çoklu kalıtımı desteklemediğini hatırlayınız! Bunun anlamı Java'da birden fazla temel sınıf içeren bir sınıf türetemezsiniz. Java çoklu kalıtımın işlevini yerine getiren başka bir yapıya sahiptir : **interface**.

```
public interface Drawable {

    public void setColor(Color c);

    public void setPosition(double x, double y);

    public void draw(DrawWindow dw);

}
```

Uses of Interfaces

- Interfaces are useful for
  - Declaring methods that one or more classes are expected to implement
  - Determining an object's programming interface without revealing the actual body of the class
  - Capturing similarities between unrelated classes without forcing a class relationship
  - Describing "function-like" objects that can be passed as parameters to methods invoked on other objects

```
public class DrawableRectangle extends Rectangle implements Drawable {
    private Color c;
    private double x, y;
    public DrawableRectangle(double w, double h) { super(w, h); }
    public void setColor(Color c) { this.c = c; }
    public void setPosition(double x, double y) { this.x = x; this.y = y; }
    public void draw(DrawWindow dw) {
        dw.drawRect(x, y, w, h, c);
    }
}
```

```
Shape[] shapes = new Shape[3];
Drawable[] drawables = new Drawable[3];
DrawableCircle dc = new DrawableCircle(1.1);
DrawableSquare ds = new DrawableSquare(2.5);
DrawableRectangle dr = new DrawableRectangle(2.3, 4.5);
shapes[0] = dc;   drawables[0] = dc;
shapes[1] = ds;   drawables[1] = ds;
shapes[2] = dr;   drawables[2] = dr;
double total_area = 0;
for(int i = 0; i < shapes.length; i++) {
    total_area += shapes[i].area();
    drawables[i].setPosition(i*10.0, i*10.0);
    drawables[i].draw(draw_window);
}
```

Çoklu interface

```
public class DrawableScalableRectangle
        extends DrawableRectangle
        implements Drawable, Scalable {
    // The methods of the Scalable interface must be implemented here.
}
```

interface ve Kalıtım

```
public interface Transformable
                    extends Scalable, Rotateable, Reflectable
{
 }
public interface DrawingObject extends Drawable, Transformable {
 }
public class Shape implements DrawingObject
{ ...
}
```

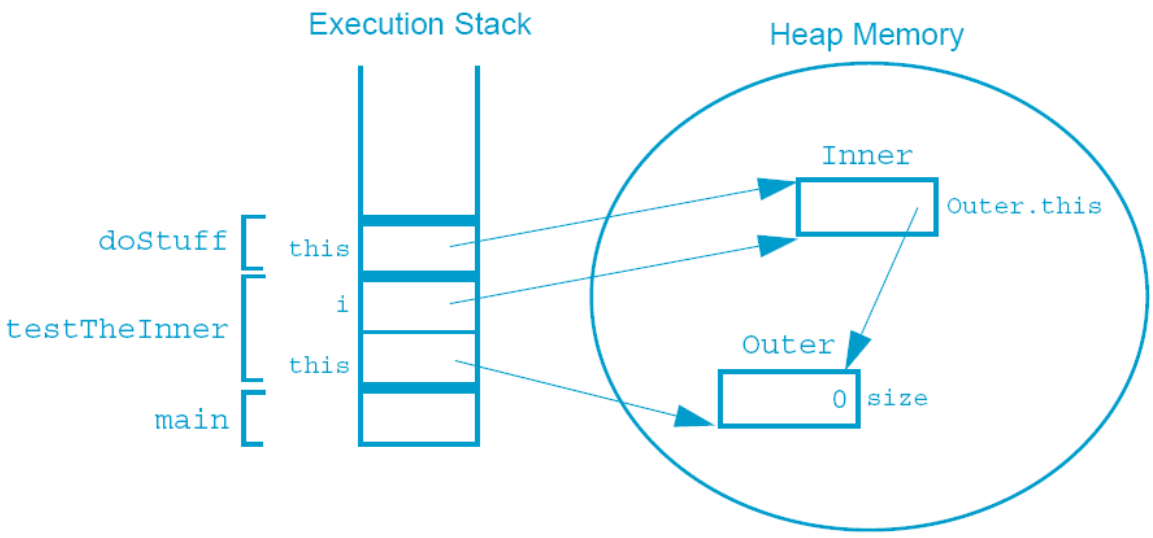
interface içinde sabit tanımlama

```
class A { static final int CONSTANT1 = 3; }
interface B { static final int CONSTANT2 = 4; }
class C implements B {
    void f() {
        int i = A.CONSTANT1;
        int j = CONSTANT2;
    }
}
```

