Exception interview Question 1. What is exception in java?

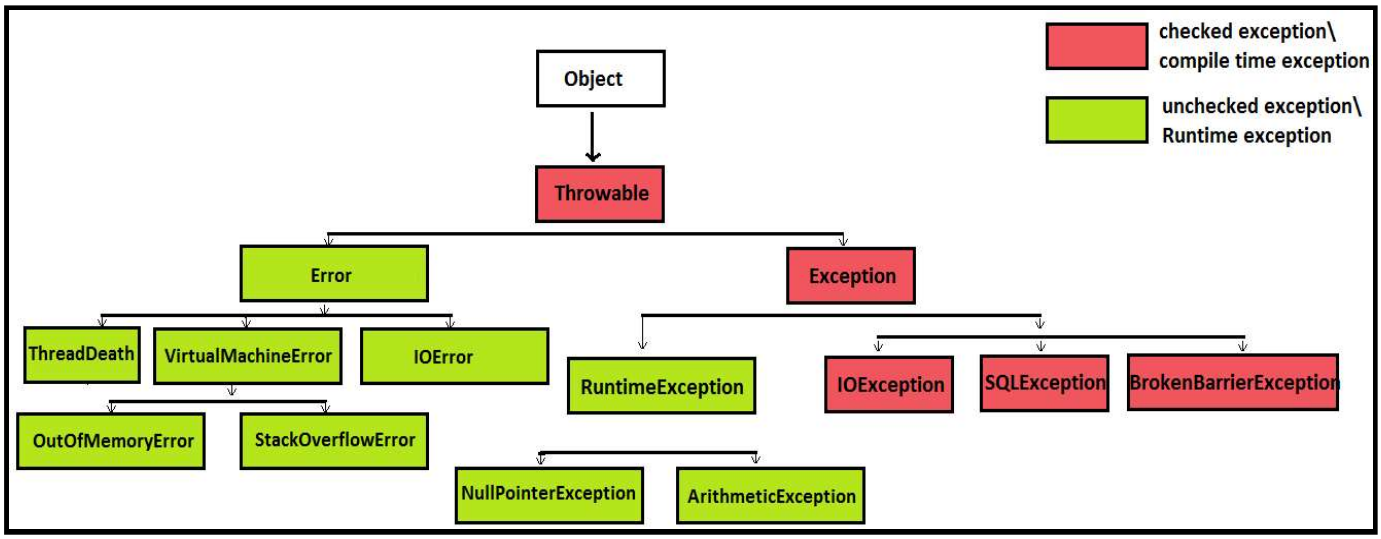
Answer. It's the basic Exception framework interview question. Freshers must know about this. Java Exception handling provides a mechanism to handle compile and runtime errors in java.

- To make application robust Exception must be handled appropriately,
- by handling exceptions we end up giving some meaningful message to end user rather than giving meaningless message in java.

Exception interview Question 2. Explain exception hierarchy in java?

Answer. It's also the basic Exception handling interview question. Freshers must know about this.

Exception hierarchy in java >



java.lang.**Object** is superclass of all classes in java.

java.lang.**Throwable** is superclass of java.lang.**Exception** and java.lang.**Error**

java.lang.**Exception** is superclass of java.lang.**RuntimeException, IOException, SQLException,** BrokenBarrierException and many more other classes in java.

java.lang.RuntimeException is superclass of java.lang.NullPointerException**, ArithmeticException** and many more other classes in java.

java.lang.Error is superclass of java.lang.**VirtualMachineError, IOError, AssertionError,** ThreadDeath and many more other classes in java.

java.lang.**VirtualMachineError** is superclass of java.lang.**OutOfMemoryError, StackOverflowError** and many more other classes in java.

Exception interview Question 3. What are differences between checked and unchecked exceptions in java?

Answer. This is very important Exception handling interview question in java.

	Property	***checked exception***	***unchecked exception***
1	Also known as	**checked** exceptions are also known as **compileTime** exceptions in java.	**unchecked** exceptions are also known as **runtime** exceptions in java.
2	Should be solved at compile or runtime?	Checked exceptions are those which need to be taken care at compile time in java.	Unchecked exceptions are those which need to be taken care at runtime in java.
3	Benefit/ Advantage	We cannot proceed until we fix compilation issues which are most likely to happen in program, this helps us in avoiding runtime problems upto lot of extent in java.	Whenever runtime exception occurs execution of program is interrupted, but by handling these kind of exception we avoid such interruptions and end up giving some meaningful message to user in java.

