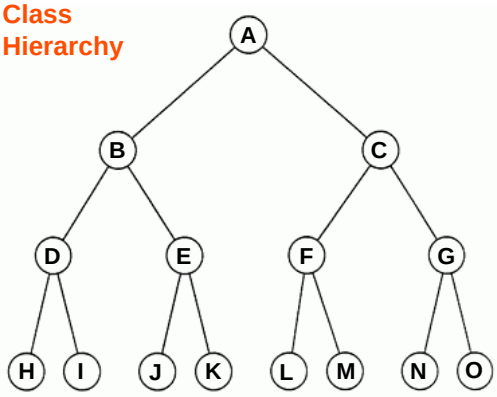
Exceptions

November 30, 2007

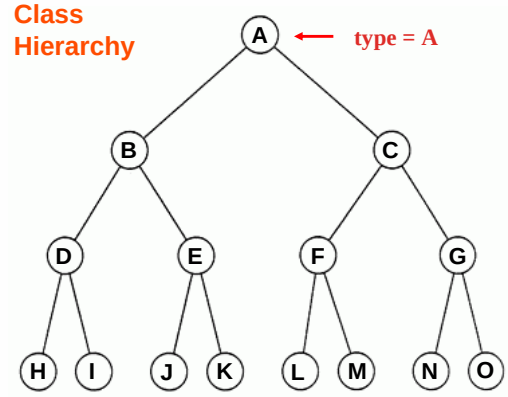
ComS 207: Programming I (in Java)
Iowa State University, FALL 2007
Instructor: Alexander Stoytchev

