**PREREQUISITES**

The reader should have minimum knowledge of

* Basic concepts of Object Oriented Programming

**Exception Handling**

**Points to remember**

**Catching an Exception Using try and catch**

* An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
* There are 3 main advantages for exception handling

1. Separates error handling code from “regular” code
2. Propagating errors up the call stack (without tedious programming)
3. Grouping error types and error differentiation

* Exceptions come in two flavors: checked and unchecked.
* Checked exceptions include all subtypes of Exception, *excluding* classes that

extend RuntimeException.

* Checked exceptions are subject to the *handle or declare* rule; any method that *might* throw a checked exception (including methods that invoke methods that can throw a checked exception) must either declare the exception using the throws keyword, or handle the exception with an appropriate *try/catch*.
* An exception causes a jump to the end of try block. If the exception occurred in a method called from a try block, the called method is abandoned.
* If there’s a catch block for the occurred exception or a parent class of the exception, the exception is now considered handled.
* If there’s no code in try block that may throw exceptions specified in the catch blocks, compiler will produce an error.
* If there was no exception or the exception was handled, execution continues at the statement after the try/catch/finally blocks.
* If the exception is not handled, the process repeats looking for next enclosing try block up the call hierarchy. If this search reaches the top level of the hierarchy (the point at which the thread was created), then the thread is killed and message stack trace is dumped to System.err.
* Subtypes of Error or RuntimeException are unchecked, so the compiler doesn’t enforce the handle or declare rule. You’re free to handle them, and you’re free to declare them, but the compiler doesn’t care one way or the other hence (for example **Java.lang.RuntimeException** and **java.lang.Error** need not be handled or declared).
* finally and catch can come only with try, they cannot appear on their own.
* At least one ‘catch’ block or one ‘finally’ block must accompany a ‘try’ statement. If all 3 blocks are present, the order is important. (try/catch/finally)
* If you use an optional finally block, it will always be invoked, regardless

of whether an exception in the corresponding *try* is thrown or not, and

regardless of whether a thrown exception is caught or not.

* Regardless of whether or not an exception occurred or whether or not it was handled, if there is a finally block, it’ll be executed always. (Even if there is a return statement in try block).
* Just because finally is invoked does not mean it will complete.
* The only exception to the *finally*-will-always-be-called rule is that a *finally* will *not* be invoked if the JVM shuts down. That could happen if code fromthe try or catch blocks calls System.exit(), in which case the JVMwill not start your finally block.Code in the finally block could itself raise an exception or issue a System.exit().
* Uncaught exceptions propagate back through the call stack, starting from the method where the exception is thrown and ending with either the first method that has a corresponding catch for that exception type or a JVM shutdown.
* You can create your own exceptions, normally by extending Exception or one

of its subtypes. Your exception will then be considered a checked exception, and the compiler will enforce the handle or declare rule for that exception.

* Use throw new xxxException() to throw an exception. If the thrown object is null, a NullPointerException will be thrown at the handler.
* If an exception handler re-throws an exception (throw in a catch block), same rules apply. Either you need to have a try/catch within the catch or specify the entire method as throwing the exception that’s being re-thrown in the catch block. Catch blocks at the same level will not handle the exceptions thrown in a catch block – it needs its own handlers.
* The method fillInStackTrace() in Throwable class throws a Throwable object. It will be useful when re-throwing an exception or error.
* All catch blocks must be ordered from most specific to most general.

For example, if you have a catch clause for both IOException and Exception, you must put the catch for IOException first (in order, top to bottom in your code). Otherwise, the IOException would be caught by catch(Exception e), because a catch argument can catch the specified exception or any of its subtypes! The compiler will stop you from defining catch clauses that can never be reached (because it sees that the more specific exception will be caught first by the more general *catch*).

* An overriding method may not throw a checked exception unless the overridden method also throws that exception or a super-class of that exception. In other words, an overriding method may not throw checked exceptions that are not thrown by the overridden method. If we allow the overriding methods in sub-classes to throw more general exceptions than the overridden method in the parent class, then the compiler has no way of checking the exceptions the sub-class might throw. (If we declared a parent class variable and at runtime it refers to sub-class object) This violates the concept of checked exceptions and the sub-classes would be able to by-pass the enforced checks done by the compiler for checked exceptions. This should not be allowed.