4	Creating custom/own exception	`class UserException extends Exception {`
		`  UserException(String s) {`
		`  super(s);`
		`  }`
		`}`
		By extending java.lang.**`Exception`**, we can create checked exception.
			`class UserException extends RuntimeException {`
			`  UserException(String s) {`
			`  super(s);`
			`  }`
			`}`
			By extending java.lang.**`RuntimeException`**, we can create unchecked exception.

5	Exception propagation	For **propagating checked** exceptions method must throw exception by using **throws** keyword.	**unchecked** exceptions are **automatically propagated** in java.
6	handling checked and unchecked exception while overriding superclass method	*If superclass method throws/declare **checked exception*** > • overridden method of subclass **can** declare/**throw narrower** (subclass of) **checked exception** (As shown in **Program**), or • overridden method of subclass **cannot** declare/**throw broader** (superclass of) **checked exception** (As shown in **Program**), or	*If superclass method throws/declare **unchecked*** > • overridden method of subclass **can** declare/**throw any unchecked /RuntimeException (superclass or subclass)** (As shown in **Program**), or • overridden method of subclass **cannot** declare/**throw any checked exception** (As shown in **Program**),

		• overridden method of subclass **can** declare/**throw any unchecked /RuntimeException** (As shown in **Program**)	
	Which classes are which type of exception? either **checked or unchecked** exception?	The class **Exception and all its subclasses** that are **not also subclasses of RuntimeException** are checked exceptions in java.	The class **RuntimeException and all its subclasses** are unchecked exceptions. Likewise, The class **Error and all its subclasses** are unchecked exceptions in java.
7	Most frequently faced exceptions	SQLException, IOException, ClassNotFoundException	NullPointerException, ArithmeticException ArrayIndexOutOfBoundsException.

Read more on : **Checked (compile time exceptions) and UnChecked (RuntimeExceptions) in java - Definition and differences**

Exception interview Question 4. What are 5 exception handling keywords in java?

Answer. This is another very important exception handling interview question in java.

5 keyword in java exception handling in java

- **try** **- Any exception occurring in try block is catched by catch block.**
- **catch** **-** catch block is always followed by try block in java.
- **finally** ***finally block*** *can can only exist if try or try-catch block is there, finally block can't be used alone in java.*

  Features *of finally >*
  - finally block is **always executed** irrespective of exception is thrown or not.
  - finally is **keyword** in java.
  - finally block is optional in java, we may use it or not.

  finally block is ***not executed*** *in following scenarios >*
  - finally is not executed when **System.exit** is called.
  - if in case **JVM crashes** because of some java.util.**Error**.
- **throw** **throw** is a **keyword** in java.
  - **throw** keyword allows us to throw **checked** or **unchecked** exception.

- **throws** **throws** is written in method's definition to indicate that method can throw exception in java.

Exception interview Question 5. Explain what is Error in java?

Answer. Experienced developers must know in detail about Exception handling interview question in java. ***java.lang.Error***

- Error is a subclass of **Throwable in java.**
- Error **indicates some serious problems** that our **application should not try to catch in java.**
- Errors are **abnormal conditions in application.**
- Error and its subclasses are regarded as **unchecked** exceptions in java

Must know :
ThreadDeath is an error which application must not try to catch but it is normal condition in java.

Exception interview Question 6. What are differences between Exception and Error in java?

Answer. It is another very important exception interview question to differentiate between Exception and Error in java.

