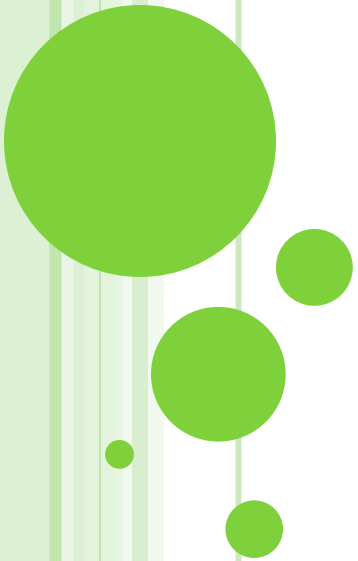


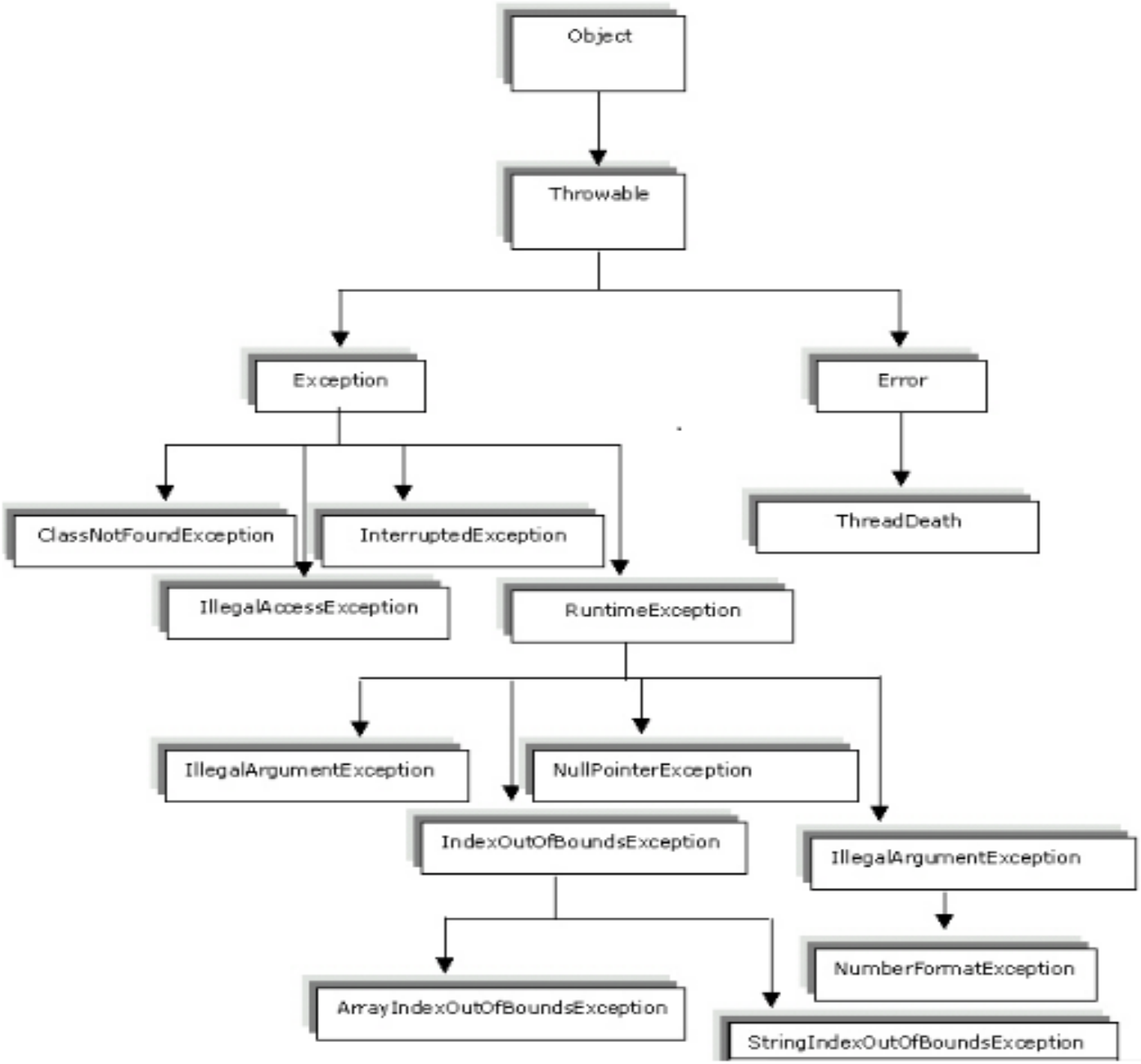
LECTURE

EXCEPTION HANDLING IN JAVA

Dr Rubeena DOOMUN
Open University of Mauritius

Exception

- An exception is a problem that arises during the execution of a program.
- When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally.
- Exceptions need to be handled.
- There are two types of exceptions:
  - Checked Exception
  - Unchecked Exception


