All interface methods are public by default (even if you don't specify it explicitly in the interface definition), so all implementing methods must be public too, since you can't reduce the visibility of the interface method.

CS205 Object Oriented Programming in Java

Module 3 - More features of Java (Part 6)

Prepared by

Renetha J.B.

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

Topics

- **More features of Java :**
  - ☑ **Exception Handling:**
    - **Checked Exceptions**
    - **Unchecked Exceptions**
    - ***try*** Block and ***catch Clause***

Exception Handling

- An ***exception*** *is* an **abnormal condition** that occur in a code sequence at ***run time.***
  - Exception is a RUN TIME ERROR
- A Java exception is an **object** that describes an exceptional (that is, error) condition that occurred in a piece of code.
- When an exceptional condition arises,
  - an object representing that exception is created and
  - *It is thrown in the method that caused the error.*
    - That method may choose to handle the exception itself, or pass it on.
    - The exception is then *caught and processed*

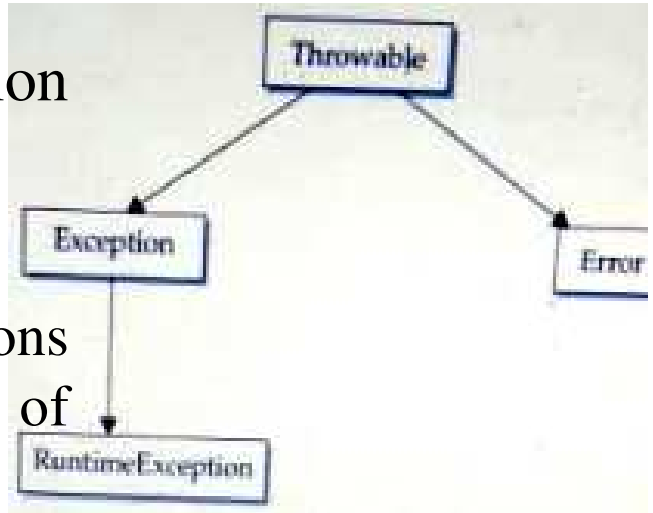
Exception Handling(contd.)

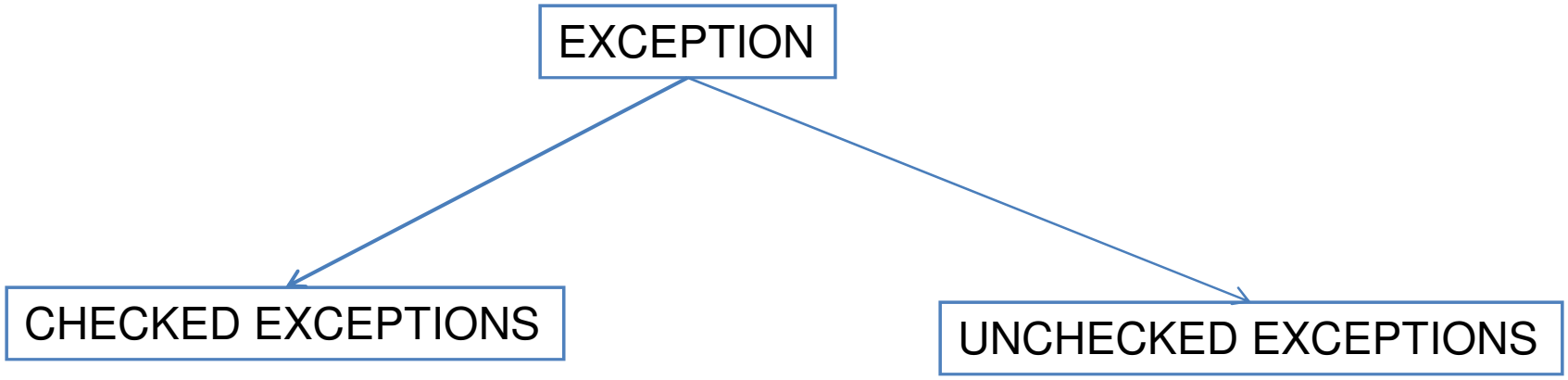
- Exceptions can be **generated by**
  - the Java run-time system, or
  - they can be manually generated by your code.
- Exceptions thrown by Java are related to
  - **Fundamental errors** that **violate the rules** of
    - the Java language or
    - the constraints of the Java execution environment.