Quick Review of Polymorphism & Polymorphic Variables

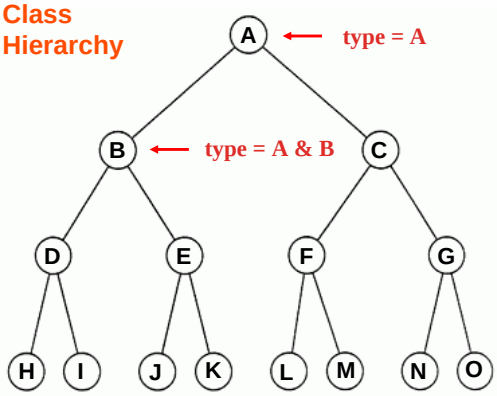
Class Hierarchy



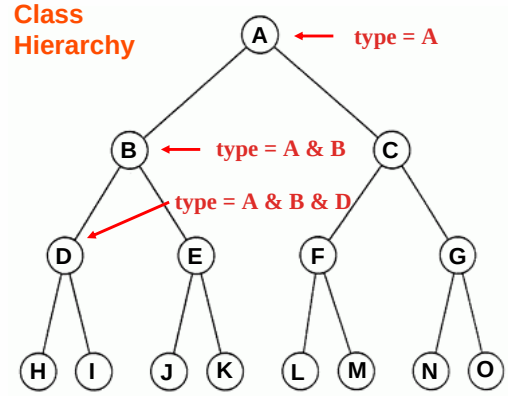
Class Hierarchy

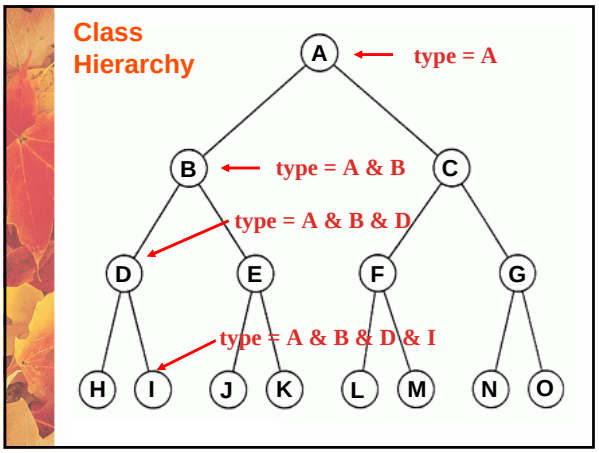


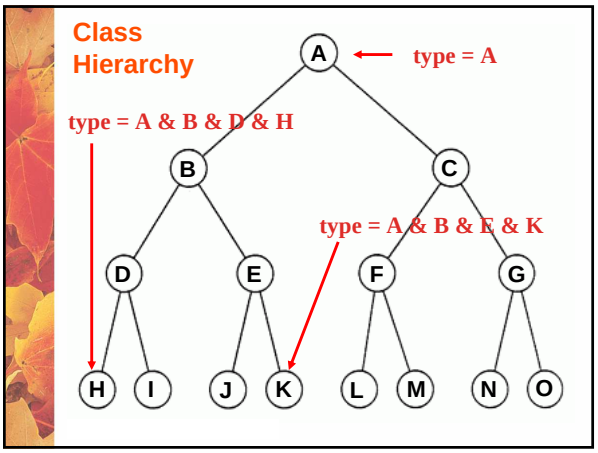
Class Hierarchy

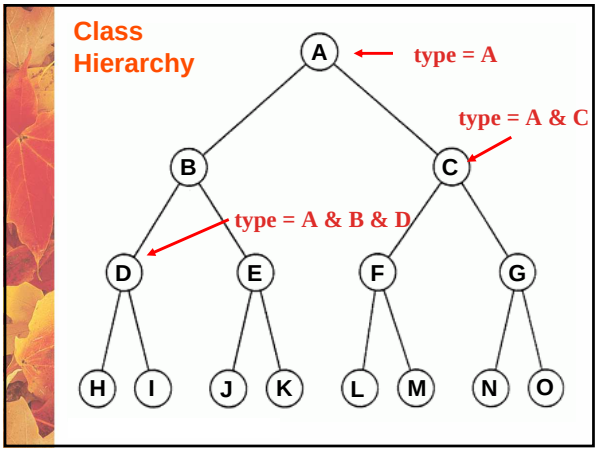


Class Hierarchy



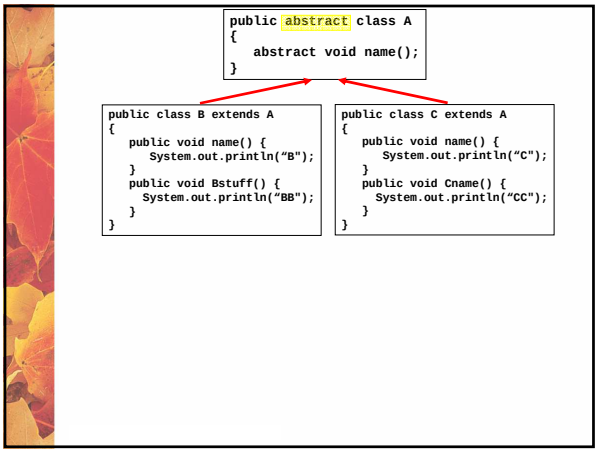






Example: Letters class hierarchy

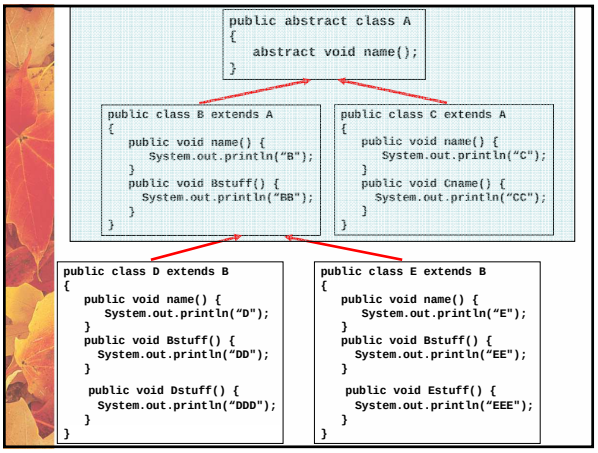
- A.java
- B.java
- C.java
- D.java
- E.java
- F.java
- G.java
- Driver.java



```
public abstract class A
{
   abstract void name();
}
```

```
public class B extends A
{
   public void name() {
      System.out.println("B");
   }
   public void Bstuff() {
     System.out.println("BB");
   }
}
```

```
public class C extends A
{
   public void name() {
      System.out.println("C");
   }
   public void Cname() {
     System.out.println("CC");
   }
}
```



