# Exception Handling Quiz-1

**1.** Following code will result in: float num = 5/0;
A. Compilation error: Divisions must be in a try block
B. Compilation error: DivideByZeroException
C. Runtime Exception
D. No Error: a is NaN

**Answer:** C

**2**. What exception will be thrown from the following block of code?

```
try {
 throw new TryException();
}
catch {
 throw new CatchException();
}
finally {
 throw new FinallyException();
}
```

a) TryException
b) CatchException
c) FinallyException

**Answer: c**

**3**. We know that the method printStackTrace of an exception prints the stack of methods that have been call when that exception has ocurred. What stack trace will be printed after calling method1?

```
public void method1() throws Exception {
 method2();
}

public void method2() throws Exception {
 throw method3();
}

public Exception method3() {
 return new Exception();
}
```

a)
Exception in thread "main" java.lang.Exception
at mypackage.MyClass.method3(MyClass.java:30)
...
b)
Exception in thread "main" java.lang.Exception
at mypackage.MyClass.method2(MyClass.java:20)
...
c)
Exception in thread "main" java.lang.Exception
at mypackage.MyClass.method1(MyClass.java:10)
...

Answer: a

**4:** What will be the output of the program?

```
public class Foo
{
    public static void main(String[] args)
    {
        try
        {
            return;
        }
        finally
        {
            System.out.println( "Finally" );
        }
    }
}
```

**A.** Finally

**B.** Compilation fails.

**C.** The code runs with no output.

**D.** An exception is thrown at runtime.

**Answer: A**

**Explanation:** If you put a finally block after a try and its associated catch blocks, then once execution enters the try block, the code in that finally block will definitely be executed except in the following circumstances:

1. An exception arising in the finally block itself.
2. The death of the thread.
3. The use of System.exit()
4. Turning off the power to the CPU.

I suppose the last three could be classified as VM shutdown.

---

**5.** What will be the output of the program?

```
try
{
    int x = 0;
    int y = 5 / x;
}
catch (Exception e)
{
    System.out.println("Exception");
}
catch (ArithmeticException ae)
{
    System.out.println(" Arithmetic Exception");
}
System.out.println("finished");
```

**A.** finished

**B.** Exception

**C.** Compilation fails.

**D.** Arithmetic Exception

Option **C**

**Explanation:**

Compilation fails because ArithmeticException has already been caught.ArithmeticException is a subclass of java.lang.Exception, by time theArithmeticException has been specified it has already been caught by theException class.

If ArithmeticException appears before Exception, then the file will compile. When catching exceptions the more specific exceptions must be listed before the more general (the subclasses must be caught before the superclasses).

---

**6.** What will be the output of the program?

```
public class X
{
    public static void main(String [] args)
    {
        try
        {
            badMethod();
            System.out.print("A");
        }
        catch (Exception ex)
        {
            System.out.print("B");
        }
        finally
        {
            System.out.print("C");
        }
        System.out.print("D");
    }
    public static void badMethod()
    {
        throw new Error(); /* Line 22 */
    }
}
```

**A.** ABCD

**B.** Compilation fails.

**C.** C is printed before exiting with an error message.

**D.** BC is printed before exiting with an error message.

Option **C**

**Explanation:**

Error is thrown but not recognised line(22) because the only catch attempts to catch an Exception and Exception is not a superclass of Error. Therefore only the code in the finally statement can be run before exiting with a runtime error (Exception in thread "main" java.lang.Error).

**7.** What will be the output of the program?

```
public class X
{
    public static void main(String [] args)
    {
        try
        {
            badMethod();
            System.out.print("A");
        }
        catch (RuntimeException ex) /* Line 10 */
        {
            System.out.print("B");
        }
        catch (Exception ex1)
        {
            System.out.print("C");
        }
        finally
        {
            System.out.print("D");
        }
        System.out.print("E");
    }
    public static void badMethod()
    {
        throw new RuntimeException();
    }
}
```

**A.** BD

**B.** BCD

**C.** BDE

**D.** BCDE

**Answer:** Option **C**

**Explanation:**

A Run time exception is thrown and caught in the catch statement on line 10. All the code after the finally statement is run because the exception has been caught.

**8.** What will be the output of the program?

```
public class RTExcept
{
    public static void throwit ()
    {
        System.out.print("throwit ");
        throw new RuntimeException();
    }
    public static void main(String [] args)
    {
        try
        {
            System.out.print("hello ");
            throwit();
        }
        catch (Exception re )
        {
            System.out.print("caught ");
        }
        finally
        {
            System.out.print("finally ");
        }
        System.out.println("after ");
    }
}
```

**A.** hello throwit caught

**B.** Compilation fails

**C.** hello throwit RuntimeException caught after

**D.** hello throwit caught finally after

**Answer:** Option **D**

**Explanation:**

The main() method properly catches and handles the RuntimeException in the catch block, finally runs (as it always does), and then the code returns to normal.

A, B and C are incorrect based on the program logic described above. Remember that properly handled exceptions do not cause the program to stop executing.

**9.** What will be the output of the program?

```
public class Test
{
    public static void aMethod() throws Exception
    {
        try /* Line 5 */
        {
            throw new Exception(); /* Line 7 */
        }
        finally /* Line 9 */
        {
            System.out.print("finally "); /* Line 11 */
        }
    }
    public static void main(String args[])
    {
        try
        {
            aMethod();
        }
        catch (Exception e) /* Line 20 */
        {
            System.out.print("exception ");
        }
        System.out.print("finished"); /* Line 24 */
    }
}
```

**A.** Finally

**B.** exception finished

**C.** finally exception finished

**D.** Compilation fails

**Answer & Explanation**

**Answer:** Option **C**

**Explanation:**

This is what happens:

(1) The execution of the try block (line 5) completes abruptly because of the throwstatement (line 7).

(2) The exception cannot be assigned to the parameter of any catch clause of thetry statement

therefore the finally block is executed (line 9) and "finally" is output (line 11).

(3) The finally block completes normally, and then the try statement completes abruptly because of the throw statement (line 7).

(4) The exception is propagated up the call stack and is caught by the catch in the main method (line 20). This prints "exception".

(5) Lastly program execution continues, because the exception has been caught, and "finished" is output (line 24).

**10.** What will be the output of the program?

```
public class X
{
    public static void main(String [] args)
    {
        try
        {
            badMethod();
            System.out.print("A");
        }
        catch (Exception ex)
        {
            System.out.print("B");
        }
        finally
        {
            System.out.print("C");
        }
        System.out.print("D");
    }
    public static void badMethod() {}
}
```

**A.** AC

**B.** BC

**C.** ACD

**D.** ABCD

**Answer & Explanation**

**Answer:** Option **C**

**Explanation:**

There is no exception thrown, so all the code with the exception of the catch statement block is run.