Object
Throwable
Exception
Error
ClassNotFoundException
InterruptedException
ThreadDeath
IllegalAccessException
RuntimeException
IllegalArgumentException
NullPointerException
IndexOutOfBoundsException
IllegalArgumentException
NumberFormatException
ArrayIndexOutOfBoundsException
StringIndexOutOfBoundsException

CHECKED EXCEPTION

- Checked exceptions are checked at compile-time.
- These are also called as compile time exceptions.
- It means if a method is throwing a checked exception then it should handle the exception using **try-catch block** or it should declare the exception using **throws keyword**, otherwise the program will give a compilation error.

EXAMPLES OF CHECKED EXCEPTIONS

- ClassNotFoundException
- IllegalAccessException
- IOException
- SQLException

Example using keyword throws

```
import java.io.*;
class Example {
   public static void main(String args[]) throws IOException
   {
      FileInputStream fis = null;
      fis = new FileInputStream("B:/myfile.txt");
      int k;

      while(( k = fis.read() ) != -1)
      {
           System.out.print((char)k);
      }
      fis.close();
   }
}
```

Example using try catch block

```
import java.io.*;
class Example {
   public static void main(String args[])
   {
        FileInputStream fis = null;
        try{
            fis = new FileInputStream("B:/myfile.txt");
        }catch(FileNotFoundException fnfe){
            System.out.println("The specified file is not " +
                        "present at the given path");
         }
        int k;
        try{
            while(( k = fis.read() ) != -1)
            {
                System.out.print((char)k);
            }
            fis.close();
        }catch(IOException ioe){
            System.out.println("I/O error occurred: "+ioe);
         }
   }
}
```

UNCHECKED EXCEPTION

- An unchecked exception is an exception that occurs at the time of execution.
- Also known as **Runtime Exceptions**.
- These include programming bugs, such as logic errors or improper use of an API.
- Runtime exceptions are ignored at the time of compilation.

Examples of unchecked exception

- NullPointerException
- ArrayIndexOutOfBoundsException
- ArithmeticException
- IllegalArguementException

EXAMPLE

```
class Example {
        public static void main(String args[])
        {
                try{
                int arr[] ={1,2,3,4,5};
                System.out.println(arr[7]);
                }catch(ArrayIndexOutOfBoundsException e){
                        System.out.println("The specified index does not exist " +
                                        "in array. Please correct the error.");
                }
        }
}
```

KEYWORD THROW V/S THROWS

- **Throws** keyword is used to declare an exception and **throw** keyword is used to throw an exception explicitly.
- Syntax wise, **throw** is followed by an instance variable and **throws** is followed by exception class names.
- The keyword **throw** is used inside method body to invoke an exception and **throws** keyword is used in method declaration (signature).

Example using Throw keyword

```
....
static{
try {
throw new Exception("Something went wrong!!");
} catch (Exception exp) {
System.out.println("Error: "+exp.getMessage());
}
}
....
```

Example using throws keyword

```
public void sample() throws ArithmeticException{
 //Statements

.....

 //if (Condition : There is an error)
ArithmeticException exp = new ArithmeticException();
 throw exp;
...
}
```

More Examples

- When using **Throw keyword** in java, you cannot throw more than one exception but when using **throws keyword** you can declare multiple exceptions.

Throw:

```
throw new ArithmeticException("An integer should not be divided by zero!!")
throw new IOException("Connection failed!!")
```

Throws:

```
throws IOException, ArithmeticException, NullPointerException,
ArrayIndexOutOfBoundsException
```

TRY CATCH FINALLY BLOCK

```
try {
    //try block
    ...
    return success;
}
catch (Exception ex) {
    //catch block
    .....
    return failure;
}
finally {
    System.out.println("Inside finally");
}
```

FINALLY BLOCK

- The finally block will execute whether or not an exception is thrown in the corresponding try block.
- It will execute if a try block exits by using a return, break or continue statement or simply by reaching its closing right brace.
- If the application exits early from a try block by calling method *System.exit()*, then the finally block will not execute.

Labsheet

- Write a simple program in Java to demonstrate error handling for each exception listed below:
  - ArithmeticException
  - ArrayIndexOutOfBounds Exception
  - NumberFormat Exception

REFERENCES

- http://cs.lmu.edu/~ray/notes/paradigms/
- https://www.tutorialspoint.com/java/java_exceptions.htm
- https://beginnersbook.com/2013/04/java-exception-handling/
- https://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/
- https://beginnersbook.com/2013/04/difference-between-throw-and-throws-in-java/
- https://beginnersbook.com/2013/04/exception-handling-examples/