	Property	Exception	Error
1	serious problem?	Exception does **not indicate any serious problem**.	Error **indicates some serious problems** that our **application should not try to catch.**
2	divided into **checked** and **unchecked**	Exception are divided into **checked** and **unchecked exceptions in java**.	Error are **not divided** further into such classifications in java.
3	Which classes are which type of exception? either **checked or unchecked** exception?	The class **Exception and all its subclasses** that are **not also subclasses of RuntimeException** are checked exceptions. The class **RuntimeException and all its subclasses** are unchecked exceptions. Likewise, The class **Error and all its subclasses** are unchecked exceptions in java.	Error and its subclasses are regarded as **unchecked** exceptions in java
4	Most frequently faced exception and errors	**checked exceptions>** SQLException, IOException, ClassNotFoundException	**VirtualMachineError, IOError, AssertionError,** ThreadDeath, **OutOfMemoryError, StackOverflowError.**

		unchecked exceptions> NullPointerException, ArithmeticException,	
5	Why to catch or not to catch?	Application **must catch** the Exception because they does not cause any major threat to application in java.	Application **must not catch** the Error because they does cause any major threat to application. Example > Let's say errors like OutOfMemoryError and StackOverflowError occur and are caught then JVM might not be able to free up memory for rest of application to execute, so it will be better if application don't catch these errors and is allowed to terminate in java.

Exception interview Question 7. Explain throw keyword in java?

Answer. This is also frequently asked exception handling interview question.

- **throw** is a **keyword** in java.
- **throw** keyword allows us to throw **checked** or **unchecked** exception.

throw unchecked exception in java >

- We need **not to handle** unChecked exception either by catching it or throwing it in java.

```
1  package throwUnChecked;
2  /** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
3  public class ExceptionTest {
4      public static void main(String[] args) {
5          m();
6      }
7      static void m(){
8          throw new NullPointerException();
9      }
10 }
11
12 /*OUTPUT
13
14 Exception in thread "main" java.lang.NullPointerException
15         at throwUnChecked.ExceptionTest.m(ExceptionTest.java:8)
16         at throwUnChecked.ExceptionTest.main(ExceptionTest.java:5)
17
18 */
```

We throw NullPointerException (unChecked exception) and didn't handled it, no compilation error was thrown in java.

throw checked exception >

- We need to handle checked exception either by catching it, or
- throwing it by using **throws** keyword. (When thrown, exception must be handled in calling environment)

Exception interview Question 8. Explain throws keyword in java?

Answer. This exception interview question will be covered in below question but it give you more detailed information about throw keyword in java.

throws is written in method's definition to indicate that method can throw exception.

throws unChecked exception in java >

- We need **not to handle** unChecked exception either by catching it or throwing it.

```
2  /** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
3  public class ExceptionTest {
4      public static void main(String[] args) {
5          m();
6      }
7      static void m() throws NullPointerException{
8
9      }
10 }
```

Above code **throws** NullPointerException (unChecked exception) and didn't handled it from where method m() was called and no compilation error was thrown.

throws Checked exception in java >

- We need **to handle** checked exception either by catching it or throwing it further, if not handled we will face compilation error.

```
5  /** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
6  public class ExceptionTest {
7      public static void main(String[] args) {
8          m();
9      }
10     static void m() throws FileNotFoundException{
11     }
12 }
```

Exception interview Question 9. What is difference between throw and throws in java?

Answer. This is also another important and frequently asked exception handling interview question. To confuse interviewees Interviewers might give you code snippet and ask you to insert throw or throws keyword in java.

	throw	throws
1	throw keyword is used to throw an exception explicitly in java.	throws keyword is used to declare an exception in java.

2	**throw** is used **inside method**. Example in java >
```
static void m(){
   throw new FileNotFoundException();
}
```
| **throws** is used **in method declaration**.

Example in java >
```
static void m() throws FileNotFoundException{
}
```
|
| 3 | **throw** is always **followed by instanceof** Exception class in java.

Example >
```
throw new FileNotFoundException()
```
| **throws** is always **followed by name of Exception class in java**.

Example >
```
throws FileNotFoundException
```
|
| 4 | **throw** can be used to throw **only one exception at time**.

Example >
```
throw new FileNotFoundException()
```
| **throws** can be used to throw **multiple exception at time**.

Example >
```
throws FileNotFoundException,
NullPointerException
```

and many more... |
| 5 | **throw** cannot propagate exception to calling method in java.

```
6  public class ExceptionTest {
7      public static void main(String[] args) {
8          m();
9          System.out.println("after calling m()");
10     }
11     static void m(){
12         throw new FileNotFoundException();
13     }
14 }
```
| **throws** can propagate exception to calling method.

Please see these programs to understand how exception is propagated to calling method.
Program 1 - Handling Exception by throwing it from m() method (using throws keyword) and handling it in try-catch block from where call to method m() was made.

