Object Technology

Dr. Priti Srinivas Sajja

Introduction

- **Name:** Dr. Priti Srinivas Sajja
- **Email :** priti@pritisajja.info
- **Mobile :** +91 9824926020
- **URL:** http://pritisajja.info
- **Academic Quali.:** Ph. D in Computer Science
- **Thesis Title:** Knowledge-Based Systems for Socio-Economic Rural Development
- **Area of Interest:** Artificial Intelligence
- **Publications:** 216 in International/ National Conferences, Journals & Books

Unit 4: Inheritance, Interfaces and Exception handling

- Use of Inheritance, Inheriting Data members and Methods, constructor in inheritance
- Multilevel Inheritance
- Creation and Implementation of an interface, Interface reference
- Introduction to the Concept of Exception Handling

Inheritance in Java

```
class room{
  double width;
  double length;

//constructor……
  room(double x, double y){
    width=x;
    length=y; }

 //overloading constructor……
 room(double x){
    width=length=x;}

//Method with return type……
  double area(){ return(length*width);}
```

Inheritance in Java

```
public static void main(String args[]){
    room r1= new room(10.56,2.0);
    room r2= new room(10.06);

    System.out.println("Area of the first  room is " +
    r1.area());
    System.out.println("Area of the second room is " +
    r2.area());

    }
}
```

Inheritance in Java

```
//consider class of room as discussed in previous slides
class bedroom extends room
 { double height;

// constructor for the extended class
  bedroom(double x, double y, double z){
    super(x,y);
    height=z;   }

// method of the extended class
   double volume(){ return(length*width*height);}
}
```

Inheritance in Java

Later we can use this constructor as follows.

bedroom b1= new bedroom(14.5,12.0,8.0);

System.out.println("Area of the bedroom is " +b1.area());

System.out.println("Volume of the bedroom is " +b1.volume());

Interface

- An interface is basically **a kind of class**.
- Like classes interface **contains methods and variables**.
- The interface defines the **abstract method and final fields** only.
- That means interfaces **do not supply any code** to implement these methods.
- It is necessary to **implement an interface** and it is the responsibility of a class which wants to use it.

Interface

- To implement
- access class *classname*[extends *superclass*]
- [implements interface [,interface…]]
- {//class body….
- }

Think……..

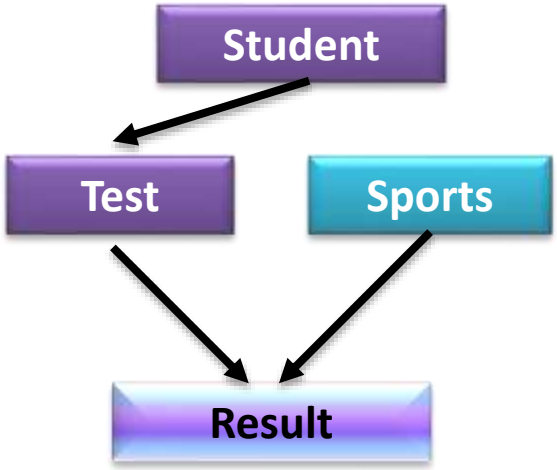
- *Can we use public as access?*
- *Can we extend an interface?*

Inheritance and Interfaces

```
class Student{
    int rollNumber;
    void getNumber(int n) { rollNumber=n;}
    void putNumber() {System.out.println("RollNo:" +rollNumber);}
}
class Test extends Students
{ float part1, part2;
void getMarks(float m1, float m2) {part1=m1; part2=m2;}
void putMarks()          {System.out.println("Marks Obtained");
                          System.out.println(" Part1= " + part1);
                          System.out.println(" Part2= " + part2); }}
interface Sports{ float sportWt=6.0;  void putWt();}
```

Inheritance and Interfaces

```
class Results extends Test implements Sports{
   float total;
   public void putWt(){System.out.println("SporWt= "+ sportWt);}
   void display ()        { putNumber(); putMarks(); PutWt();
                            total= part1+part2+sportWt;
                            System.out.println("Total= "+ total);}}
class Hybrid{
 public static void main(String args[]){
   Results student1= new Results();
   student1.getNumber(1234);
   student1.getMarks(27.5, 33.0);
   student1.display();
}}
```