Exception Types

- All exception types are subclasses of the built-in class **Throwable.**
- **Throwable** has two subclasses that partition exceptions into two distinct branches.
  - ❑ One branch is headed by **Exception.**
    - This class is used for exceptional conditions that ***user programs should catch***. Subclass of this helps to create custom exception types.
    - **RuntimeException** is a subclass of **Exception.**
  - ❑ The other branch is headed by **Error**
    - This defines exceptions that are ***not expected to be caught*** under normal circumstances by our program.(***unchecked***)
    - Exceptions of type Error are used by the Java run-time system to indicate errors.

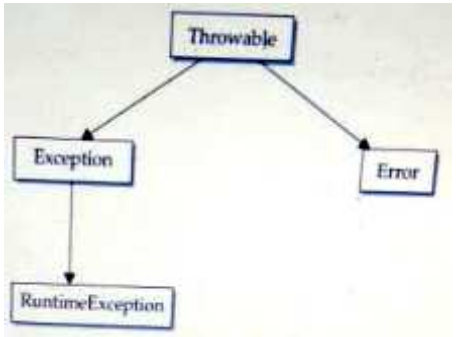
      E.g. Stack overflow,Out of Memory error




EXCEPTION
CHECKED EXCEPTIONS
UNCHECKED EXCEPTIONS

Unchecked exception

- Unchecked exception classes are defined inside **java.lang** package.
  - The **unchecked exceptions** are subclasses of the standard type **RuntimeException.**
  - In the Java language, these are called ***unchecked exceptions*** *because* ***the compiler does not check*** *to see whether there is a method that handles or* throws these exceptions.
  - If the program has unchecked exception then it will *compile without error* but exception occurs when program runs.



•E.g Exceptions under Error , ArrayIndexOutOfBoundException

Unchecked exception(contd.)

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

Checked exception

- There are some exceptions that are defined by **java.lang** that **must be included in a method's throws list,** if a method generates such exceptions and that ***method does not handle it itself***. These are called **checked exceptions**

Exception	Meaning
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the **Cloneable** interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

- **IOException**
- **FileNotFoundException**
- **SQLException**

Checked exception(contd.)

- Checked exceptions are the exceptions (in java.lang) that are checked at compile time.
  - If some statement in a method **throws a checked exception**, then that method must
    - either handle the exception or
    - it must specify the exception using *throws* keyword.

Checked exceptions

- Checked at compile time.(COMPILE TIME EXCEPTIONS)
- Not sub class of RunTimeException
- The method must either handle the exception or it must specify the exception using *throws* keyword.
- Shows compile error if checked exception is not handled.
- E.g. *ClassNotFoundException, IOException*

Unchecked exceptions

- NOT checked at compile time.(RUN TIME EXCEPTINS)
- Sub class of RunTimeException
- It is NOT needed to handle or catch these exceptions
- DO NOT Show compile error if exception is not handled. But shows run-time error.
- Eg. *ArithmeticException, ArrayIndexOutOfBoundsException*

Exception handling fundamentals

Exception handling fundamentals(contd.)

- Program statements that we want to check for exceptions are written within a **try block.**
  - If an exception occurs within the try block, it is **thrown**.
  - The code inside **catch** can catch this exception and handle it in some manner.
- ***System-generated exceptions*** are automatically thrown by the Java run-time system.
- To **manually throw** an exception, use the keyword **throw.**
- Any exception that is thrown out of a **method** must be specified as such by a **throws** clause.
- Any code that absolutely must be executed after a **try block completes** is put in a **finally block.**

```
try {
    // block of code to monitor for errors
}
catch (ExceptionType1 exOb)
{
    // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb)
{
    // exception handler for ExceptionType2
}
// ...
finally
{
    // block of code to be executed after try block ends
}
```

finally de munne try catch il return ondelum finally execute ayitte return cheyyyu

catch with most specific exception must come at first tazottu pokum thorum specificity koranju koranju varanam

