**Exception Handling – Part-02 – Default Exception Handling**

* **Default Exception Handling:**

Story:

To remember the default exception handling recall the below story.

Two friends are travelling through a bike from Hyderabad to Vijayawada. In between they met with an accident the truck hit them. Someone has to inform this to 108 or nearby police station.

Then the police or the 108 persons will come and take the necessary action.

1. Inside a method if any exception occurs, the method in which it is raised is responsible to create exception object by including the following information.
2. Name of exception.
3. Description of exception.
4. Location of exception occurred (Stack Trace).

1. After creating exception object method handovers that object to the JVM.
2. JVM will check whether the method contains any exception handling code or not. If the method doesn’t contain exception handling code then JVM terminates that method abnormally and removes the corresponding entry from the stack.
3. Then JVM identifies caller method and checks whether caller method contains any handling code or not. If the caller method doesn’t contain handling code, then JVM terminates that caller method also abnormally and removes the corresponding entry from the stack. This process will be continued until main method and if the main method also doesn’t contain handling code then JVM terminates main method also abnormally and removes corresponding entry from the stack.
4. Then JVM handovers responsibility of exception handling to Default Exception Handler, which is the part of JVM.
5. Default Exception Handler prints exception information in the following format and terminates program abnormally.

Exception in thread “xxx”: Name of exception: Description

Stack trace

Example:

class Test{

public static void main(String[] args){

doStuff();

}

public static void doStuff(){

doMoreStuff();

}

public static void doMoreStuff(){

System.out.println(10/0);

}

}

Output:

Exception in thread “main”: java.lang.AE: / by 0

at Test.doMoreStuff()

at Test.doStuff()

at Test.main()

class Test{

public static void main(String[] args){

doStuff();

}

public static void doStuff(){

doMoreStuff();

System.out.println(10/0);

}

public static void doMoreStuff(){

System.out.println(“Hello”);

}

}

Output:

Hello

Exception in thread “main”: java.lang.AE: / by 0

at Test.doStuff()

at Test.main()

Note: In this example one method (doMoreStuff) terminated normally and other two methods (doStuff and main) terminated abnormally.

class Test{

public static void main(String[] args){

doStuff();

System.out.println(10/0);

}

public static void doStuff(){

doMoreStuff();

System.out.println(“Hi”);

}

public static void doMoreStuff(){

System.out.println(“Hello”);

}

}

Output:

Hello

Hi

Exception in thread “main”: java.lang.AE: / by 0

at Test.main()

Note: In this example two method (doMoreStuff and doStuff) terminated normally and other method (main) terminated abnormally.

* **Note:**

In a program if at least one method terminates abnormally then the program termination is abnormal termination.

If all methods terminated normally then only program termination is normal termination.

* **Exception Hierarchy:**

Throwable acts as root for java exception hierarchy.

Throwable class defines two child classes:

Exception

Error

Exception:

Most of the times exceptions are caused by our program and these are recoverable. For example, if our programming requirement is to read data from remote located at London at runtime if remote file is not available then we will get runtime exception saying FileNotFoundException.

If FileNotFoundException occurs we can provide local file and continue rest of the program normally.

try{

Read data from remote file located at London.

} catch(FileNotFoundException e){

Use local file and continue rest of the program normally.

}

Note: Recoverable means, if something happens no need to worry we can have a hold and do something as an alternate.

Error:

Most of the times errors are not caused by our program and these are due to lack of system resources. Errors are non-recoverable.

For example, if OutOfMemoryError occurs being a programmer we can’t do anything and the program will be terminated abnormally.

System admin or server admin is responsible to increase head memory.



