1. What is the Throwable class? Differentiate between Exception and Error class.

Ans: The Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class are thrown by the Java Virtual machine or can be thrown by the Java throw statement. It has two subclasses :- Exception and Error.

|  |  |
| --- | --- |
| Exception | Error |
| Used for exceptional conditions that user programs should catch | Defines exceptions that are not expected to  be caught under normal circumstances by the program |
| Can be handled at runtime | Cannot be handled at runtime |
| Can be handled by using try and catch block | Program execution will be terminated |
| Exceptions are of two types: Checked and Unchecked | Error are by default unchecked exception |
| Are of type java.lang.Exception | Are of type java.lang.Error |
| Exceptions are related to applications | Errors are related to the environment in which the application is running |

1. What is the default behaviour if a RuntimeException occurs without using try and catch? Give an example.

Ans: Java being an object oriented programming language, whenever an error occurs while executing a statements, creates an exception object and then the normal flow of the program halts and JRE tries to find someone that can handle the raised exception. The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception occured , type of exception etc. When the exception occurs in a method , the process of creating the exception object and handling it over to runtime environment is called “throwing the exception”.

This small program includes an expression that intentionally causes a divide-by-zero error:

class Exc0 {

public static void main(String args[])

{

int d = 0;

int a = 42 / d;

}

}

When the Java run-time system detects the attempt to divide by zero, it constructs a new

exception object and then *throws* this exception. This causes the execution of **Exc0** to stop,

because once an exception has been thrown, it must be *caught* by an exception handler and

dealt with immediately. In this example, we haven’t supplied any exception handlers of our

own, so the exception is caught by the default handler provided by the Java run-time

system. Any exception that is not caught by your program will ultimately be processed by

the default handler. The default handler displays a string describing the exception, prints a

stack trace from the point at which the exception occurred, and terminates the program.

1. Can we catch two or more exceptions occurring within the same try block?

Ans: Yes, it is possible to catch two or more exceptions within the same try block. In order to handle this kind of situation we need to specify two or more **catch** clauses, each catching a different type 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.

// Demonstrate multiple catch statements.

class MultipleCatches {

public static void main(String args[]) {

try {

int a = args.length;

System.out.println("a = " + a);

int b = 42 / a;

int c[] = {1};

c[42] = 99;

} catch(ArithmeticException e) {

System.out.println("Divide by 0: " + e);

} catch(ArrayIndexOutOfBoundsException e) {

System.out.println("Array index oob: " + e);

}

System.out.println("After try/catch blocks.");

}

}

When you use multiple **catch** statements, it is important to remember that exception subclasses must come before any of their superclasses. This is because a **catch** statement that uses a superclass will catch exceptions of that type plus any of its subclasses. Thus, a subclass would never be reached if it came after its superclass. Further, in Java, unreachable code is an error.