Overview

- Abstract Classes
- Multiple inheritance and Interfaces
- Nested classes

Shapes

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

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??

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 abstract void draw();
}
```
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 tmp = a[index];
        a[index] = a[left];
        a[left] = tmp;
        ++left;
    }
}
```

Sort Objects

◆ Any objects that has a lessThan() method

```java
public abstract class Comparable {
    public Comparable() {}
    // return true if this object is less than o
    public abstract boolean lessThan(Object 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 tmp = a[index];
        a[index] = a[left];
        a[left] = tmp;
        ++left;
    }
}
```

The Need for Multiple Inheritance

◆ 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 void foobar() {
        ...
    }
    public int x;
}
public class B {
    public void foobar() {
        ...
    }
    public int x;
}
public class C extends A, B {
    public void foobar() {
        ...
    }
    public int x;
}
```

◆ Which `x` or `foobar()` does C inherit?

Interface

◆ 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 ae);
}
```

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

```java
public interface MouseListener {
    public void mousePressed();
    public void mouseClicked();
    public void mouseReleased();
    public void mouseEntered();
    public void mouseExited();
}
```

```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 ??
    }
} // end of class Point
```

**Exercise: Interface Constants**

```java
public interface InterA {
    public void print();
}
```

```java
public interface InterB {
    public void print();
}
```

```java
public class C implements InterA, InterB {
    public static void main(String[] args) {
        C c = new C();
        c.print();
    }
}
```

**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
  }
  ```
An "Instance Class"

```java
class Foo {
    int a;  // instance variable
    public void print() {
        System.out.print(a);
    }

class Bar {  // "instance class"
    int b;
}
```

An instance of a nested class is always associated with an instance of the outer class.

A Nested Class Example

- ArrayWrapper
- ArrayWrapperIterator
- Iterator
  - hasNext()
  - next()

**Properties of Nested Class**

- **Type**
  - Inside outer class: InnerClassName
  - Outside outer class: OuterClassName.InnerClassName
- **Instantiation**
  - Inside outer class: new
  - Outside outer class: outerClassObject.new
- **Can access all members of the outer class, including private members**

**Variants of Nested Classes**

- Non-static nested class: Inner Class
- Static nested class
- Anonymous class
- Local class