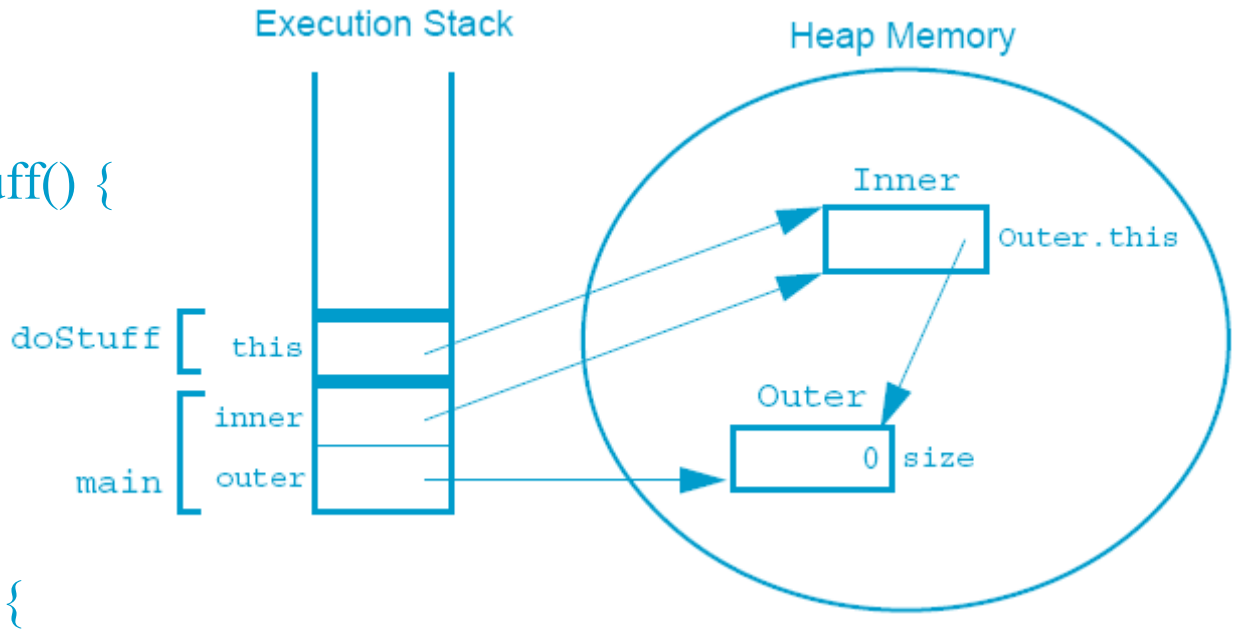
Inner Classes

- Were added to JDK 1.1
- Allow a class definition to be placed inside another class definition
- Group classes that logically belong together
- Have access to their enclosing class's scope