Program 2 - Throwing Exception from m() method and then again throwing it from calling method [i.e. main method] |

Exception interview Question 10. How to create user defined checked and unchecked Exception in java?

Answer. Very important exception handling interview question. Interviewers generally expects interviewees to write code to create checked and unchecked Exception in java.

*Creating user defined **checked** exception in java >*

```
class UserDefinedException extends Exception {

    UserDefinedException(String s) {
           super(s);
    }

}
```

By extending java.lang.**Exception**, we can create checked exception.

*Creating user defined **unchecked** exception in java >*

```
class UserDefinedException extends RuntimeException {

    UserDefinedException(String s) {
           super(s);
    }

}
```

By extending java.lang.**RuntimeException**, we can create unchecked exception.

Exception interview Question 11. How to use try-catch-finally in java? Can we use try,catch or finally block alone in java?

Answer. This exception handling interview question will test your practical/basic understanding of Exception handling in java.

We can enclose exception prone code in >

- **try-catch** block, or
- **try-finally** block, or
- **try-catch-finally** block.

Using try-catch block in java

```
       try{
              //Code to be enclosed in try-catch block
       }catch(Exception e){
       }
```

Using try-finally block in java

```
       try{
              //Code to be enclosed in try-finally block
       }finally{
       }
```

We cannot use **try block** alone, it must be followed by either **catch** or **finally**.

Using only try block will cause *compilation error*

```
       try{
              //only try block will cause compilation error
       }
```

*Likewise, we cannot use **catch block** alone, it always follows **try block.***

Using only catch block will cause *compilation error*

```
       catch{
              //only catch block will cause compilation error
       }
```

*Likewise, we cannot use **finally block** alone, it always follows **try block.***

Using only finally block will cause *compilation error*

```
        finally{
                //only finally block will cause compilation error
        }
```

Exception interview Question 12. Is it allowed to use multiple catch block in java?

Answer. Another exception handling interview question which will test your practical knowledge and understanding of Exception handling in java. Java exception handling allows us to use multiple catch block in java.

Important Point about multiple catch block in java >

1. **Exception class handled in starting catch block must be subclass of Exception class handled in following catch blocks (otherwise we will face compilation error).**
2. Either one of the multiple catch block will handle exception at time in java.

Program - Let's understand the concept of multiple catch block in java>

```
/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ExceptionTest {
    public static void main(String[] args) {

            try{
                    int i=10/0; //will throw ArithmeticException
            }catch(ArithmeticException ae){
                    System.out.println("Exception handled - ArithmeticException");
            }catch(RuntimeException re){
                    System.out.println("Exception handled - RuntimeException");
            }catch(Exception e){
                    System.out.println("Exception handled - Exception");
            }

    }
}

/*OUTPUT

Exception handled - ArithmeticException

*/
```

In the above above >
`ArithmeticException` has been used in **first** catch block
`RuntimeException` has been used in **second** catch block
`Exception` has been used in **third** catch block

`Exception` is superclass of **`RuntimeException`** and **`RuntimeException`** is superclass of **`ArithmeticException.`**

Exception interview Question 13. What is Automatic resource management in java 7?

Answer. Experienced java developers must be well versed with this exception interview question. As we know java allows us to handle multiple exceptions by using multiple catch blocks.

Now, java 7 has done improvements in multiple exception handling by introducing **multi catch syntax** which helps in **automatic resource management.**

*Features of **multi catch syntax in java** >*

- Has **improved way of catching multiple exceptions.**
- This syntax does **not looks clumsy in java**.
- **Reduces developer efforts** of writing multiple catch blocks in java.
- Allows us to **catch more than one exception in one catch block**.

Here is the **multi catch syntax** >

```
            try{
                //code . . . . .
        }catch(IOException | SQLException ex){
                //code . . . . .
        }
```

We could separate different exceptions using **pipe** (**|**) in java.

Exception interview Question 14. Explain try-with-resource in java?

Answer. Again experienced java developers must be well versed with this exception interview question.

