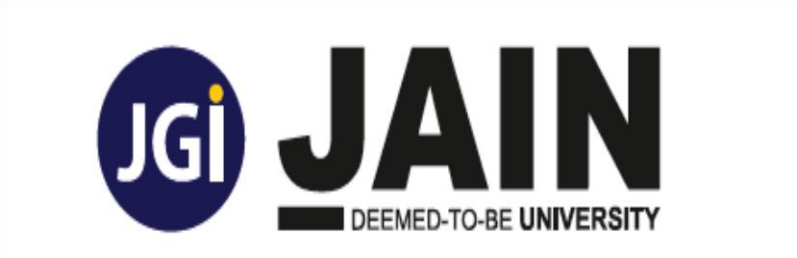



**School of Computer Science and IT**
**JAIN (DEEMED-TO-BE UNIVERSITY)**
**Department of Bachelor of Computer Applications**

# Module 3 Introduction to Packages and Exceptions

## Chapter 2 Exceptions and Exception Handling

# Exception handling Fundamentals

# Java Exception

Definition:

- A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That process may choose to handle the exception itself, or pass it on. In Java, these problems are termed as **exceptions**
- The flow of the program will get disrupted if an exception occurs or the program gets terminated abnormally without recommending any solutions. Hence, these exceptions should be handled properly and timely.

# Reasons for Exceptions to Occur

- If a user enters an indefensible data
- Searching a file which is to be opened but not in its track (not found)
- Losing a network connection in the middle of communication or the pragmatic machine run out of memory.

# Exception Handling

Java exception handling is accomplished via five keywords: **try, catch, throw, throws,** and **finally**.

- Program statements that you want to monitor for exceptions are received within a try block.
- . If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch) and handle it in some rational manner.
- The Java run-time system automatically throws System-generated exceptions.
- To manually throw an exception, just use the keyword throw. Any exception that is thrown out of a method must be specified as such in a throws clause.
- Any code that absolutely must be executed after a try block completes is put in a final block.

# General Structure of Exception Handling Block

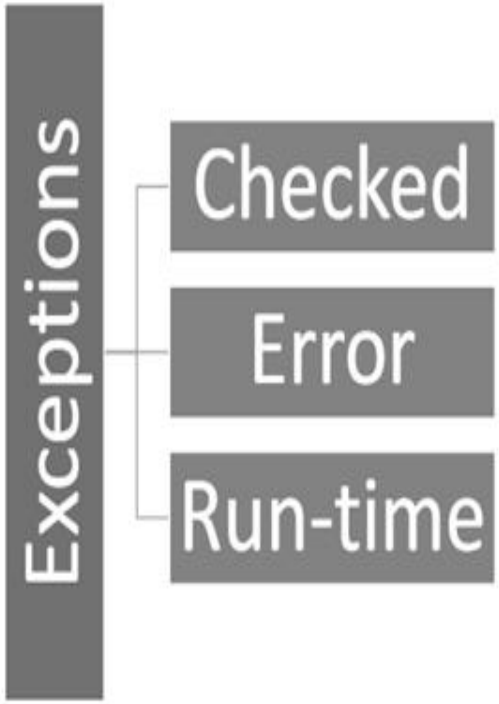
```
try {
// block of code to observe for errors
}
catch (Exception1 ex) {
// exception handler for Exception1
}
catch (Exception2 ex) {
// exception handler for Exception2
}
// ...
finally {
// block of code will be execute after try block ends
}
```

# Exception Types

# Exception Types

There are three types of exceptions occur while exception handling.

They are as follows:



# Checked Exception

Checked exceptions are the one of the kinds of exceptions that either must be caught or declared in the method in which it is thrown.

## Example:

Assume a user is using FileReader class to read a character file.

**publicFileReader(String fileName)**

**throwsFileNotFoundException** //FileReader constructor is invoked and the exception is thrown//

# Error Exceptions

The error is the second kind of exception. When this sort of exception occurs, JVM (Java Virtual Machine) will create an exception object. These objects all derive from' throwable class where it has two main subclasses, the first one is**Error** and second one is **Exception.**

## Example:

When hardware fails, JVM might run out of resources but it is possible for an application to catch the error and notify the user, saying the application will close down until the underlying problem is dealt with.

# Run Time Error

A program may contain so many errors like “an element of an array does not exist, logic error, termination errors, etc." They may cause because of human error or may be a programmatic error like method called will return a null value, etc.

When a program executes with these errors an exception occurs which is termed as run time exception.

# Exception Handling Mechanism in Java

- Exception handling mechanism provides a clear way of checking errors without modifying the code. Also used to identify the errors directly by providing signals (error statement) without making any side effects to the methods or functions of the program.
- Java exceptions are objects and they are primarily termed as checked exceptions. That is, the compiler automatically verifies the method (declared by the programmer) and throws the exception using the **throwable function**. There are many ways to handle the exceptions.

  - Using try and catch block
  - Multiple catch clauses
  - Nested try statements
  - The throw keyword
  - The throws keyword
  - The finally block

# Using try and catch Block

- To defend against and examine a run-time error, just enclose the code that you want to monitor inside a try block.
- Immediately following the try block, include a catch clause that specifies the exception type that you wish to capture.
- To explain how easily this can be done, the following program includes a try block and a catch clause that processes the ArithmeticException generated by the division-by-zero error:

```
class Except2 {
public static void main(String args[]) {
int p, q;
try { // monitor a block of code.
p = 0;
q = 42 / p;
System.out.println("It will not be printed.");
} catch (ArithmeticException e) { // catch divide-by-zero error
System.out.println("Division by zero.");
}
System.out.println("After catch block statement.");
}
}
```

This program generates the following output:

```
Division by zero.
```

# Try Block

The Try block is a cooperative block of catch. Without catch block, try block is illogical. A catch block executes only if an exception of a particular type occurs within the try block. A try block is always followed by a catch block.