**Tips**

* All objects of type java.lang.Exception are checked exceptions. (Except the classes under java.lang.RuntimeException) If any method that contains lines of code that might throw checked exceptions, compiler checks whether you’ve handled the exceptions or you’ve declared the methods as throwing the exceptions. Hence the name checked exceptions.
* The Java language requires that methods either catch or specify all checked exceptions that can be thrown within the scope of that method.
* **System.exit() and error conditions are the only exceptions where finally block is not executed.**

**Best Practices**

* Be specific while handling the exception in the catch block.
* Be specific while throwing exception in the throws clause.
* Do not use Exception Handling to control programming flow.
* Use exception handling generously-Very little overhead is imposed by using exception handling mechanism unless an exception occurs. But when an exception occurs it imposes an overhead in terms of execution time.
* Always use the finally block to release the resources to prevent resource leaks.
* Handle exceptions locally wherever possible.
* Do not use Exception handling in loops.

**Example 4.1E (M) Level: 1**

Purpose

Explaining the importance of Exception Handling.

Code

1**.class** Divider {

2. **public** **static** **void** main(String args[]) {

3. **int** quotient;

4. **int** dividend = 10;

5. **int** divisor = 0;

*/\* Below we are trying to divide the number 10 by 0 which is undefined.This is same with the JVM and it becomes an abnormal condition and the program is terminated and the abnormal condition is raised as Exception as shown in output \*/*

6.quotient = dividend / divisor;

7. System.out.println("Quotient: " + quotient);

8. }

9.}

Output

Exception in thread "main" java.lang.ArithmeticException: / by zero

at Divider.main(Divider.java:6)

Check your understanding

1. The above program terminates abruptly because of ……….

 Check: Do you now understand

* + Why to deal an exception.

**Example 4.2E (M) Level: 1**

Purpose

Explaining the concept of Exception Handling (using try-catch block).

Code

1. **class** Divider {
2. **public** **static** **void** main(String args[]) {
3. **int** quotient;
4. **int** dividend = 10;
5. **int** divisor = 0;

*/\* All the code that is to be kept under the supervision for the occurrence of an Exception is kept in the try block. In the catch block we have to handle the Exception. In this way even if an Exception is raised there will be no termination of program. And the statements after catch block will be executed normally. \*/*

1. **try** {
2. quotient = dividend / divisor;
3. System.out.println("Quotient: " + quotient);
4. **catch**(Exception e) {
5. System.out.println("Division by Zero");
6. *// System.out.println("Exception: " + e);*
7. }
8. System.out.println("This statement is after catch block. ");
9. }
10. }

Output

Division by Zero

This statement is after catch block.

Check your understanding

1. The above class explains the concept of ………………….. (see purpose).
2. The class consists of two blocks namely…………………… and …………………… for handling exception (see line no. 6, 9).
3. The exception occurs in ……………….. block ( see line no. 6 to 9).
4. The ………………………. block handles the exception occurred in try block.( see line no. 9 to 12)
5. Which statement is executed even after the occurrence of exception? ( see line no. 13)

 Check: Do you now understand

* + What is exception handling
  + How to deal an exception
  + What are try and catch block
  + Functions of try and catch block

**Example 4.3E (C) Level: 2**

Purpose

Alternative method to display error message (using StackTrace).

Code

1. **public** **class** MultiCatchDemo {
2. **public** **static** **void** main (String args[]) {
3. **try** {
4. **double** d = Double.valueOf(args[0]).doubleValue();
5. System.out.println (d);
6. } **catch** (ArrayIndexOutOfBoundsException e) {

*/\* If the user do not give any command line argument the control is*

*transferred to this block as we are trying to access args[0] \*/*

1. e.printStackTrace(System.err);  *// prints the error message*
2. **return**;
3. } **catch** (NumberFormatException e) {

*/\* If the user enters a value that cannot be converted to a double as command line argument then the control is transferred to this block as we are trying to convert the value at command line argument to a double value \*/*

1. e.printStackTrace(System.err); *// prints the error message*
2. **return**;
3. }
4. }
5. }

Output

Output 1: when no argument is given

C:\>java MultiCatchDemo

java.lang.ArrayIndexOutOfBoundsException: 0

at MultiCatchDemo.main(MultiCatchDemo.java:4)

Output 2: When ‘abc’ is given as argument

C:\>java MultiCatchDemo abc

java.lang.NumberFormatException: For input string: "abc"

at java.lang.NumberFormatException.forInputString(NumberFormatException.

java:48)

at java.lang.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1

213)

at java.lang.Double.valueOf(Double.java:184)

at MultiCatchDemo.main(MultiCatchDemo.java:4)

Output 2: When ‘123’ is given as argument

C:\IO>java MultiCatchDemo 123

123.0

Check your understanding

1. When an error occurs, exception is raised and it is caught in…………. block (see line no. 6, 9). The error is thrown as an object to this block.
2. The error message is handled and printed by ……………… block. (see line no. 6, 9)
3. The error message is printed by using method …………………………. (see line no. 7, 10)
4. In the output, you can observe that the error message gives the name of exception, class, method and line number.