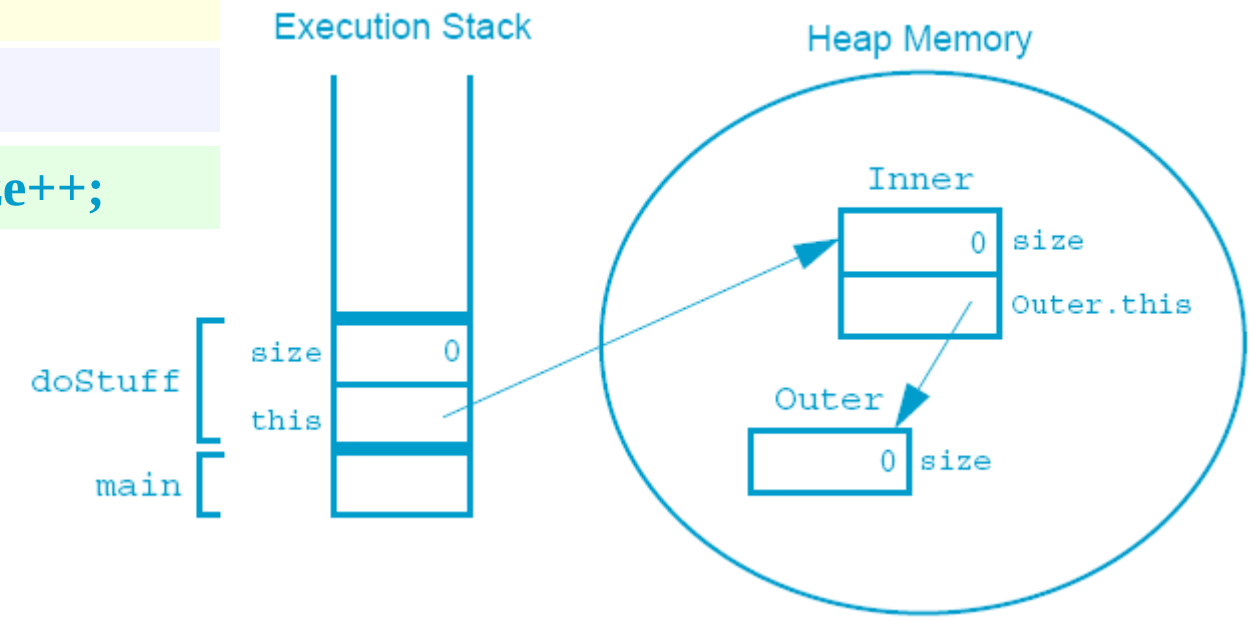
```
public class Outer1 {
    private int size ;
    /* Declare an inner class called "Inner" */
    public class Inner {
        public void doStuff() {
            // Inner class has access to 'size' from Outer
            size++;                                    Outer1.this.size++;
        }
    }
    public void testTheInner() {
      Inner i = new Inner() ;
      i.doStuf() ;
    }
}
```



```
public class Outer2 {
        private int size ;
        public class Inner {
                public void doStuff() {
                        size++;
                }
        }
}
public void testTheInner() {
        public static void main(String[] args) {
          Outer2  outer = new Outer2() ;
          Outer2.Inner inner = outer.new Inner();
          inner.doStuff() ;
        }
}
```



```
public class Outer3 {
    private int size ;
    public class Inner {
        private int size ;
        public void doStuff(int size) {
            size++;
            this.size++;
            Outer3.this.size++;
        }
    }
}
```



```
1 public class Outer4 {
2       private int size = 5;
3
4       public Object makeTheInner(int localVar) {
5             final int finalLocalVar = 6;
6
7             // Declare a class within a method!?!
8             class Inner {
9                   public String toString() {
10                        return ("#<Inner size=" + size +
11                        // " localVar=" + localVar + // ERROR: ILLEGAL
12                        "finalLocalVar=" + finalLocalVar + ">");
13                  }
14            }
15
16            return new Inner();
17 }
```

Inner Class Example

```
19    public static void main(String[] args) {
20        Outer4 outer = new Outer4();
21        Object obj = outer.makeTheInner(47);
22        System.out.println("The object is " + obj);
23    }
24 }
```

Properties of Inner Classes

- The class name can be used only within the defined scope, except when used in a qualified name. The name of the inner class must differ from the enclosing class.

- The inner class can be defined inside a method. Any variable, either a local variable or a formal parameter, can be accessed by methods within an inner class provided that the variable is marked as final.

Properties of Inner Classes

- The inner class can use both class and instance variables of enclosing classes and local variables of enclosing blocks.
- The inner class can be defined as abstract.
- Only inner classes can be declared as private or protected.
- An inner class can act as an interface implemented by another inner class.

