

# Java – Inheritance, Interfaces

Kurt Schmidt

Dept. of Computer Science, Drexel University

July 7, 2016

These notes are intended for students familiar with C++  
Originally from Bruce Char & Vera Zaychik

# Intro

# Java is Object-Oriented

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

Inheritance in Java is rather like inheritance in C++. Some differences to note:

- No multiple inheritance
  - We use *Java Interfaces*
- All methods are virtual
  - All variables of type `Object` are references, so...
- There is no destructor
  - `finalize()` is called when object is released back to the heap
  - So, not reliably called
- Java allows static attributes of any type to be initialised at the declaration
- Java allows classes to be defined inside a class
- Java allows unnamed classes

# Java Classes

# Classes as **Namespaces**

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting  
Containers

Interfaces

Exceptions

Nested  
Classes

The `Math` class, e.g., is simply a container for methods and constants.

- Can't be instantiated
  - The default constructor is made `private`
- The class is `final` – can't be subclassed
- All methods are `public static`
  - `Math.sin( a ) ;`
  - `Math.exp( x ) ;`
- Constants are `public static` attributes

# Inheritance

# Inheritance vs. Aggregation

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

## ■ Inheritance is the *is-a* relationship

- A square is a shape
- An employee is a person
- A professor is an employee
- So, a professor is a person
- It's not perfect

## ■ Aggregation is the *has-a* relationship

- A square has a color
- An employee has an address
- A car has an engine
  - And 4 tires

# Inheritance

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

Given a simple class:

```
public class Person {  
    protected String _name ;  
    public Person( String n ) { _name = n ; }  
    public String getName() { return _name ; }  
}
```

We can define a subclass:

```
public class Professor extends Person {  
    protected String _id ;  
    public Person( String n, string i )  
    { super(n) ; _id = i ; }  
    public String getName()  
    { return "Prof. " + _name ; }  
}
```



# Abstract Superclass

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

- An abstract class can not be instantiated
- It typically contains method declarations, w/out definitions
  - These are behaviors that subclasses must provide to be meaningfull objects
  - Use the `@override` annotation
- E.g., a closed shape might well know its color, and declare a method to compute its area
  - All closed shapes have an area
  - Computed differently for each shape
  - Area of an abstract shape is meaningless

# Abstract Superclass – example

```
public abstract class ClosedShape {  
    protected Color _color ;  
    public ClosedShape( Color c ) { _fill = c ; }  
    public Color color() { return _fill ; }  
    public abstract double getArea() ;  
}
```

We create an actual shape

```
public class Circle extends ClosedShape {  
    protected double _radius ;  
  
    public Circle( Color c, double r )  
    { super(c) ; _radius = r ; }  
  
    @Override  
    public double getArea()  
    { return Math.Pi * _radius * _radius ; }  
  
    public double getRadius() { return _radius ; }  
}
```

# Casting

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

- Objects can be cast up the tree
  - Always safe
  - Explicit cast not needed
  - Only methods from ancestor may be called
  - Remember, all methods are virtual

```
ClosedShape s = new Circle( Color.BLUE, 3 );
```

- Objects can be cast down the tree
  - Might throw ClassCastException
  - Only methods from ancestor may be called
  - Remember, all methods are virtual

```
Circle c = (Circle) s ; // This works fine  
Square q = (Square) s ; // This throws exception
```

# instanceof Operator

- We can test objects
- An object of a subclass is always an instance of an ancestor
- An object of a class is not, generally, an instance of a descendant

```
Circle c = new Circle( Color.PURPLE, 8 ) ;  
...  
if( c instanceof Square ) System.out.println( "c is a Square" ) ;  
if( c instanceof Circle ) System.out.println( "c is a Circle" ) ;  
if( c instanceof ClosedShape )  
    System.out.println( "c is a ClosedShape" ) ;
```

```
c is a Circle  
c is a ClosedShape
```

# Containers of Shapes

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

We can create containers of shapes:

```
public static void main( String [] args )
{
    GridLoc l = new GridLoc( 1, 2 ) ;
    ArrayList<Shape> zoo = new ArrayList<Shape>() ;
    zoo.add( new Circle( 3, 1 )) ;
    zoo.add( new Square( 5, 1 )) ;

    for( Shape s : zoo )
    {
        System.out.printf( "Area: %.2f\n", s.getArea() ) ;
    }
}
```

Area: 28.27

Area: 25.00

# Interfaces

# Interfaces

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

- Java does not support multiple inheritance
  - This is not a bad thing
  - Multiple inheritance is messy, both in design and implementation
- An interface describes behaviors which must be supplied by any implementing class
  - It *declares* methods
  - It does not *define* any
  - Attributes, however, *can* be defined

# Interfaces

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

- An interface can “inherit” from one or more other interfaces:

```
public interface Stealthy
{
    public void stalk() ;
}

public interface Predator extends Stealthy
{
    public void pounce() ;
}
```

- A class might implement multiple interfaces:

```
public class Cat extends Animal implements Predator, Yowler
{ ... }
```



# Containers and Interfaces

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

- Instances of a class implementing an interface can be viewed as objects of that type
  - A `KeyListener` object, whatever else it is, has methods `keyTyped`, `keyPressed`, and `keyReleased`
  - All `Animals` that implement the `Yowler` interface can be contained together:

```
public static void main( String [] args )
{
    ArrayList<Yowler> zoo = new ArrayList<Yowler>() ;
    zoo.add( new Cat( "Sylvester" ) ) ;
    zoo.add( new Wolf( "Nighteyes" ) ) ;
    ...
    for( Yowler y : zoo )
    {
        y.singAncientSongOfYourPeople() ;
    }
}
```