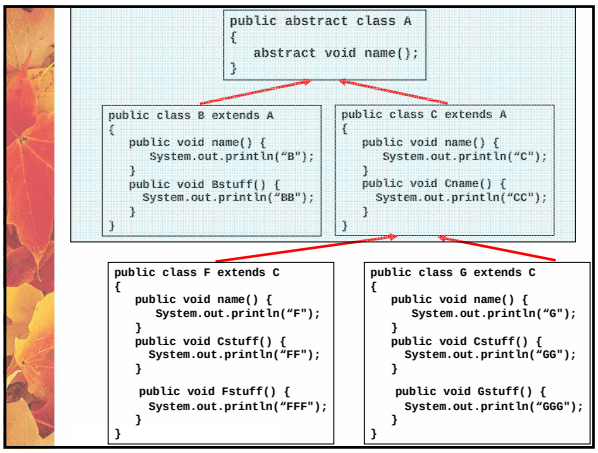
```
public abstract class A
{
   abstract void name();
}
```

```
public class B extends A
{
   public void name() {
      System.out.println("B");
   }
   public void Bstuff() {
     System.out.println("BB");
   }
}
```

```
public class C extends A
{
   public void name() {
      System.out.println("C");
   }
   public void Cname() {
     System.out.println("CC");
   }
}
```

```
public class D extends B
{
   public void name() {
      System.out.println("D");
   }
   public void Bstuff() {
     System.out.println("DD");
   }
    public void Dstuff() {
     System.out.println("DDD");
   }
}
```

```
public class E extends B
{
   public void name() {
      System.out.println("E");
   }
   public void Bstuff() {
     System.out.println("EE");
   }
    public void Estuff() {
     System.out.println("EEE");
   }
}
```




```
public class Driver1
{
        public static void main(String[] args)
        {
                A[] letter= new A[6];

                letter[0]= new B();
                letter[1]= new D();
                letter[2]= new E();
                letter[3]= new C();
                letter[4]= new F();
                letter[5]= new G();

                for(int i=0; i<6; i++) {
                     letter[i].name ();
                }
        }
}
```

Result:

```
B
D
E
C
F
G
```

```
public class Driver2
{
        public static void main(String[] args)
        {
                A[] letter= new A[6];

                letter[0]= new B();
                letter[1]= new D();
                letter[2]= new E();
                letter[3]= new C();
                letter[4]= new F();
                letter[5]= new G();

                for(int i=0; i<3; i++) {
                     ((B)letter[i]).Bstuff();
                }

                for(int i=3; i<6; i++) {
                     ((C)letter[i]).Cstuff();
                }
        }
}
```

Result:

```
BB
DD
EE
CC
FF
GG
```

```
public class Driver3
{
        public static void main(String[] args)
        {
                B[] b_letter= new B[3];
                b_letter[0]= new B();
                b_letter[1]= new D();
                b_letter[2]= new E();

                C[] c_letter= new C[3];
                c_letter[0]= new C();
                c_letter[1]= new F();
                c_letter[2]= new G();

                for(int i=0; i<3; i++)     {
                    b_letter[i].Bstuff();
                }

                for(int i=0; i<3; i++)     {
                    c_letter[i].Cstuff();
                }
        }
}
```

Result:

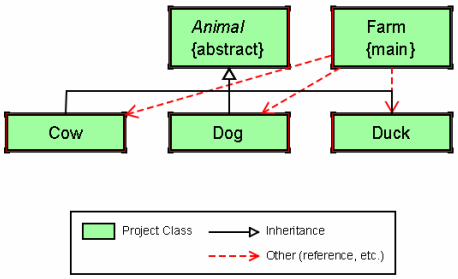
```
BB
DD
EE
CC
FF
GG
```

Quick Review of Last Lecture

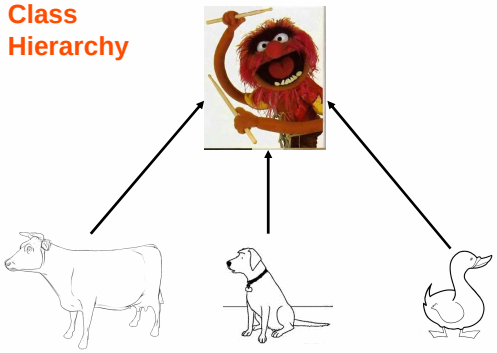
Example: Animals class hierarchy

- **Animal.java**
- **Cow.java**
- **Duck.java**
- **Dog.java**
- **Farm.java**

You can use jGrasp to draw diagram like this one



Class Hierarchy



```
public abstract class Animal
{
   abstract void makeSound();
}
```

```
public class Cow extends Animal
{
   public void makeSound()
   {
      System.out.println("Moo-Moo");
   }
}
```

```
public class Dog extends Animal
{
   public void makeSound()
   {
      System.out.println("Wuf-Wuf");
   }
}
```