In some compilers you have to put catch(java.lang.Exception e)

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

Uncaught Exceptions

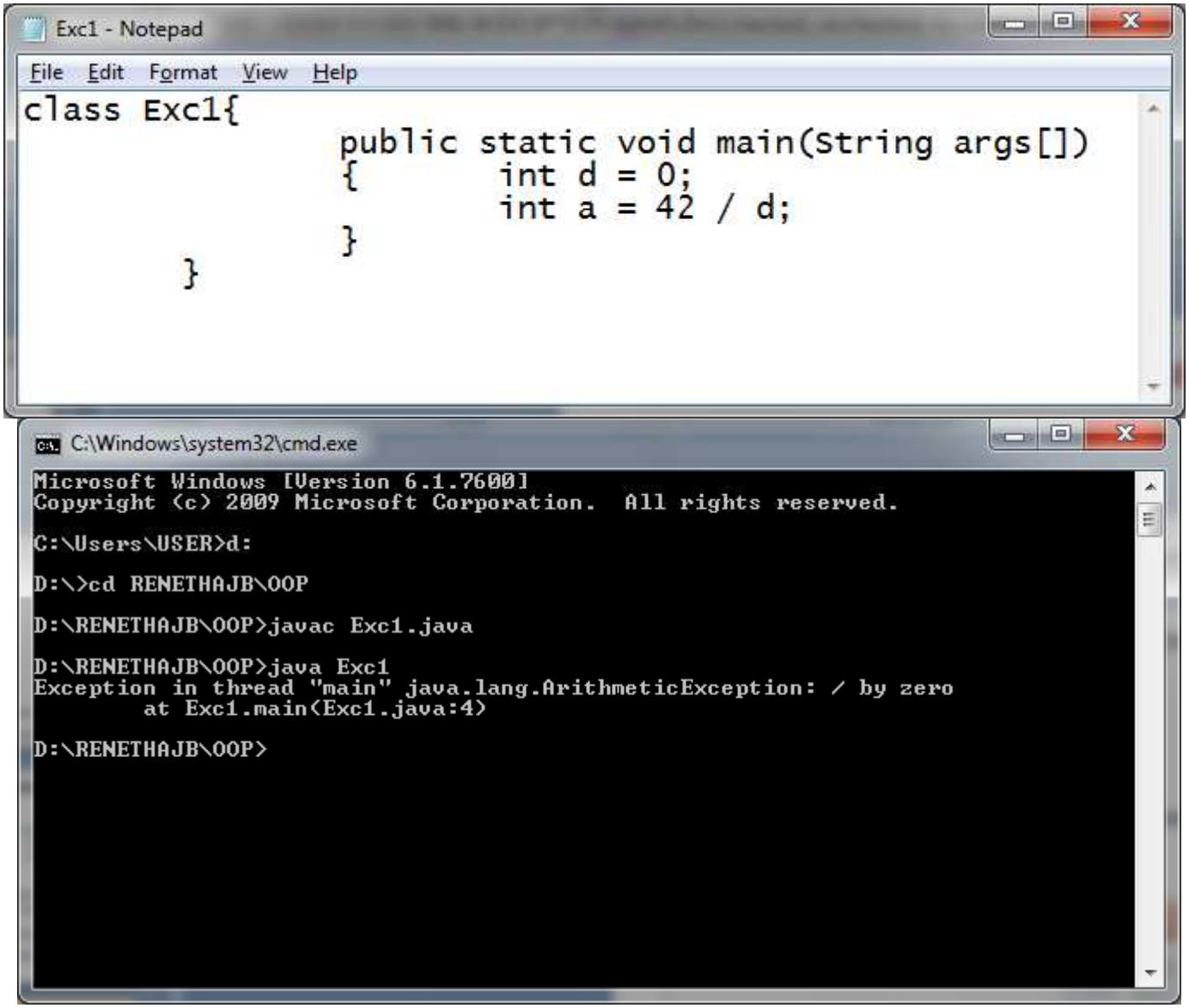
- Consider the program

```
Lineno.1        class Ex{
Lineno.2        public static void main(String args[])
Lineno.3                        {       int d = 0;
Lineno.4                                int a = 42 / d;
Lineno.5                        }
Lineno.6                }
```