 Check: Do you now understand

* + How an object is created in catch block
  + What the object contains
  + The use of printStackTrace()
  + How the output appears while using this method

**Example 4.4E (M) Level: 2**

Purpose

Building an erroneous program using multiple catch blocks.

Code

*/\* The program would not run as you expected because there is a pitfall in the usage of multiple catch blocks.\*/*

1. **public** **class** MultiCatchDemo {
2. **public** **static** **void** main (String args[]) {
3. **try** {
4. **double** d = Double.valueOf(args[0]).doubleValue();
5. System.out.println (d);
6. **if**(d == 100.0)
7. System.out.println(d / 0); *//divding by zero*
8. } **catch**(Exception e) {
9. System.out.println("Division by zero");
10. **return**;
11. } **catch** (ArrayIndexOutOfBoundsException e) {
12. System.out.println (“An argument is required.");
13. **return**;
14. } **catch** (NumberFormatException e) {
15. System.out.println("the argument must be a real number.”);
16. **return**;
17. }
18. }
19. }

Output

E:\IO>javac MultiCatchDemo.java

MultiCatchDemo.java:11: exception java.lang.ArrayIndexOutOfBoundsException has already been caught

} catch (ArrayIndexOutOfBoundsException e) {

^

MultiCatchDemo.java:14: exception java.lang.NumberFormatException has already been caught

} catch (NumberFormatException e) {

^

2 errors

Check your understanding

1. The important point to be noted when we use multiple catch blocks is that a sub class of exception must come before its super class in a series of catch statements.
2. Here, ………………………………… is a super class of exception and appears first in the catch block (see line no. 8)
3. This indicates that the ……………………. class catches the error. Thereby, the above program gives a compilation error as shown in output.

 Check: Do you now understand

* + When do compilation error occurs
  + The sequence of use of Exception class and its sub classes

**Example 4.5E (M) Level: 1**

Purpose

Illustrating the use of finally block in Exception Handling.

Code

1. **class** Divider {
2. **public** **static** **void** main(String args[]) {
3. **int** quotient;
4. **int** dividend = 10;
5. **int** divisor = 0;
6. **try** {
7. quotient = dividend / divisor;
8. System.out.println("Quotient: " + quotient);
9. } **catch**(Exception e) {
10. System.out.println("Division by Zero");
11. *// System.out.println("Exception: " + e);*

} **finally** { // *finally block is always executed.*

1. System.out.println("Finally Block is always Executed. ");
2. }
3. System.out.println("This statement is after catch block. ");
4. }
5. }

Output

Division by Zero

Finally Block is always Executed.

This statement is after catch block.

Check your understanding

1. In the above class the blocks for exception handling are try, catch and ………………………… (see line no. 12)
2. The finally block is written after try-catch block. There can be a try block without catch block, but with a ……….. block (see line no. 12)
3. Even if the catch block is not executed, the final block is executed. There cannot be a final block without a …………….block. (see line no6)
4. finally block is used to close the files or release some resource before the program is terminated.

 Check: Do you now understand

* + What is finally block
  + How does it work
  + Its importance in a code

**Example 4.6E (M) Level: 1**

Purpose

Using **throw** keyword in Exception Handling within a method.

Code

1. **class** ThrowDemo {
2. **public** **static** **void** main(String args[]) {
3. **try** {
4. *// throw keyword is used to raise an exception by the user*
5. **throw** **new** NullPointerException("Throw Demo");
6. } **catch**(NullPointerException e) {
7. System.out.println("Exception: " + e);
8. }
9. }
10. }

Output

Exception: java.lang.NullPointerException: Throw Demo

Check your understanding

1. A user can raise an exception by using keyword…………………. within a method. (see line no. 5)
2. At this point an exception is thrown and is caught by the catch block.

 Check: Do you now understand

* + What is **throw** keyword
  + How does it work
  + Its importance in a code

**Example 4.7E (S) Level: 3**

Purpose

Explaining how to rethrow an Exception which occurs in a program.

Code

1. **class** ThrowDemo {
2. **static** **void** function() {
3. **try** {
4. *// throw keyword is used to raise an exception by the user*
5. **throw** **new** NullPointerException("Throw Demo");
6. } **catch**(NullPointerException e) {
7. System.out.println("Exception in function: " + e);
8. **throw** e; *//rethrowing same exception*
9. }
10. }
11. **public** **static** **void** main(String args[]) {
12. **try** {
13. function();
14. } **catch**(NullPointerException e) {
15. System.out.println("Exception in main: " + e);
16. }
17. }
18. }

Output

Exception in function: java.lang.NullPointerException: Throw Demo

Exception in main: java.lang.NullPointerException: Throw Demo

Check your understanding

1. An exception is raised by the user in ……………. block by using the keyword …………….. (see line no. 3, 5)
2. In the method function(), the catch block catches the exception thrown by try block. The same exception is ones again thrown as in line no. 8. It is then caught by catch block of the main function.