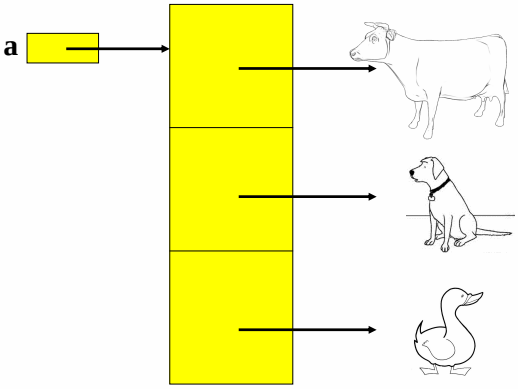
```
public class Duck extends Animal
{
   public void makeSound()
   {
      System.out.println("Quack-Quack");
   }
}
```

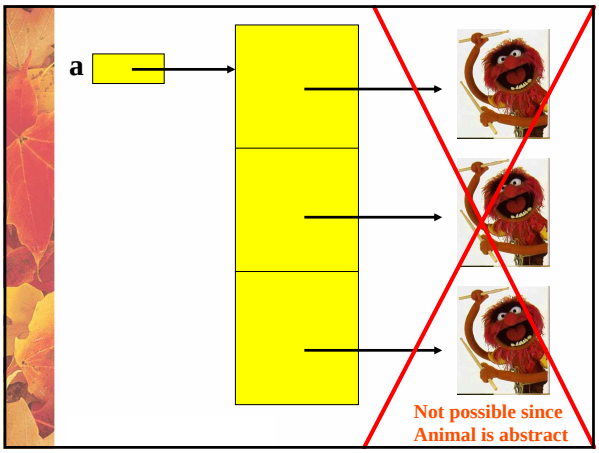
```
public class Farm
{
      public static void main(String[] args)
      {
            Cow c=new Cow();
            Dog d=new Dog();
            Duck k= new Duck();

            c.makeSound();
            d.makeSound();
            k.makeSound();
      }
}
```

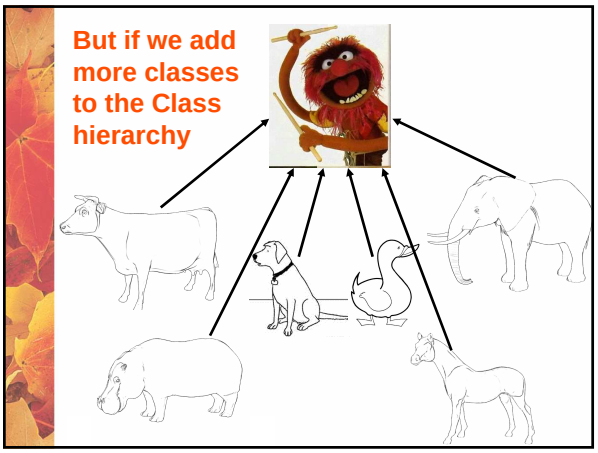
Result:
Moo-Moo
Wuf-Wuf
Quack-Quack

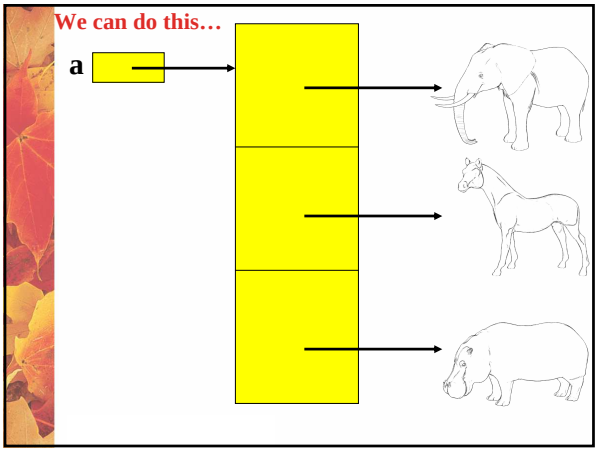
```
public class Farm2
{
      public static void main(String[] args)
      {
            Animal[] a = new Animal[3];

            a[0] = new Cow();
            a[1] = new Dog();
            a[2] = new Duck();

            for(int i=0; i< a.length; i++)
                  a[i].makeSound();
      }
}
```