Inheritance in Vehicle Class

```
public class Vehicle {
    ..............
    ..............
 }
```

IS-A Relationship

```
public class Car extends Vehicle {
    ..............
    ..............
}
```

IS-A Relationship

```
public class Alto extends Car {
    ..............
    ..............
 }
```

Fig a: Is-A relationship between classes

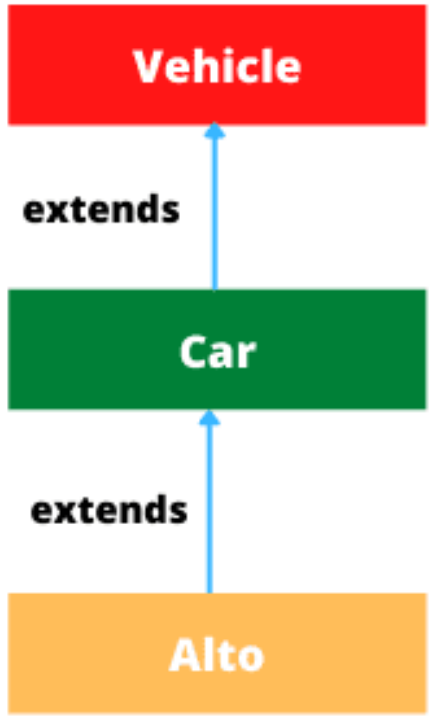
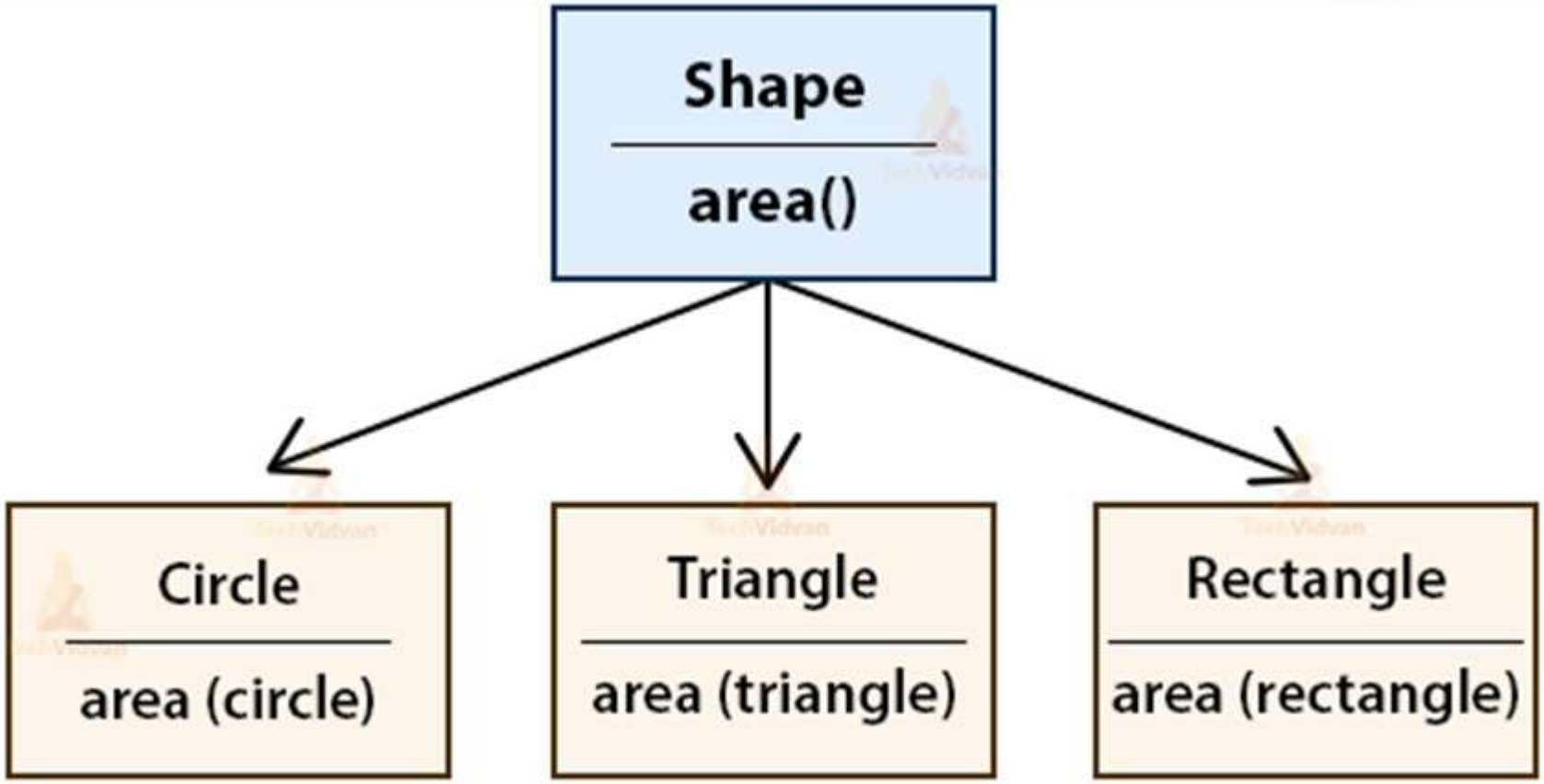
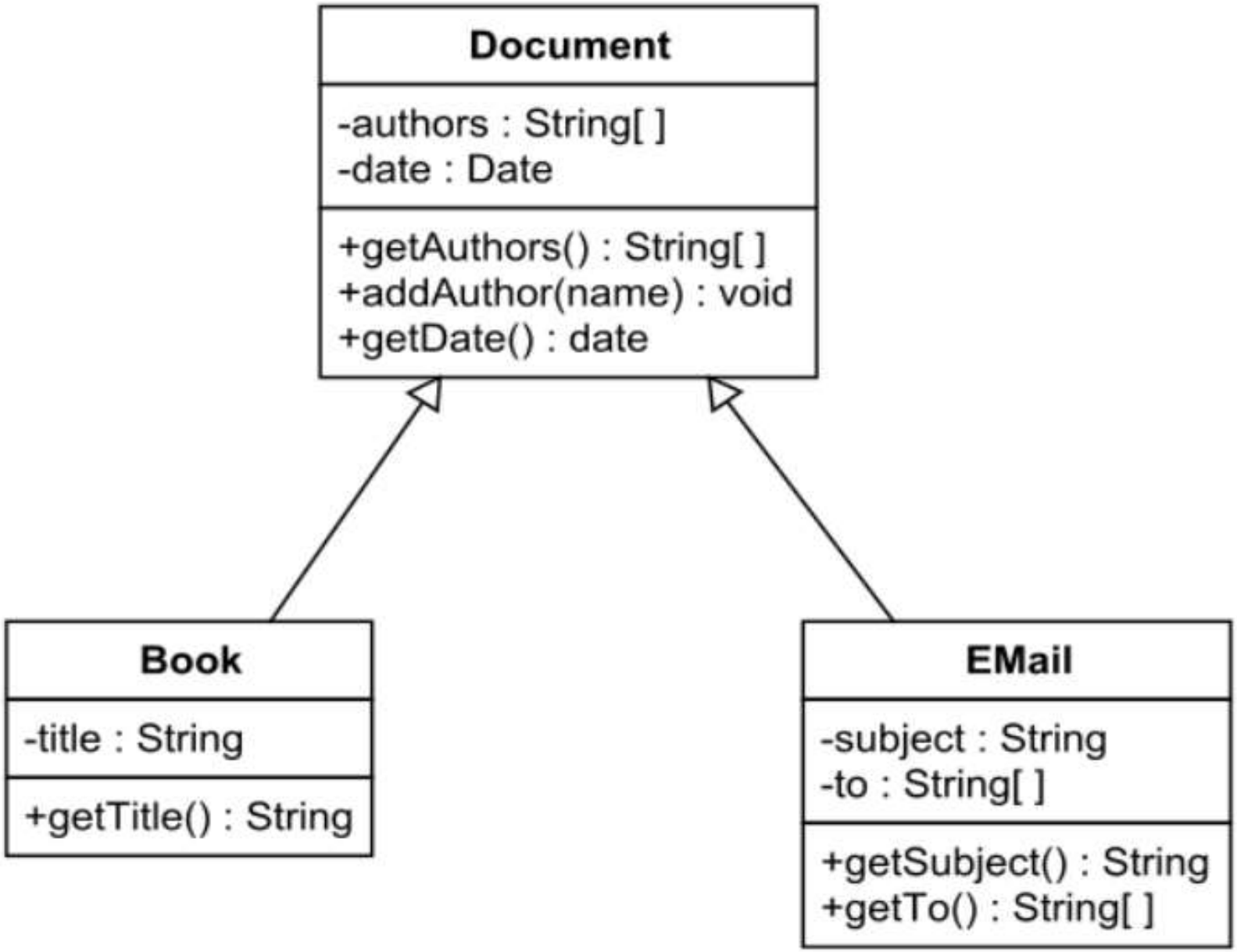