Before java 7, we used to write **explicit code for closing file in finally block by using try-finally block** like this >

```
/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class TryWithResourseTest {
    public static void main(String[] args) throws IOException {
            InputStream inputStream = null;
            try{
                    inputStream = new FileInputStream("c:/txtFile.txt");
                    //code......
            }finally{
                    if(inputStream!=null)
                    inputStream.close();
            }
    }
}
```

In java 7, using Try-with-resources >

- we need not to write **explicit code for closing file**.

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class TryWithResourseTest {
```

```
   public static void main(String[] args) throws IOException {
         try (InputStream inputStream = new
FileInputStream("c:/txtFile.txt")) {
               //code...
         }
   }
}
```

*Using multiple resources inside **Try-with-resources is also allowed in java.***

Exception interview Question 15. Now, question comes why we need not to close file when we are using Try-with-resources in java?

Answer. Again experienced java developers must be well versed with this exception interview question.

Because `FileInputStream` implements java.lang.**AutoCloseable interface** (**AutoCloseable** interface's close method automatically closes resources which are no longer needed) in java.

Which classes can be used inside **Try-with-resources in java?**

All the classes which implements **AutoCloseable** interface can be used inside **Try-with-resources in java.**

Exception interview Question 16. Explain finally keyword in java?

Answer. Fresher and experienced java developers must be well versed with this exception handling interview question in java.

try or try-catch block can be followed by finally block in java >

- try-**finally** block, or

```
try{
            //Code to be enclosed in try-finally block
      }finally{
      }
```

- try-catch-**finally** block.

```
      try{
            //Code to be enclosed in try-catch-finally block
      }catch(Exception e){
      }finally{
      }
```

> ***finally block** can can only exist if try or try-catch block is there, finally block can't be used alone in java.*

***Features** of finally in java >*

- finally block is **always executed** irrespective of exception is thrown or not.

- finally is **keyword** in java.

*finally block is **not executed** in following scenarios in java >*

- finally is not executed when **System.exit** is called.
- if in case **JVM crashes** because of some java.util.**Error**.

Application of finally block in java programs in java >

- We may use finally block to execute code for **database connection closing**, because closing connection in try or catch block may not be safe.
    - **Why closing connection in try block may not be safe?**
    - Because exception may be thrown in try block before reaching connection closing statement.

    - **Why closing connection in catch block may not be safe?**
    - Because inappropriate exception may be thrown in try block and we might not enter catch block to close connection in java.

For programs to demonstrate finally. Please refer this post.

Exception interview Question 17. Is it allowed to use nested try-catch in java?

Answer. It's basic java exception handling interview question.
java exception handling allows us to use nested try-catch block.

Nested try-catch block means using try-catch block inside another try-catch block in java.

```
/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ExceptionTest {
    public static void main(String[] args) {

            try{
                    int i=10/0; //will throw ArithmeticException
            }catch(ArithmeticException ae){
                    System.out.println("try-catch block handled - ArithmeticException");

                    //using nested try-catch block
                    try{
                            String s=null;
                            s.charAt(0); //will throw NullPointerException
                    }catch(NullPointerException npe){
                            System.out.println("NESTED try-catch block handled - "
                                                            + "NullPointerException");
                    }

            }

    }
}
```

Exception interview Question 18. Discuss which checked and unchecked exception can be thrown/declared by subclass method while overriding superclass method in java?

Answer. It's very very important exception handling interview question. Experienced and freshers all must be able to answer this question.

*If superclass method throws/declare **unchecked/RuntimeException in java** >*

- overridden method of subclass **can** declare/**throw any unchecked /RuntimeException (superclass or subclass)**, or
- overridden method of subclass **cannot** declare/**throw any checked exception in java**, or
- overridden method of subclass **can** declare/**throw same exception in java**, or
- overridden method of subclass **may not** declare/**throw any exception in java**.

*If superclass method throws/declare **checked/compileTime exception in java** >*

- overridden method of subclass **can** declare/**throw narrower** (subclass of) **checked exception**, or
- overridden method of subclass **cannot** declare/**throw broader** (superclass of) **checked exception**, or
- overridden method of subclass **can** declare/**throw any unchecked /RuntimeException**, or
- overridden method of subclass **can** declare/**throw same exception**, or
- overridden method of subclass **may not** declare/**throw any exception in java**.

*If superclass method does **not throw**/declare any **exception in java** >*

- overridden method of subclass **can** declare/**throw any unchecked /RuntimeException** , or
- overridden method of subclass **cannot** declare/**throw any checked exception**, or
- overridden method of subclass **may not** declare/**throw any exception in java**.

For programs please refer > Throw/declare checked and unchecked exception while overriding superclass method in java

Exception interview Question 19. What will happen when catch and finally block both return value, also when try and finally both return value in java?

Answer. This is very important exception handling interview question for experienced developers.

When **catch and finally block** both return value, **method will ultimately return value returned by** finally block irrespective of value returned by catch block.