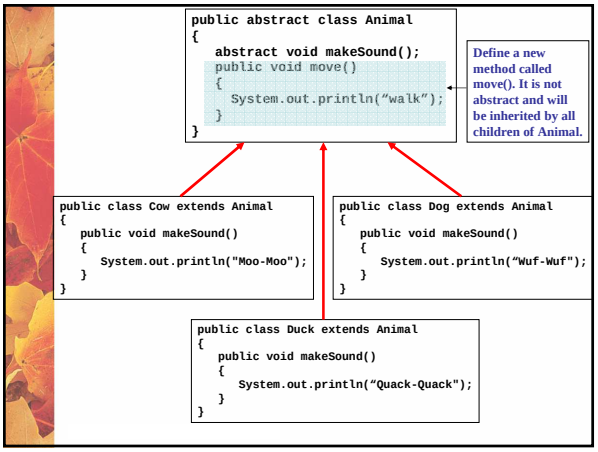
Result:
Moo-Moo
Wuf-Wuf
Quack-Quack





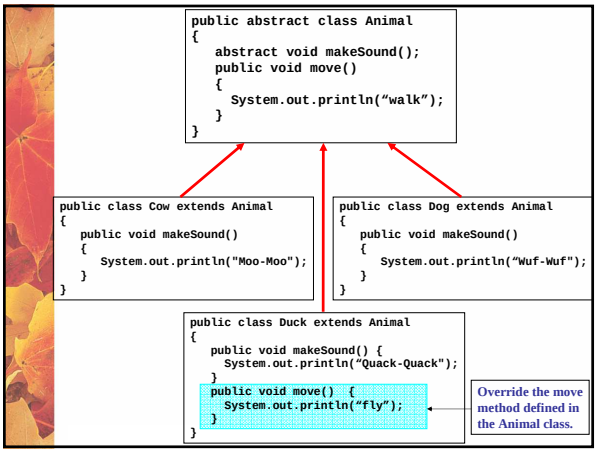







```
public class Farm2b
{
     public static void main(String[] args)
     {
          Animal[] a = new Animal[3];

          a[0] = new Cow();
          a[1] = new Dog();
          a[2] = new Duck();

          for(int i=0; i< a.length; i++)
               a[i].move();
     }
}
```

Result:
walk
walk
walk

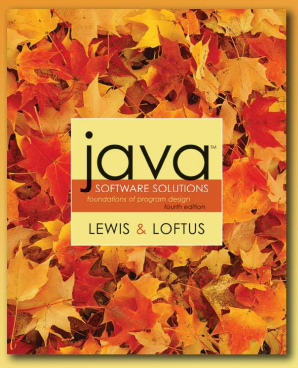


```
public class Farm2c
{
      public static void main(String[] args)
      {
            Animal[] a = new Animal[3];

            a[0] = new Cow();
            a[1] = new Dog();
            a[2] = new Duck();

            for(int i=0; i< a.length; i++)
                  a[i].move();
      }
}
```

Result:
Walk
Walk
Fly

Chapter 9

Section 9.1 & 9.2




```
public abstract class Animal
{
   abstract void makeSound();
   public void move()
   {
     System.out.println("walk");
   }
}
```

```
public class Cow extends Animal
{
   public void makeSound()
   {
     System.out.println("Moo-Moo");
   }
}
```

```
public class Dog extends Animal
{
   public void makeSound()
   {
     System.out.println("Wuf-Wuf");
   }
}
```

```
public class Duck extends Animal
{
   public void makeSound() {
     System.out.println("Quack-Quack");
   }
   public void move()  {
     System.out.println("fly");
   }
   public void dive() {
     System.out.println("Diving…");
   }
}
```

Only Ducks can dive.

```
public class Farm2d
{
      public static void main(String[] args)
      {
            Animal[] a = new Animal[3];

            a[0] = new Cow();
            a[1] = new Dog();
            a[2] = new Duck();

            for(int i=0; i< a.length; i++)
                  a[i].dive();
      }
}
```

Compile Error, since dive() is defined only for Duck objects and not for all objects derived from Animal.

```
public class Farm2d
{
      public static void main(String[] args)
      {
            Animal[] a = new Animal[3];

            a[0] = new Cow();
            a[1] = new Dog();
            a[2] = new Duck();

            ((Duck)a[2]).dive();
      }
}
```