Properties of Inner Classes

- Inner classes that are declared static automatically become top-level classes.
- Inner classes cannot declare any static members; only top-level classes can declare static members.
- An inner class wanting to use a static must declare static in the top-level class.

7# Advanced Class Features

►Exercise-2: “Working with Interfaces and Abstract Classes”

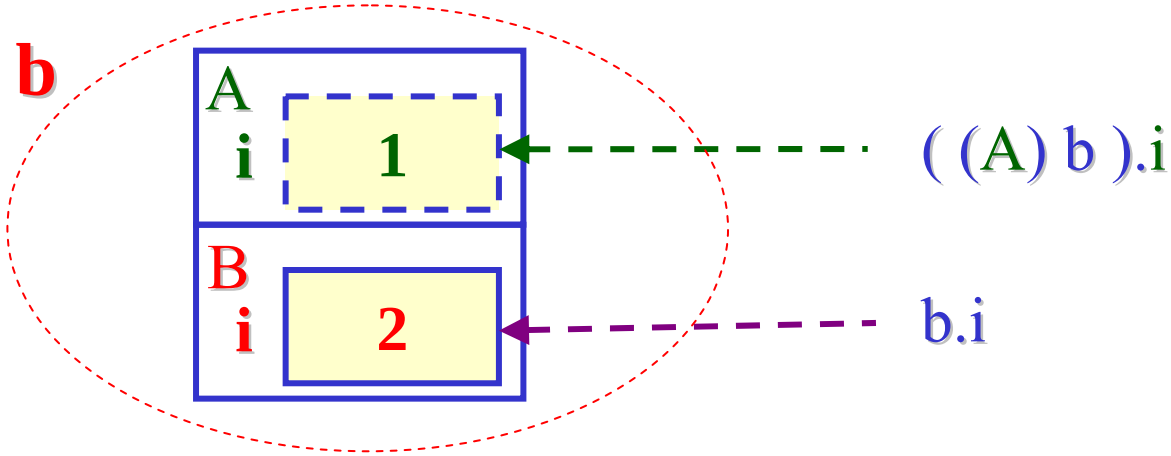
Example Revisited # 1

```
public class A {
  public int i = 1;
  public int f() { return i;}
}
public class B extends A {
 public int i = 2;
 public int f(){ return -i;}
}
```

```
public class override_test {
    public static void main(String args[]) {
        B b = new B();
        System.out.println(b.i);
        System.out.println(b.f());
        A a = (A) b;
        System.out.println(a.i);
        System.out.println(a.f());
    }
}
```

polymorphism

B b = new B() ;



Polymorphism does not work on data members but methods:

b.f()

((A)b).f()

Q: How can one access to B::i using a = (A)b ?

A: Use the operator : instanceof !

```
if (a instanceof B)
{
    B bb ;
    bb = (B) a ;
    System.out.println(b.i);
}
```

```
Employee e = new Manager() //legal

e.department = "Finance" //illegal
```

page.158

```
if (e instanceof Manager)
{
    Manager m ;
    m = (Manager) e ;
    m.department = "Finance" //legal
}
```

Example Revisited # 2


```
public class A{
    int x = 1 ;
    public int f() {return x ;}
}
public class B extends A {
    int x = 2 ;
   public int f() {return 2*x ;}
}
public class C extends B{
    int x = 3 ;
   public int f() {return ? ;}
}
```

- x
- this.x
- super.x
- ((B)this).x
- ((A)this).x
- super.super.x

```
public class polymorphism_test {
    public static void main(String args[]) {
        CC c = new CC();
        System.out.println(c.x);
        System.out.println(((BB)c).x);
        System.out.println(((AA)c).x);
        System.out.println(c.f());
        AA a = (AA) c;
        System.out.println(a.x);
        System.out.println(a.f());
        BB b = (BB) c;
        System.out.println(b.x);
        System.out.println(b.f());
        b = (BB) a ;
        System.out.println(b.x);
        System.out.println(b.f());
        Class cls = a.getClass();
        System.out.println(cls.getName()) ;
    }
}
```