# Exceptions

# Library Exceptions

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

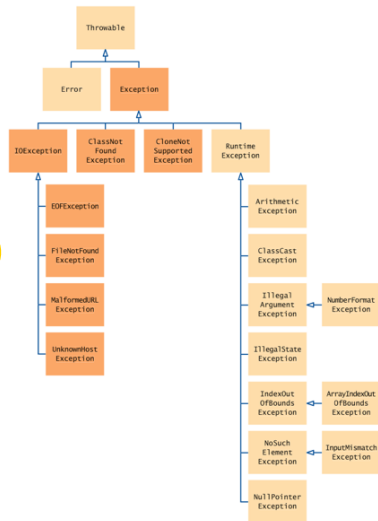
Containers

Interfaces

Exceptions

Nested  
Classes

- These are the **exceptions** in the Java standard library
- Exceptions in darker boxes are **checked exceptions**
  - Must be caught, or listed in a throws statement
  - All *should* be so listed
- Inherit off of any of these to **make your own exceptions**
  - No behavior need be defined
  - Its value is its type



Taken from Cay Horstmann's Big Java, 4<sup>th</sup> ed.

# User-Defined Exceptions

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

```
public class ThatsOdd extends IllegalArgumentException
{
    public ThatsOdd( String s ) { super( s ); }
}
```

Can be used as any other exception:

```
public static void foo( int i ) throws ThatsOdd {
    if( i%2==1 )
        throw new ThatsOdd( "We're partial to evens, in this method." );
    ...
}

public static void bar( int n ) {
    try {
        foo( n/2 );
        ...
    } // try
    catch( ThatsOdd e ) {
        System.err.printf( "bar> caught ThatsOdd: %s\n", e.toString() );
        e.printStackTrace();
    }
}
```

# Nested Classes

# Nested Classes in Java

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

- Java allows classes to be defined inside other classes
  - Even inside methods
  - Even unnamed
- We'll briefly look at these, describe common uses
  - We will not discuss nuances of design in this course

# Types of Nested Classes

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

- The possibilities are:
  - Static nested class
  - Non-static inner class
    - Need an instance of outer class to instantiate
  - Defined inside a method (*method-local*)
    - Can only be instantiated in that method
    - Object can be returned from method
- Do not confuse visibility (scope) with access
  - A private member is not accessible, outside of that class

# Public Static Nested Classes

- Just a container for similar classes

```
public class Public
{
    public static class Inner1
    {
        public void talk()
        { System.out.println( "In Public.Inner1.talk" ) ; }

    } // class Inner1

    public static class Inner2
    {
        public void talk()
        { System.out.println( "In Public.Inner2.talk" ) ; }

    } // class Inner1
} // class Public
```



# Public Static Nested Classes (cont.)

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting  
Containers

Interfaces

Exceptions

Nested  
Classes

- Used like any other class
- Note the scope

```
public static void main( String [] args )
{
    Public.Inner1 i1 = new Public.Inner1() ;
    Public.Inner2 i2 = new Public.Inner2() ;

    i1.talk() ;
    i2.talk() ;
} // class Inner1
```

# Non-Static Nested Classes

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

- Also called *inner classes*
- About all Java offers in the way of a *closure*
  - Needs an instance of outer class
  - These objects capture their surrounding scope, even if the containing object is no longer accessible
- Can be an alternative to exposing outer class' attributes to entire package (or world)
- An inner class may be *unnamed*
  - Commonly used to install event handlers

# Public Static Nested Classes (cont.)

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

E.g., inside a Dialog we might have event handlers for some controls

```
buttonYes = new JButton() ;
buttonNo = new JButton() ;
...
buttonYes.addActionListener(
    new java.awt.ActionListener() // class definition here
    {
        public void actionPerformed( java.awt.event.ActionEvent e )
        {
            doSomething() ;
        }
    }
)
```

# Factory Methods

Java –  
Inheritance,  
Interfaces

Kurt Schmidt

Intro

Java Classes

Inheritance

Casting

Containers

Interfaces

Exceptions

Nested  
Classes

- Object creation is abstracted
- Class will choose which subclass to create
- Subclasses are hidden (can't be instantiated directly)
  - Private static nested class, OR
  - Method local (defined inside the factory method)
- Consider the Sorting Hat, from Hogwarts
  - It decides which House a student belongs to
  - All houses have same i/f:

```
public abstract class House {  
    ...  
    public abstract represent() ;  
}
```

# Factory Methods – Hogwarts

- Inside we could define our subclasses:

```
private static class Gryffyndor extends House {
    @Override
    public void represent()
    { System.out.println( "Gryffyndor!" ) ; }
}

private static class Slytherin extends House {
    @Override
    public void represent()
    { System.out.println( "Slytherin!" ) ; }
}

private static class RavenClaw extends House {
    @Override
    public void represent()
    { System.out.println( "RavenClaw!" ) ; }
}
```

# Hogwarts (cont.)

- Here's our factory method, with exclusive access to the subclasses:

```
public static House SortingHat( int i )
{
    House rv = null ;
    switch( i%3 ) {
        case 0 : rv = new Gryffindor() ; break ;
        case 1 : rv = new Slytherin() ; break ;
        case 2 : rv = new RavenClaw() ; break ;
    }
    return rv ;
}
```

- We let the factory decide the proper subclass to use:

```
public static void main( String [] args )
{
    House h = House.SortingHat( 27 ) ;
    h.represent() ;
}
```