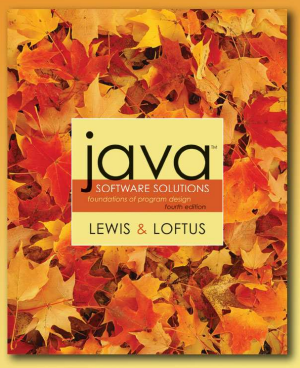
This works OK, but requires a cast from a reference to Animal to a reference to Duck.

```
public class Farm2d
{
      public static void main(String[] args)
      {
            Animal[] a = new Animal[3];

            a[0] = new Cow();
            a[1] = new Dog();
            a[2] = new Duck();

            ((Duck)a[2]).dive();
      }
}
```

Result:
Diving…

Chapter 9

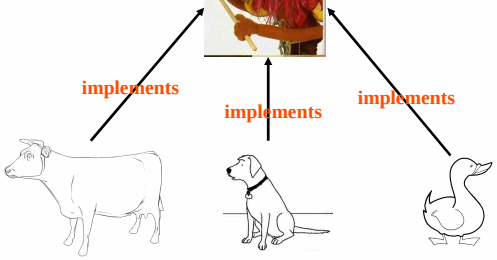
Section 9.3



PEARSON Addison Wesley

The Animals example with interfaces

In this case Animal is an interface.




```
public interface Animal
{
   public void makeSound();
}
```

```
public class Cow implements Animal
{
   public void makeSound()
   {
      System.out.println("Moo-Moo");
   }
}
```

```
public class Dog implements Animal
{
   public void makeSound()
   {
      System.out.println("Wuf-Wuf");
   }
}
```

```
public class Duck implements Animal
{
   public void makeSound()
   {
      System.out.println("Quack-Quack");
   }
}
```

```
public class iFarm
{
      public static void main(String[] args)
      {
            Animal domestic;
            domestic = new Cow();
            domestic.makeSound();

            domestic = new Dog();
            domestic.makeSound();

            domestic = new Duck();
            domestic.makeSound();
      }
}
```

Result:
Moo-Moo
Wuf-Wuf
Quack-Quack

```
public interface Animal
{
   public void makeSound();
   public void move();
}
```

Define a new method called move(). Because Animal is an interface this method cannot be defined as in the previous example in which Animal was an abstract class.

```
public class Cow implements Animal
{
   public void makeSound() {
      System.out.println("Moo-Moo");
   }
   public void move() {
      System.out.println("walk");
   }
}
```

```
public class Dog implements Animal
{
   public void makeSound() {
      System.out.println("Wuf-Wuf");
   }
   public void move() {
      System.out.println("walk");
   }
}
```

Replicated code!!!

```
public class Duck implements Animal
{
   public void makeSound() {
      System.out.println("Quack-Quack");
   }
   public void move() {
      System.out.println("fly");
   }
}
```

The only move() method that is different from the rest.

```
public class iFarm2
{
      public static void main(String[] args)
      {
            Animal domestic;
            domestic = new Cow();
            domestic.move();

            domestic = new Dog();
            domestic.move();

            domestic = new Duck();
            domestic.move();
      }
}
```

Result:
walk
walk
fly

```
public interface Animal
{
   public void makeSound();
   public void move();
}
```

```
public class Cow implements Animal
{
   public void makeSound() {
      System.out.println("Moo-Moo");
   }
   public void move() {
      System.out.println("walk");
   }
}
```

```
public class Dog implements Animal
{
   public void makeSound() {
      System.out.println("Wuf-Wuf");
   }
   public void move() {
      System.out.println("walk");
   }
}
```

```
public class Duck implements Animal
{
   public void makeSound() {
      System.out.println("Quack-Quack");
   }
   public void move() {
      System.out.println("fly");
   }
   public void dive() {
      System.out.println("Diving…");
   }
}
```

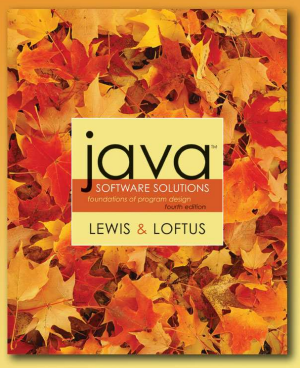
Only Ducks can dive().

```
public class iFarm3
{
      public static void main(String[] args)
      {
            Animal domestic;
            domestic = new Cow();
            //domestic.dive(); // error

            domestic = new Dog();
            //domestic.dive(); // error

            domestic = new Duck();
            // domestic.dive(); // error

            ((Duck)domestic).dive(); // OK, but uses a cast
      }
}
```

Result:
Ducks can dive.