 Check: Do you now understand

* + What we mean by rethrowing an exception
  + How does it work
  + How to implement it

**Example 4.8E (M) Level: 3**

Purpose

Explaining an erroneous program while using **throw** with a Checked Exception type.

Code

1. **class** ThrowDemo {
2. **static** **void** function() {
3. *// throwing a checked exception from a method results to compilation error*
4. **throw** **new** IllegalAccessException("Throw Demo");
5. }
6. **public** **static** **void** main(String args[]) {
7. **try** {
8. function();
9. } **catch**(IllegalAccessException e) {
10. System.out.println("Exception in main: " + e);
11. }
12. }
13. }

Output

ThrowDemo.java:4: unreported exception java.lang.IllegalAccessException; must be

caught or declared to be thrown

throw new IllegalAccessException("Throw Demo");

^

ThrowDemo.java:9: exception java.lang.IllegalAccessException is never thrown in

body of corresponding try statement

} catch(IllegalAccessException e) {

^

2 errors

Check your understanding

1. Here**,** we are using **throw** to throw an exception occurring within a method. But the exception is of ……………………………… type. (see line no. 4)
2. The compiler does not understand because it is not being reported regarding the occurrence of such an exception.
3. Thereby, it is not being caught by catch block and results to a compilation error as shown in output.

Check: Do you now understand

* + Use of **throw** keyword with unchecked exception
  + How to implement it

**Example 4.9E (M) Level: 3**

Purpose

Using keyword **throws** in method declaration during Exception Handling

Code

1. **class** ThrowsDemo {

*/\* using throws in method declaration so that callers of method can know that such an exception may occur\*/*

1. **static** **void** function() **throws** IllegalAccessException {
2. **throw** **new** IllegalAccessException("Throws Demo");
3. }
4. **public** **static** **void** main(String args[]) {
5. **try** {
6. function();
7. } **catch**(IllegalAccessException e) {
8. System.out.println("Exception in main: " + e);
9. }
10. }
11. }

Output

Exception in main: java.lang.IllegalAccessException: Throws Demo

Check your understanding

1. If a methodis not handling an exception that occurs within then ……………… is used in method’s declaration. (see line no. 2)
2. **throws** lists the type of exception that may occur in that method. (see line no. 2)
3. by using **throws** in a method we let the caller i.e. ……………………. to know that an exception may occur which is not handled by the function().

 Check: Do you now understand

* + When is **throws** used
  + What is its significance
  + How does it work
  + How to implement it

**Example 4.10E (C) Level: 2**

Purpose

Creating user defined Exception and giving user defined messages for that Exception class.

Code

*// extending class Exception or Throwable*

1. **class** UserDefinedException **extends** Exception {
2. UserDefinedException() {
3. }
4. UserDefinedException(String msg) { *// defining user defined*

*// methods*

1. super(msg);
2. }
3. }
4. **class** UserDefinedDemo {
5. **public** **static** **void** main(String args[]) {
6. **try** {
7. **throw** **new** UserDefinedException("My Exception");
8. } **catch**(UserDefinedException e) {
9. System.out.println("User Defined Exception: "+ e);
10. }
11. }
12. }

Output

User Defined Exception: UserDefinedException: My Exception

Check your understanding

1. To create user defined exception, we extend ……………………………. (see line no. 1)
2. To give user defined messages for the Exception class defined we have to define a ………………… which takes one parameter. (see line no. 4)
3. User defined exception is thrown along the parameter passed as shown in line no. 11.

 Check: Do you now understand

* + To create user defined exceptions and messages
  + How does it work
  + How to implement it

**Example 4.11E (C) Level: 4**

Purpose

Showing that a user defined exception class is of checked exception type.

Code

1. **class** UserDefinedException **extends** Exception {
2. UserDefinedException() { }
3. UserDefinedException(String msg) {
4. super(msg);
5. }
6. }
7. **class** UserDefinedDemo {
8. **static** **void** function() **throws** UserDefinedException { *// using throws*
9. **throw** **new** UserDefinedException("My Exception");
10. }
11. **public** **static** **void** main(String args[]) {
12. **try** {
13. function();
14. } **catch**(UserDefinedException e) {
15. System.out.println("User Defined Exception: "+ e);
16. }
17. }
18. }

Output

User Defined Exception: UserDefinedException: My Exception

Check your understanding

1. When a user defined exception is used then how to declare the function which is called. ( see line no. 9)
2. The user defined exception of ……………………….. type.

