Overview

- Abstract Classes
- Multiple inheritance and Interfaces
- Nested classes

Shapes

- Attributes
  - Location
  - Length, width, Radius
- Operations
  - Move
  - Draw

Shape Class

```java
public class Shape {
    protected int x, y; // Initial location
    public Shape(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public void move(int newx, int newy) {
        x = newx;
        y = newy;
    }
    public void draw() { ??? }
}
```

Abstract Shape Class

- An abstract class
- Some operations are known and some are not
- Unknown operations can be declared as abstract methods
- Cannot be instantiated

```java
public abstract class Shape {
    int x, y; // location
    public Shape(int x, int y) {
        this.x = x;
        this.y = y;
    }
    void move(int newx, int newy) {
        x = newx;
        y = newy;
    }
    public abstract void draw();
}
```

Subclasses of Shape

- Point, Rectangle, and Circle
- A concrete class
  - A subclass of an abstract superclass
  - Must implement (override) the abstract methods
  - Can be instantiated
- Why do we need a superclass when there's so little code reuse??
Sort Integers
```java
public void sort(int[] a) {
    int left = 0;
    while (left < a.length - 1) {
        int index = left;
        for (int i = left; i < a.length; i++)
            if (a[i] < a[index]) index = i;
    // swap a[index] and a[left]
    int temp = a[index];
    a[index] = a[left];
    a[left] = temp;
    left++;
    }
}
```

Sort Objects
```java
// Any objects that has a lessThan() method
public abstract class Comparable {
    public Comparable() {}
    // return true if this object is less than o
    public abstract boolean lessThan( Comparable o );
}
```

A More General Sort
```java
public void sort( Comparable a[] ) {
    int left = 0;
    while (left < a.length - 1) {
        int index = left;
        for (int i = left; i < a.length; i++)
            if (a[i].lessThan(a[index]) ) index = i;
    // swap a[index] and a[left]
    int temp = a[index];
    a[index] = a[left];
    a[left] = temp;
    left++;
    }
}
```

The Need for Multiple Inheritance
```java
// What if we want to sort an array of Point?
// Inherit both Shape and Comparable?
```

The Problem of Multiple Inheritance
```java
public class A {
    public class B {
        // ...
        public void fooBar() {
            // ...
        }
    }
}
```

Interface
```java
// Java's answer to multiple inheritance
// A interface only contains
// - Method declarations
//   - No method implementations
//   - All methods are implicitly public and abstract
// - Constants
//   - All constants are implicitly public, static, and final
```
**Interface Examples**

```java
public interface ActionListener {
    public void actionPerformed(ActionEvent e);
}

public interface AdjustmentListener {
    public void adjustmentValueChanged(AdjustmentEvent e);
}

public interface MouseListener {
    public void mousePressed(MouseEvent e);
    public void mouseClicked(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);
}
```

**Comparable Interface**

```java
public interface Comparable {
    boolean lessThan( Object c );
}
```

---

**Interface Usage**

```java
public class Point extends Shape implements Comparable {
    public Point( int x, int y ) { super(x,y); }
    public void draw() { ... }
    public boolean lessThan( Object o ) {
        Point p = (Point) o; // cast to a Point for comparable
        return x < p.x;
    }
}
```

**Exercise: Interface Constants**

```java
public interface InterA {
    final int x = 10;
    void print()
        { System.out.println(x); }
}

public interface InterB {
    public static void main( String args[] ) {
        final int x = 20;
        C c = new C();
        c.print();
    }
}
```

---

**Exercise: Interface Constants**

1. Try run the code above, observe the error, and correct it

---

**Abstract Class vs. Interface**

- **Abstract class**
  - An incomplete class
  - Class variables
  - Constructors
  - Methods and abstract methods
  - extends
  - Single inheritance
  - Cannot be instantiated

- **Interface**
  - Not a class at all
  - Only constants
  - No constructors
  - Only abstract methods (method declarations)
  - Implements
  - Multiple implementation
  - Cannot be instantiated
Nested Classes

◆ A class inside another class

```java
public class A {

    ...

    // a nested class
    class B { ... }

}
```

Properties of Nested Class

◆ Can access all members of the outer class, including private members
◆ Type
  ▪ Inside outer class: InnerClassName
  ▪ Outside outer class: OuterClassName.InnerClassName
◆ Can be declared as public, protected, or private
◆ Can be static or non-static (inner class)

Simple Nested Class Example

◆ ArrayWrapper and Iterator
  ▪ hasMoreElements()
  ▪ nextElement()