Fig b: Inheritance hierarchy for Vehicle, Car, and Alto


Shape
area()
Circle
area (circle)
Triangle
area (triangle)
Rectangle
area (rectangle)



Source: Cs.utsa.edu

Exception Handling in Java

- An **exception is an abnormal condition** that arise in a code sequence at run time. An exception therefore, is a **run time error**.
- A java **exception is an object that describes an exceptional condition** that has occurred in a piece of a code.
- When an exception condition arise, an **object representing that exception is created and thrown** in the method that cause the error.
- This method may choose **to handle that exception or pass it to other method**.
- Not only by a run time system of Java, **bur manually also an exception is created and thrown**.

- Java exception handling is managed through five key words:
  - try, catch, throw, throws and finally.
- Program statement which you'd like to monitor are enclosed in try (and catch) block.
- If any exception occurs, within that try block, it is thrown.
- Your code can catch the exception thrown and takes necessary action.
- System generated exception are thrown automatically by the java run time system.
- To manually throw the exception, use the keyword throw.
- Any exception which must be thrown out of the method, must be specified with the throws clause.
- Any code that absolutely must be executed before a method returns, is put in finally block.

- **Here is the general form:**
- **try**

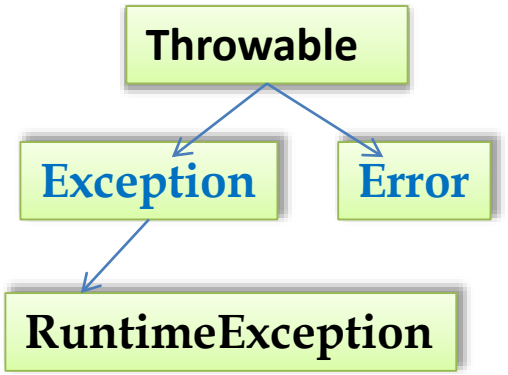
  { // block of code to monitor }
- **catch(Exceptiontype1 ob1)**

  { //exception handling code…1}
- **catch(Exceptiontype1 ob2)**

  { //exception handling code…2}
- **//…**
- **Finally {//… must be executed before return}**

Here exception type is the type of exception that has occurred.

- All exception types are subclasses of the **built in class Throwable**. Thus Throwable is at the top of the exception class hierarchy.



- Immediately below this Throwable into two branches.
- One branch is headed by Exception. And there is an important subclass of Exception that is called RuntimeException.
- The another branch is Error, which defines exceptions that are not expected to be caught under normal conditions.
- Error is used by the Java run time system to indicates errors. Eg. Stack overflow→run time error given by the Java.

Exception Handling

```
class ex1{
 public static void main(String args[]){
   int d=0;
   int a=42/d;
}}
```

- Results in

```
javac ex1.java
ex1.java:4: Arithmetic exception.
   int a=42/d;
          ^
```

When java run time system detects the attempt to **divide by zero**, it constructs a new exception and then throws the exception (outside the method).

This cause the execution of the program stopped.

That is such exception **must be caught within the program** and dealt with immediately

- Here we have not specified any such exception handling code, so the exception is caught by a default handler of the Java.
- Any exception that that is not caught by your program will ultimately processed by the default handler.
- The default handler displays a string describing the exception and terminates the program.