Chapter 10

Exceptions



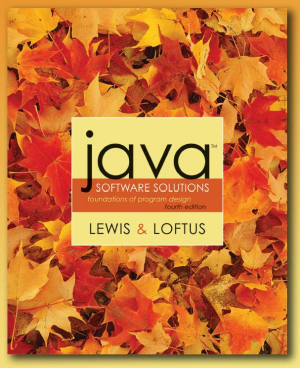
PEARSON Addison Wesley

Exceptions

- **An *exception* is an object that describes an unusual or erroneous situation.**

Chapter 10

Sections 10.1 -10.3



PEARSON Addison Wesley

Exceptions

- **Exceptions are *thrown* by a program, and may be *caught* and *handled* by another part of the program**
- **A program can be separated into a normal execution flow and an *exception execution flow***
- **An *error* is also represented as an object in Java, but usually represents a unrecoverable situation and should not be caught**

Exception Handling

- Java has a predefined set of exceptions and errors that can occur during execution
- A program can deal with an exception in one of three ways:
  - ignore it
  - handle it where it occurs
  - handle it an another place in the program
- The manner in which an exception is processed is an important design consideration

Exception Handling

- If an exception is ignored by the program, the program will terminate abnormally and produce an appropriate message
- The message includes a *call stack trace* that:
  - indicates the line on which the exception occurred
  - shows the method call trail that lead to the attempted execution of the offending line
- See `Zero.java` (page 533)

Examples:

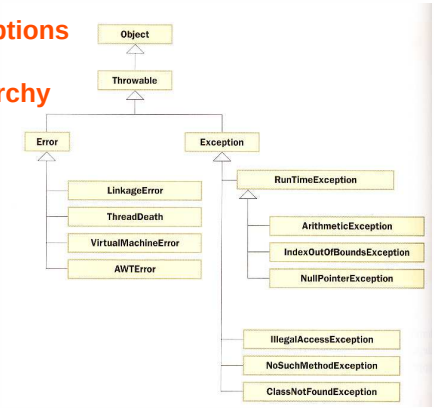
`Zero.java`

`Zero_Caught.java`

The Exception Class Hierarchy

- Classes that define exceptions are related by inheritance, forming an exception class hierarchy
- All error and exception classes are descendents of the `Throwable` class
- A programmer can define an exception by extending the `Exception` class or one of its descendants
- The parent class used depends on how the new exception will be used

Exceptions Class Hierarchy



Exception Hierarchy

- java.lang.Object
- java.lang.Throwable
- java.lang.Exception
- java.lang.RuntimeException
- java.lang.ArithmeticException

The try Statement

- To handle an exception in a program, the line that throws the exception is executed within a *try block*
- A try block is followed by one or more *catch* clauses
- Each catch clause has an associated exception type and is called an *exception handler*
- When an exception occurs, processing continues at the first catch clause that matches the exception type

The finally Clause

- A try statement can have an optional clause following the catch clauses, designated by the reserved word `finally`
- The statements in the finally clause always are executed
- If no exception is generated, the statements in the finally clause are executed after the statements in the try block complete
- If an exception is generated, the statements in the finally clause are executed after the statements in the appropriate catch clause complete

Examples:

`OutOfBounds.java`

`OutOfBounds_Caught.java`

Exception Hierarchy

- java.lang.Object
- java.lang.Throwable
- java.lang.Exception
- java.lang.RuntimeException
- java.lang.IndexOutOfBoundsException
- java.lang.ArrayIndexOutOfBoundsException

Examples:

`NullReference.java`

`NullReference_Caught.java`

Exception Hierarchy

- java.lang.Object
- java.lang.Throwable
- java.lang.Exception
- java.lang.RuntimeException
- java.lang.NullPointerException

Examples:

`ClassCast.java`

`ClassCast_Caught.java`

Exception Hierarchy

- java.lang.Object
- java.lang.Throwable
- java.lang.Exception
- java.lang.RuntimeException
- java.lang.ClassCastException

On-line Java Documentation

- http://java.sun.com/j2se/1.5.0/docs/api/index.html

THE END