 Check: Do you now understand

* + When **throws** is required inuser defined exception
  + How does it work

**Example 4.12E (C) Level: 5**

Purpose

Demonstrating that a user defined exception can be created by extending RuntimeException class and is of Unchecked exception type.

Code

1. **class** UserDefinedException **extends** RuntimeException {
2. UserDefinedException() {
3. }
4. UserDefinedException(String msg) {
5. super(msg);
6. }
7. }
8. **class** UserDefinedDemo {
9. **static** **void** function() {
10. **throw** **new** UserDefinedException("My Exception"); }
11. **public** **static** **void** main(String args[]) {
12. **try** {
13. function();
14. } catch(UserDefinedException e) {
15. System.out.println("User Defined Exception: "+ e);
16. }
17. }
18. }

Output

User Defined Exception: UserDefinedException: My Exception

Check your understanding

1. To make the user defined exception of type unchecked exception, we extend…………………….. (see line no. 1)
2. Thus it does not require to use ……………….. clause in function declaration. (see line no. 9)

 Check: Do you now understand

* + How to create a user defined unchecked type exception
  + Why we do not require **throws** clause

**Quiz**

**1. Analyze the following code:**

1. class Test {
2. public static void main(String[] args) throws MyException {
3. System.out.println("Welcome to Java");
4. }
5. }
6. class MyException extends Error {
7. }
   1. You should not declare a class that extends Error, because Error raises a fatal error that terminates the program.
   2. You cannot declare an exception in the main method.
   3. You declared an exception in the main method, but you did not throw it.
   4. The program has a compilation error.

**2. Analyze the following code:**

1. class Test {
2. public static void main(String[] args) {
3. try {
4. String s = "5.6";
5. Integer.parseInt(s); // Cause a NumberFormatException
6. int i = 0;
7. int y = 2 / i;
8. }
9. catch (Exception ex) {
10. System.out.println("NumberFormatException");
11. }
12. catch (RuntimeException ex) {
13. System.out.println("RuntimeException");
14. }
15. }
16. }
    1. The program displays NumberFormatException.
    2. The program displays RuntimeException.
    3. The program displays NumberFormatException followed by RuntimeException.
    4. The program has a compilation error.

**3. Analyze the following code:**

1. class Test {
2. public static void main(String[] args) {
3. try {
4. String s = "5.6";
5. Integer.parseInt(s); // Cause a NumberFormatException
6. int i = 0;
7. int y = 2 / i;
8. System.out.println("Welcome to Java");
9. }
10. catch (Exception ex) {
11. System.out.println(ex);
12. }
13. }
14. }
    1. An exception is raised due to Integer.parseInt(s);
    2. An exception is raised due to 2 / i;
    3. The program has a compilation error.
    4. The program compiles and runs without exceptions.

**4.What is displayed on the console when running the following program?**

1. class Test {
2. public static void main(String[] args) {
3. try {
4. method();
5. System.out.println("After the method call");
6. }
7. catch (NumberFormatException ex) {
8. System.out.println("NumberFormatException");
9. }
10. catch (RuntimeException ex) {
11. System.out.println("RuntimeException");
12. }
13. }
14. static void method() {
15. String s = "5.6";
16. Integer.parseInt(s); // Cause a NumberFormatException
17. int i = 0;
18. int y = 2 / i;
19. System.out.println("Welcome to Java");
20. }
21. }
    1. The program displays NumberFormatException.
    2. The program displays NumberFormatException followed by After the method call.
    3. The program displays NumberFormatException followed by RuntimeException.
    4. The program has a compilation error.
    5. The program displays RuntimeException.

**5. What is displayed on the console when running the following program?**

1. class Test {
2. public static void main(String[] args) {
3. try {
4. method();
5. System.out.println("After the method call");
6. }
7. catch (RuntimeException ex) {
8. System.out.println("RuntimeException");
9. }
10. catch (Exception ex) {
11. System.out.println("Exception");
12. }
13. }
14. static void method() throws Exception {
15. t ry {
16. String s = "5.6";
17. Integer.parseInt(s); // Cause a NumberFormatException
18. int i = 0;
19. int y = 2 / i;
20. System.out.println("Welcome to Java");
21. }
22. catch (RuntimeException ex) {
23. System.out.println("RuntimeException");
24. }
25. catch (Exception ex) {
26. System.out.println("Exception");
27. }
28. }
29. }
    1. The program displays RuntimeException twice.
    2. The program displays Exception twice.
    3. The program displays RuntimeException followed by After the method call.
    4. The program displays Exception followed by RuntimeException.
    5. The program has a compilation error.