- This small program causes a ***divide-by-zero error***((42/0)
- Java run time system constructs a new exception object and then *throws this exception.*
- *Th erogram stops by showing the following exception(run time erroe)*
- java.lang.ArithmeticException: / by zero at **Ex**.**main**(**Ex**.java:**4**)

- java.lang.**ArithmeticException**: / by zero at **Ex.main(Ex**.java:**4)**

- Here **Ex** is the class name , **main** is the method name,**;** **Ex.**java is the file name**;** and the exception is inline number **4.**
- These details are all included in the simple stack trace.
- The type of exception thrown is a subclass of Exception called **ArithmeticException** (describes what type of error happened.)


Exc1 - Notepad
File Edit Format View Help
class Exc1{
public static void main(String args[])
{
int d = 0;
int a = 42 / d;
}
}
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\USER>d:
D:\>cd RENETHAJB\OOP
D:\RENETHAJB\OOP>javac Exc1.java
D:\RENETHAJB\OOP>java Exc1
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Exc1.main(Exc1.java:4)
D:\RENETHAJB\OOP>

```
Lineno.1        class Exc1 {
Lineno.2        static void subroutine()
Lineno.3         { int d = 0;
Lineno.4          int a = 10 / d;
Lineno.5         }
Lineno.6        public static void main(String args[])
Lineno.7         { Exc1.subroutine();
Lineno.8         }
Lineno.9        }
```

- *java.lang.**ArithmeticException**: / by zero*

  *at Exc1.**subroutine**(Exc1.java:**4**)*

  *at Exc1.**main**(Exc1.java:**7**)*

try Block and *catch* *Clause*

- Benefits of exception handling
  - First, it allows us to **fix the error.**
  - Second, it **prevents** the program from **automatically terminating.**
- To **guard against** and **handle a run-time error**, simply enclose the code that we want to monitor inside a **try block.**
- Immediately ***after the try block***, there is a **catch clause** that **specifies the exception type** that we wish to catch . The catch block can process that exception..

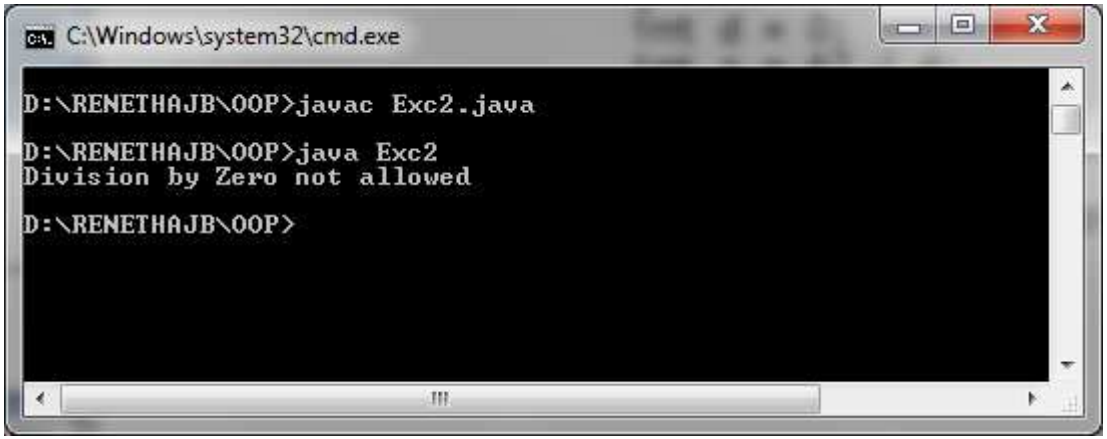
```
class Exc2{
        public static void main(String args[])
        {       try
                {
                int d = 0;
                int a = 42 / d;
                }
                catch(ArithmeticException ae)
                {
                System.out.println("Division by Zero not allowed");
                }
        }
}
```

```
class Exc2{
        public static void main(String args[])
        {       try
                {
                int d = 0;
                int a = 42 / d;
                }
                catch(ArithmeticException ae)
                {
                System.out.println("Division by Zero not allowed");
                }
        }
}
```