```
/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ExceptionTest {
    public static void main(String[] args) {
           System.out.println("method return -> "+m());
    }
```

```
    static String m(){
            try{
                    int i=10/0; //will throw ArithmeticException
            }catch(ArithmeticException e){
                    return "catch";
            }finally{
                    return "finally";
            }

    }
}

/*OUTPUT

method return -> finally

*/
```

In above program, i=10/0 will throw `ArithmeticException` and enter catch block to return `"catch"`, but ultimately control will enter finally block to return `"finally"`.

Likewise, when **try and finally block** both return value, **method will ultimately return value returned by finally block** irrespective of value returned by try block. For program please refer.

Exception interview Question 20. What is exception propagation in java?

Answer.

Experienced developers must know in detail about Exception handling interview question in java. Even freshers must try and understand this in depth concept of exception propagation in java.

Whenever methods are called stack is formed and an exception is first thrown from the top of the stack and if it is not caught, it starts coming down the stack to previous methods until it is not caught.

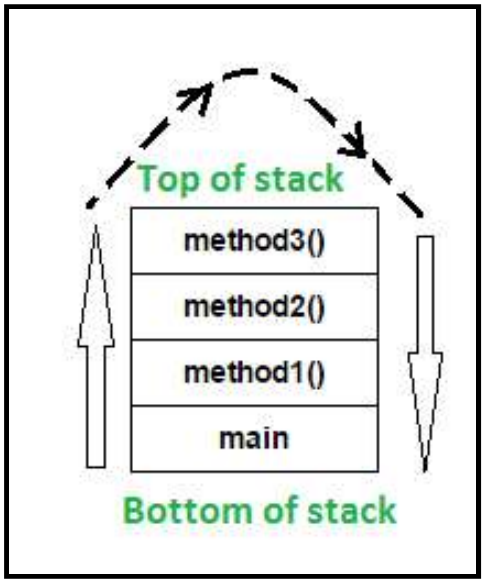
If exception remains uncaught even after reaching bottom of the stack it is propagated to JVM and program is terminated in java.

*Propagating **unchecked** exception (NullPointerException) >*

unchecked exceptions are **automatically propagated** in java.

```java
package propogation;

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ExceptionTest {
    public static void main(String[] args)
    {
        method1();
        System.out.println("after calling m()");
    }

    static void method1()
    {
        method2();
    }

    static void method2()
    {
        method3();
    }

    static void method3()
    {
        throw new NullPointerException();
    }

}

/*OUTPUT

Exception in thread "main" java.lang.NullPointerException
        at propogation.ExceptionTest.method3(ExceptionTest.java:21)
        at propogation.ExceptionTest.method2(ExceptionTest.java:17)
        at propogation.ExceptionTest.method1(ExceptionTest.java:13)
        at propogation.ExceptionTest.main(ExceptionTest.java:8)