**6. What is displayed on the console when running the following program?**

1. class Test {
2. public static void main(String[] args) {
3. try {
4. method();
5. System.out.println("After the method call");
6. }
7. catch (RuntimeException ex) {
8. System.out.println("RuntimeException");
9. }
10. catch (Exception ex) {
11. System.out.println("Exception");
12. }
13. }
14. static void method() throws Exception {
15. try {
16. String s = "5.6";
17. Integer.parseInt(s); // Cause a NumberFormatException
18. int i = 0;
19. int y = 2 / i;
20. System.out.println("Welcome to Java");
21. }
22. catch (NumberFormatException ex) {
23. System.out.println("NumberFormatException");
24. throw ex;
25. }
26. catch (RuntimeException ex) {
27. System.out.println("RuntimeException");
28. }
29. }
30. }
    * 1. The program displays NumberFormatException twice.
      2. The program displays NumberFormatException followed by After the method call.
      3. The program displays NumberFormatException followed by RuntimeException.
      4. The program has a compilation error.

**7. What is wrong in the following program?**

* + - 1. class Test {
      2. public static void main (String[] args) {
      3. try {
      4. System.out.println("Welcome to Java");
      5. } } }

1. You cannot have a try block without a catch block.
2. You cannot have a try block without a catch block or a finally block.
3. A method call that does not declare exceptions cannot be placed inside a try block.
4. Nothing is wrong.

**8. What is displayed on the console when running the following program?**

* 1. class Test {
  2. public static void main (String[] args) {
  3. try {
  4. System.out.println("Welcome to Java");
  5. }
  6. finally {
  7. System.out.println("The finally clause is executed");
  8. }
  9. }
  10. }

1. Welcome to Java
2. Welcome to Java followed by The finally clause is executed in the next line
3. The finally clause is executed
4. None of the above

**9. What is displayed on the console when running the following program?**

1. class Test {
2. public static void main (String[] args) {
3. try {
4. System.out.println("Welcome to Java");
5. return;
6. }
7. finally {
8. System.out.println("The finally clause is executed");
9. } } }
   1. Welcome to Java
   2. Welcome to Java followed by The finally clause is executed in the next line
   3. The finally clause is executed
   4. None of the above

**10. What is displayed on the console when running the following program?**

* + 1. class Test {
    2. public static void main(String[] args) {
    3. try {
    4. System.out.println("Welcome to Java");
    5. int i = 0;
    6. int y = 2/i;
    7. System.out.println("Welcome to HTML");
    8. }
    9. finally {
    10. System.out.println("The finally clause is executed");
    11. }
    12. }
    13. }
        1. Welcome to Java.
        2. Welcome to Java followed by The finally clause is executed in the next line.
        3. The program displays three lines: Welcome to Java, Welcome to HTML, The finally clause is executed.
        4. None of the above.

**11. What is displayed on the console when running the following program?**

1. class Test {
2. public static void main(String[] args) {
3. try {
4. System.out.println("Welcome to Java");
5. int i = 0;
6. double y = 2.0 / i;
7. System.out.println("Welcome to HTML");
8. }
9. finally {
10. System.out.println("The finally clause is executed");
11. }
12. }
13. }
    1. Welcome to Java.
    2. Welcome to Java followed by The finally clause is executed in the next line.
    3. The program displays three lines: Welcome to Java, Welcome to HTML, The finally clause is executed.
    4. None of the above.

**12. What is displayed on the console when running the following program?**

* + 1. class Test {
    2. public static void main(String[] args) {
    3. try {
    4. System.out.println("Welcome to Java");
    5. int i = 0;
    6. int y = 2/i;
    7. System.out.println("Welcome to Java");
    8. }
    9. catch (RuntimeException ex) {
    10. System.out.println("Welcome to Java");
    11. }
    12. finally {
    13. System.out.println("End of the block");
    14. }
    15. }
    16. }

1. The program displays Welcome to Java three times followed by End of the block.
2. The program displays Welcome to Java two times followed by End of the block.
3. The program displays Welcome to Java three times.
4. The program displays Welcome to Java two times.

**13. What is displayed on the console when running the following program?**

* 1. class Test {
  2. public static void main(String[] args) {
  3. try {
  4. System.out.println("Welcome to Java");
  5. int i = 0;
  6. int y = 2/i;
  7. System.out.println("Welcome to Java");
  8. }
  9. catch (RuntimeException ex) {
  10. System.out.println("Welcome to Java");
  11. }
  12. finally {
  13. System.out.println("End of the block");
  14. }
  15. System.out.println("End of the block");
  16. }
  17. }

1. The program displays Welcome to Java three times followed by End of the block.
2. The program displays Welcome to Java two times followed by End of the block.
3. The program displays Welcome to Java two times followed by End of the block two times.
4. You cannot catch RuntimeException errors.