Exception Handling (see ex2.java)

```
class ex2{
 public static void main(String args[]){
   int d, a;
   try { d=0;
         a=42/d;
        } catch (ArithmeticException e)
         { System.out.println("Division by 0");}
    }
}
```

Continuing after Exception Handling (see ex3.java)

```
class ex3{
 public static void main(String args[]){
   int d, a;
   try { d=0;  a=42/d; System.out.println("No...");
       } catch (ArithmeticException e)
                { System.out.println("Division by 0");
                  System.out.println("System will say :" +e);
                  a=0;//setting a==0 and continue....
                }
System.out.println("a:= " + a );

}}
```

Multiple Catches: (see ex4.java)

```
class ex4{
public static void main(String args[]){
  int a[] ={5,10};
  int b=5;
  try  { int  x=a[1]/(b-a[0]);} // change it to int x=a[2]/b-a[1] and run
  catch(ArithmeticException e)
        {System.out.println("Division by zero...");}
  catch(ArrayIndexOutOfBoundsException e)
        { System.out.println("Array index error...");}
  catch(ArrayStoreException e)
        {System.out.println(“Wrong data type...");}

   int y=a[1]/a[0];
   System.out.println("Y is = "+y);
} }
```

Throw:

- So far, you have only been catching exceptions that are thrown by java run time system.
- The general form of throw is
  - Throw ThrowableInstance
- Here the ThrowableInstance must be an object of type throwable or a subclass of throwable.
- Simple types such as integer, char and String are non-throwable.
- There are two way you can obtain a throwable object
  - Using a parameter into a catch statement
  - Creating one with new operator.

Throw:

```
class ThrowDemo{
 static void demoproc(){
  try{ throw new NullPointerException("Demo");
    } catch (NullPointerException e){
      System.out.println("Caught inside demo");  throw e;}
 }
 public static void main(String args[]){
  try{ demoproc();
    } catch (NullPointerException e){
      System.out.println("Recaught: "+e);  }
 }}
```

Caught inside demo

Recaught:
java.lang.NullPointerException:
Demo

Throws:

- If a method is capable of causing an exception that it does not handle, it must specify them with the **throws** statement.

- A throws clause lists the types of exceptions that a method might throw.

- This is necessary for all exceptions except Error type or RuntimeException, or any of their subclasses.

Finally:

- When exceptions are thrown, execution may take non-linear path(disturbed).
- The finally block will execute weather or not an exception is thrown.
- This can be useful in operation like closing the files and releasing the resources before returning.
- The finally clause is optional, however one try statement requires at least one catch or one finally clauses.
- If a finally block is attached with a try, it will be executed upon conclusion of try.

Java's Built-in Exception:

- **ArithmeticException**: Arithmetic error, such as divide by zero.
- **ArrayIndexOutOf BoundsException**: Using array subscript out of bound
- **ArrayStoreException**: Assignment to an array element of an incompatible type
- **ClassCastException**: Invalid Cast.
- **IllegalArgumentException**: Illegal argument used to invoke a method.

Java's Built-in Exception:

- **NegativeArraySizeException** : Array created with a negative size.
- **NumberFormatException** : Invalid conversion of string to a number format.
- **StringIndexOutOfBounds** : Attempt to index outside the bounds of a string.

Java's java.lang Exception:

- **ClassNotFoundException** : Class not found.
- **ClassNotSupportedException**: Attempt to clone an object that doesnot implement the cloneable interface.
- **InstantiationException** : Attempt to create an object of an abstract class.
- **IllegalAccessException** : Access to a class is denied.

Java's java.lang Exception:

- **InterruptedException** : One thread has been interrupted by another thread.
- **NoSuchFieldException**: A requested field does not exist.
- **NoSuchMethodException** : A requested method does not exist.

Reading characters from console: (Extra) - Use of Throws

```
import java.io.*;
class Read{
public static void main(String args[]) throws IOException{
 char c;
 BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Enter characters:, 'q' to quit");
 do{
    c=(char)br.read();
    System.out.println(c);
    } while (c!='q');
 } }
```

Main Reference:

- Patrick Naughton and Herbert Schildt, The Complete Reference Java 2, Seventh, Tata McGraw Hill Pub., 2007

Visit pritisajja.info to download this show….