Exception handling essay

Exception is an abnormal condition that arises when executing a program. In the languages that do not support exception handling, errors must be checked and handled manually, usually through the use of error codes. In contrast, Java: 1) provides syntactic mechanisms to signal, detect and handle errors 2) ensures a clean separation between the code executed in the absence of errors and the code to handle various kinds of errors 3) brings run-time error management into object-oriented programming

Exception Handling An exception is an object that describes an exceptional condition (error) that has occurred when executing a program. Exception handling involves the following: 1) when an error occurs, an object (exception) representing this error is created and thrown in the method that caused it 2) that method may choose to handle the exception itself or pass it on 3) either way, at some point, the exception is caught and processed

Exception Sources Exceptions can be: 1) generated by the Java run-time system Fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. 2) manually generated by programmer’s code Such exceptions are typically used to report some error conditions to the caller of a method.

Exception Constructs Five constructs are used in exception handling: 1) try – a block surrounding program statements to monitor for exceptions 2) catch – together with try, catches specific kinds of exceptions and handles them in some way 3) finally – specifies any code that absolutely must be executed whether or not an exception occurs 4) throw – used to throw a specific

exception from the program 5) throws – specifies which exceptions a given method can throw

Exception-Handling Block General form: try { … } catch(Exception1 ex1) { … } catch(Exception2 ex2) { … } … finally { … } where: 1) try { … } is the block of code to monitor for exceptions 2) catch(Exception ex) { … } is exception handler for the exception Exception 3) finally { … } is the block of code to execute before the try block ends

Exception Hierarchy All exceptions are sub-classes of the build-in class Throwable. Throwable contains two immediate sub-classes: 1) Exception – exceptional conditions that programs should catch The class includes: a) RuntimeException – defined automatically for user programs to include: division by zero, invalid array indexing, etc. b) use-defined exception classes 2) Error – exceptions used by Java to indicate errors with the runtime environment; user programs are not supposed to catch them

Uncaught Exception What happens when exceptions are not handled? 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 throws this object. This will cause the execution of Exc0 to stop – once an exception has been thrown it must be caught by an exception handler and dealt with.

Default Exception Handler As we have not provided any exception handler, the exception is caught by the default handler provided by the Java run-time system. This default handler: 1) displays a string describing the exception, 2) prints the stack trace from the point where the exception occurred 3)

terminates the program java. lang. ArithmeticException: / by zero at Exc0. main(Exc0. java: 4) Any exception not caught by the user program is ultimately processed by the default handler.

Stack Trace Display The stack trace displayed by the default error handler shows the sequence of method invocations that led up to the error. Here the exception is raised in subroutine() which is called by main(): class Exc1 { static void subroutine() { int d = 0; int a = 10 / d; } public static void main(String args[]) { Exc1. subroutine(); } }

Own Exception Handling Default exception handling is basically useful for debugging. Normally, we want to handle exceptions ourselves because: 1) if we detected the error, we can try to fix it 2) we prevent the program from automatically terminating Exception handling is done through the try and catch block.

Try and Catch 1 Try and catch: 1) try surrounds any code we want to monitor for exceptions 2) catch specifies which exception we want to handle and how. When an exception is thrown in the try block: try { d = 0; a = 42 / d; System. out. println(“ This will not be printed.”); }

Try and Catch 2 control moves immediately to the catch block: catch (ArithmeticException e) { System. out. println(“ Division by zero.”); } The exception is handled and the execution resumes. The scope of catch is restricted to the immediately preceding try statement it cannot catch exceptions thrown by another try statements.

Try and Catch 3 Resumption occurs with the next statement after the try/catch block: try { … } catch (ArithmeticException e) { … } System. out. println(“ After catch statement.”); Not with the next statement after a = 42/d; which caused the exception! a = 42 / d; System. out. println(“ This will not be printed.”);

Catch and Continue 1 The purpose of catch should be to resolve the exception and then continue as if the error had never happened. Try/catch block inside a loop: import java. util. Random; class HandleError { public static void main(String args[]) { int a= 0, b= 0, c= 0; Random r = new Random();