**14. What is displayed on the console when running the following program?**

* 1. class Test {
  2. public static void main(String[] args) {
  3. try {
  4. System.out.println("Welcome to Java");
  5. int i = 0;
  6. int y = 2/i;
  7. System.out.println("Welcome to Java");
  8. }
  9. finally {
  10. System.out.println("End of the block");
  11. }
  12. System.out.println("End of the block");
  13. }
  14. }

1. The program displays Welcome to Java three times followed by End of the block.
2. The program displays Welcome to Java two times followed by End of the block.
3. The program displays Welcome to Java two times followed by End of the block two times.
4. The program displays Welcome to Java and End of the block, and then terminates because of an unhandled exception.

**15. Which of the following is not an advantage of Java exception handling?**

A. Java separates exception handling from normal processing tasks.

B. Exception handling improves performance.

C. Exception handling makes it possible for the caller's caller to handle the exception.

D. Exception handling simplifies programming because the error-reporting and error-handling code can be placed at the catch block.

**16. Consider the code below:**

1. void myMethod()
2. { try
3. {
4. fragile();
5. }
6. catch( NullPointerException npex )
7. {
8. System.out.println( "NullPointerException thrown " );
9. }
10. catch( Exception ex )
11. {
12. System.out.println( "Exception thrown " );
13. }
14. finally
15. {
16. System.out.println( "Done with exceptions " );
17. }
18. System.out.println( "myMethod is done" );
19. }

**What is printed to standard output if fragile() throws an IllegalArgumentException?**

* 1. "NullPointerException thrown"
  2. "Exception thrown"
  3. "Done with exceptions"
  4. "myMethod is done"
  5. Nothing is printed

**17. Assuming a method contains code which may raise an Exception (but not a RuntimeException), what is the correct way for a method to indicate that it expects the caller to handle that exception:**

1. throw Exception
2. throws Exception
3. new Exception
4. Don't need to specify anything

**18. What is the result of executing the following code, using the parameters 4 and 0:**

1. public void divide(int a, int b) {
2. try {
3. int c = a / b;
4. } catch (Exception e) {
5. System.out.print("Exception ");
6. } finally {
7. System.out.println("Finally");
8. }
   1. Prints out: Exception Finally
   2. Prints out: Finally
   3. Prints out: Exception
   4. No output

**19. Given**

1. public class AQuestion
2. {
3. public static void main(String args[])
4. {
5. System.out.println("Before Try");
6. try
7. {
8. }
9. catch(Throwable t)
10. {
11. System.out.println("Inside Catch");
12. }
13. System.out.println("At the End");
14. } }
15. Compiler error complaining about the catch block, where no Throwable object can ever be thrown.
16. Compiler error - Throwable Object can not be caught, only Exceptions must be caught.
17. No compiler error. The lines "Before Try" and "At the end" are printed on the screen.

**20. Given**

* 1. public class AQuestion
  2. {
  3. public static void main(String args[])
  4. {
  5. System.out.println("Before Try");
  6. try
  7. {
  8. }
  9. catch(java.io.IOException t)
  10. {
  11. System.out.println("Inside Catch");
  12. }
  13. System.out.println("At the End");
  14. }
  15. }

1. Compiler error complaining about the catch block where no IOException object can ever be thrown.
2. Compiler error - IOException not found. It must be imported in the first line of the code.
3. No compiler error. The lines "Before Try" and "At the end" are printed on the screen.

**21. The class java.lang.Exception**

1. Is public
2. Extends Throwable
3. Implements Throwable
4. Is serializable

**22. Which of the following statements are true?**

* 1. A method can throw an Exception
  2. A method can return an Exception

**23.** **Given**

1. public class A
2. {
3. private void method1() throws Exception
4. {
5. throw new RuntimeException();
6. }
7. public void method2()
8. {
9. try
10. {
11. method1();
12. }
13. catch(RuntimeException e)
14. {
15. System.out.println("Caught Runtime Exception");
16. }
17. catch(Exception e)
18. {
19. System.out.println("Caught Exception");
20. }
21. }
22. public static void main(String args[])
23. {
24. A a = new A();
25. a.method2();
26. }
27. }

The above lines of code -

* 1. will not compile.
  2. will compile and show - "Caught Runtime Exception".
  3. will compile and show - "Caught Exception".
  4. will compile and show both the messages one after another in the order they appear.