## Syntax:

```
try
 {
   ..... //statement;
   .....//statement;
 }
```

# Catch Block

The catch block is associated with the try block. It handles the type of exception indicated by its argument. Here, the argument type of exception inherits from the **Throwable** class.

**Syntax:**

```
try
 {
   Catch (ExceptionType name)
      {
      ..............// error handling code
      }
  Catch (ExceptionType name) {
    }
}
```

# Multiple Catch Clause

- Some cases in Multiple Catch clause, more than one exception could be raised by a single piece of code.
- To handle this type of situation, you can specify two or more catch clauses, each detecting a different kind of exception.
- When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed.
- After one catch statement compilation, another are bypassed, and execution continues after the try/catch block.

**Syntax:**

```
Catch (IOException) {
//error statement;
throw;
catch (IOException) {
//error statement;
throw;
   }
```

# Nested try Statement

- The try statement can be nested statement. It means a try statement can be inside the block of another try.
- Each time a try statement is entered, the context of that exception is pushed on the stack. If an inner/personal try statement does not have a catch handler for a particular exception, the stack is unwound and the next try statement's catch handlers are inspected for a match.
- It extends until one of the catch statements succeeds, or until all of the nested try statements are exhausted. If no catch statement matches, then the Java run-time system will handle the exception.
- Nested try statement comprises of a try block within another try block. It is used in some situation where a part of a block may cause one error and the entire clock itself may cause another error.

# Nested try Syntax

```
try
{
statement a;
statement b;
try
        {
statement a;
statement b;
        }
}
Catch (Exception e)
        {
        }
    Catch (Exception e)
        {
        }
```

# Throw Keyword

Throw keyword is used to declare an exception. It is used to give information to the programmer that exception may occur so it will be finer to provide an exceptional handling code so that the flow of the program does not get affected.

**Syntax:**

```
return_type method_name
() throws exception_class_name
{
//code of method
}
```

# The finally Block

- The final block is used to handle an unexpected exception in a program. It always executes when the try block exits.
- It uses the break statement to avoid programmers to use the clean up code in the program. But it is advisable to use clean up code even when no exceptions are anticipated. The **final** keyword is designed to address this contingency.
- The "**finally**" keyword generates a block of code that will be done after a try or catch block has completed and before the code following the try or catch block. The final block will execute either or not an exception is thrown. If an exception is thrown, the final block will perform even if no catch statement matches the exception.

# Java Built-in Exceptions

Java platform provides several exception classes under a package **java.lang**. The most general form of exception is the Runtime exception. List of Java built-in exceptions are given below:

| Exception | Description |
|---|---|
| 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. |
| 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. |
| UnsupportedOperationException | An unsupported operation was encountered. |

# Creating Exception Subclasses

In Java, the user can create their exception subclass by simply extending Java option java class. There are some steps to be followed to create exception subclasses. They are:

**Step 1:**

Define a constructor in the program where an exception subclass is to be created.

**Step 2:**

Another way of creating an exception subclass is that an user can override the tostring( ) function to display your customised message on the catch.

# Exception Subclasses

By default, no methods are defined by exceptions . Instead they are defined by the method Throwable available to them. Some of them are shown below:

- ThrowablegetCause( ) – returns the exception that underlies the current exception
- String getLocalisedMessage( ) – returns a localized description
- String getMessage( ) – returns a description of the exception

# Example Program to Create Customise Exception Types

```
classMyExceptionextends Exception {
privateint detail;
MyException(int p)
{
detail = p;
}
public String toString()
 {
return "MyException[" + detail + "]";
 }
}
publicclass Main{staticvoid compute(int a)
 throwsMyException{System.out.println("Called compute(" + p + ")");
```

To be continued in next slide..

```
if (p > 10)
thrownewMyException(a);
System.out.println("Simple exit");
 }
publicstaticvoid main(String args[])
  {try
{compute(2);
compute(21);    }
catch (MyException e)
 {System.out.println("Caught " + e);    }  }}
```

**Output:**

Simple Exit

# Types of Creating Exception Subclass

Creating exception subclass can be classified in two types. They are:

- Chained exceptions
- Using exceptions

# Chained Exceptions

Exception throwing another exception is called as chained exceptions. The first exception is the cause of the second one and it is very useful to know when it occurs.

Methods and constructors in throwable that supports chained exceptions are as follows:

- ThrowablegetCause( )
- ThrowableinitCause (Throwable)
- Throwbale (String, Throwbale)
- Throwable (Throwable)

# Chained Exception- Example

The following example shows how to use a chained exception.

```
try {

} catch (IOException e) {
throw new SampleException("Other IOException", e);
}
```

In the above example program, new SampleException is created with the original IOException and the chain exception is thrown up.

# Using Exception

An exception subclass can be used as the parent class of another exception called as linked exceptions. These exceptions are inappropriate because they are either too differentiate (specified for special function) or completely unrelated to another exception.

# Summary

- ✓ In Java, errors are termed as exceptions.
- ✓ Checked exceptions are declared in the method in which it is thrown.
- ✓ Try block is associated with a catch block.
- ✓ Java exceptions are objects and they are primarily termed as checked exceptions.
- ✓ The final block is used to handle an unexpected exception in a program.