Exc2 - Notepad

File Edit Format View Help

```
class Exc2{
                public static void main(String args[])
                {       try
                        {
                        int d = 0;
                        int a = 42 / d;
                        }
                        catch(ArithmeticException ae)
                        {
                        System.out.println("Division by Zero not allowed");
                        }
                }
        }
```



try-catch block to handle division by zero exception


```
class Ex {
public static void main(String args[]) {
int d, a;
try {                                   // monitor a block of code.
     d = 0;
     a = 42 / d;
     System.out.println("This will not be printed.");
    }
    catch (ArithmeticException e)       // catch divide-by-zero error
    {
    System.out.println("Division by zero.");
    }
System.out.println("After catch statement.");
}
}
```

OUTPUT
Division by zero.
After catch statement.

Working of the program

- In this program the *System.out.println("This will not be printed.");* inside the try block is never executed because a = 42 / d;

- Once an exception is thrown, program control transfers out of the try block into the catch block.
  - i.e. catch is not "called" but controls goes out to catch when exception occurs, so execution never "returns" to the try block from a catch.
  - Thus, the line "This will not be printed." is not displayed.

try-catch (contd.)

- A try and its catch statement form a unit.
- The scope of the catch clause is restricted to those statements specified by the immediately preceding try statement.
  - Each catch block can catch exceptions in statements inside immediately preceding try block.
- A catch statement cannot catch an exception thrown by another try statement (except in the case of nested try statements).
- The statements that are protected by try must be surrounded by curly braces. (That is, they must be within a block.)
- We cannot use try on a single statement

try-catch Example

```
import java.util.Random;
class HandleError {
public static void main(String args[]) {
int a=0, b=0, c=0;
Random r = new Random();
for(int i=0; i<32000; i++) {
try {
      b = r.nextInt();
      c = r.nextInt();
      a = 12345 / (b/c);
    }
catch(ArithmeticException e)
{
      System.out.println("Division by zero.");
      a = 0;                    // set a to zero and continue
}
System.out.println("a: " + a);
} } }
```

Here **b** and **c** are random numbers .
If the value of b or c becomes zero then
a=12345./ (b/c) becomes
a=12345/0;
(Division by zero(ArithmeticException) will occur)
This statement is inside try block
So exception will be caught by catch and prints message ***Division by zero.***
and set the value of a to 0 and proceeds
NO RUNTIME ERROR!!

Displaying a Description of an Exception

- We can display this description in a **println( )** statement by simply passing the exception as an argument.

```
class Exc3{
        public static void main(String args[])
        {       try
                {
                int d = 0;
                int a = 42 / d;
                }
                catch(ArithmeticException ae)
                {
                System.out.println("The exception occurred is "+ae);
                }
        }
}
```

Exc3 - Notepad

File Edit Format View Help

```
class Exc3{
                public static void main(String args[])
                {       try
                        {
                        int d = 0;
                        int a = 42 / d;
                        }
                        catch(ArithmeticException ae)
                        {
                        System.out.println("The exception occurred is "+ae);
                        }
                }
        }
```

C:\Windows\system32\cmd.exe

```
D:\RENETHAJB\OOP>javac Exc3.java

D:\RENETHAJB\OOP>java Exc3
The exception occured is java.lang.ArithmeticException: / by zero

D:\RENETHAJB\OOP>
```

Reference

- Herbert Schildt, Java: **The Complete Reference, 8/e, Tata McGraw Hill, 2011**.