**24. Read the following piece of code carefully.**

1. import java.io.IOException;
2. public class Question72
3. {
4. public Question72() throws IOException
5. {
6. throw new IOException();
7. }
8. }

Assume that the defination of Question72E begins with the line

public class Question72E extends Question72

It is required that none of the constructors of Question72E should throw any checked exception.

1. It can be achived by placing the call to the superclass with  a super keyword , which is placed in a try block with a catch block to handle the IOException thrown by the super class.
2. It can be achived by avoiding explicit calls to the base class constructor.
3. It cannot be done in the Java Laungage with the above definition of the base class.

**25. If a Runtime Exception is thrown in the finalize method -**

1. The running application crashes.
2. The exception is simply ignored and the object is garbage collected.
3. The exception is simply ignored, but the object is not garbage collected.
4. The Exception causes the JVM to crash.

**26. What all gets printed when the following gets compiled and run. Select the two correct answers.**

1. public class test {
2. public static void main(String args[]) {
3. int i=1, j=1;
4. try {
5. i++;
6. j--;
7. if(i == j)
8. i++;
9. }
10. catch(ArithmeticException e) {
11. System.out.println(0);
12. }
13. catch(ArrayIndexOutOfBoundsException e) {
14. System.out.println(1);
15. }
16. catch(Exception e) {
17. System.out.println(2);
18. }
19. finally {
20. System.out.println(3);
21. }
22. System.out.println(4);
23. }
24. }
    1. 0
    2. 1
    3. 2
    4. 3
    5. 4

**27. Which of these is a legal definition of a method named m assuming it throws IOException, and returns void. Also assume that the method does not take any arguments. Select the one correct answer.**

1. void m() throws IOException{ }
2. void m() throw IOException{ }
3. void m(void) throws IOException{ }
4. m() throws IOException{ }
5. void m() { } throws IOException

**28. Which of these statements are true. Select the two correct answers.**

1. For each try block there must be at least one catch block defined.
2. A try block may be followed by any number of finally blocks.
3. A try block must be followed by at least one finally or catch block.
4. If both catch and finally blocks are defined, catch block must precede the finally block.

**29. Consider these classes, defined in separate source files:**

* 1. public class Test1 {
  2. public float aMethod(float a, float b)
  3. throws IOException {
  4. }
  5. }
  6. public class Test2 extends Test1 {
  7. }

**Which of the following methods would be legal (individually) at line 2 in class Test2?**

1. float aMethod(float a, float b) { }
2. public int aMethod(int a, int b) throws Exception { }
3. public float aMethod(float a, float b) throws Exception { }
4. public float aMethod(float p, float q) { }

**30. Which of the following is correct?**

1. catch(Exception e)     will catch all exceptions.
2. catch(Error e)             will catch all errors.
3. catch(Throwable e)     will catch all exceptionns and errors

**31. EmptyStackException will be thrown when**

1. pop an empty Stack object.
2. peek an empty Stack object.
3. None of the above.

**32Bottom of Form**

1. **.ArithmeticException will occur after execution of the following code**
   1. int i = 10;
   2. float f = 0.0;
   3. double d = i/f;

1. NO.
2. Yes.

**33. A method may have several try blocks**

1. True.
2. False.

**34. Read the following program and find the answer.**

* 1. public class test1 {
  2. public test1( ) {
  3. try {
  4. \_\_\_\_\_\_
  5. }
  6. catch (Exception e) { ....... }
  7. catch (IOException eof) { ..... }
  8. }
  9. public static void main(String [] args) {
  10. new test1( );
  11. }
  12. }
      1. Above catch order is correct.
      2. Above catch order is incorrect.

**Answers**

1. A
2. D
3. A
4. A
5. C
6. C
7. A
8. B
9. B
10. B
11. B
12. B
13. C
14. D
15. B
16. B,C,D
17. A
18. A  
    Explanation: Both (A) and (B) would cause exception, but (A) occurred first, so the exception is due to (A).

19. C

20. A

21. A,B,D

22. A,B

**Rapid Fire**

1. Show that program statements that you want to monitor for exceptions are placed within a try block.
2. Show that a catch clause must immediately follow a try block.
3. Show that when you use multiple catch clauses the exception sub classes must come before any of their super classes.
4. Show that the try statements can be nested.
5. Write a program that throws an exception explicitly using throw keyword.
6. Write a program that demonstrates the use of throws keyword.
7. Show that the finally block will execute whether or not the exception has occurred.
8. Show that you can create your own Exception sub class by extending Throwable class.
9. Show that you can create your own Exception sub class by extending Exception class.
10. Show that you can create your own Exception sub class by extending RuntimeException class.

**References**

Java Complete Reference 🡪 Chapter 10 🡪 Page No: 249