*/
```


stack *of methods is formed >*

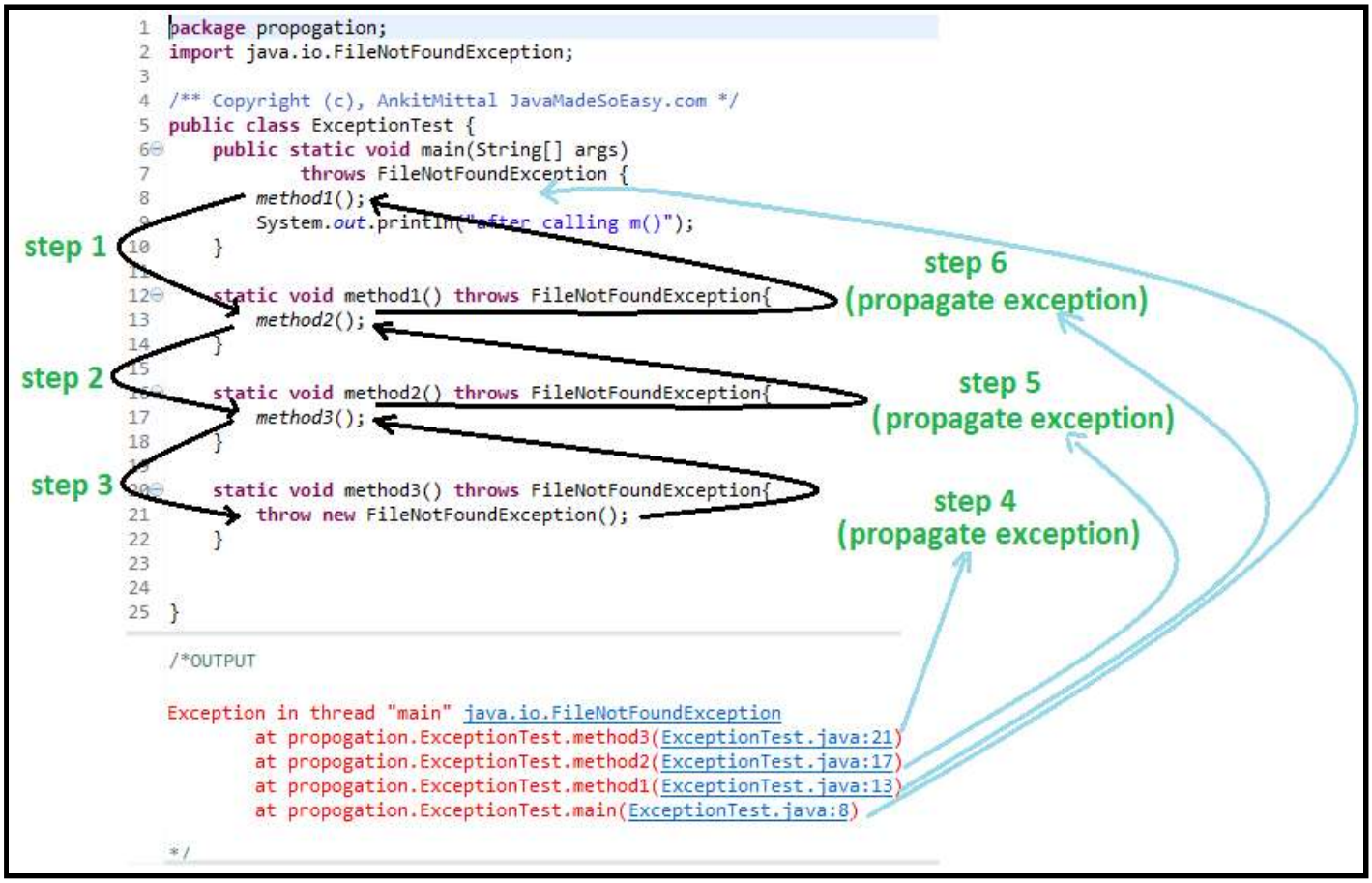


In the above program, stack is formed and an exception is first thrown from the top of the stack [**method3()**] and it remains uncaught there, and starts coming down the stack to previous methods to **method2()**, then to **method1()**, than to **main()** and it remains uncaught throughout.

exception remains uncaught even after reaching bottom of the stack [**main()**] so it is propagated to JVM and ultimately program is terminated by throwing exception [as shown in output] in java.

*Propagating **checked** exception (FileNotFoundException) using throws keyword >*

For **propagating checked** exceptions method must throw exception by using **throws** keyword.




```
package propogation;
import java.io.FileNotFoundException;

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ExceptionTest {
    public static void main(String[] args)
            throws FileNotFoundException {
        method1();
        System.out.println("after calling m()");
    }

    static void method1() throws FileNotFoundException{
        method2();
    }

    static void method2() throws FileNotFoundException{
        method3();
    }

    static void method3() throws FileNotFoundException{
        throw new FileNotFoundException();
    }


}

/*OUTPUT

Exception in thread "main" java.io.FileNotFoundException
        at propogation.ExceptionTest.method3(ExceptionTest.java:21)
        at propogation.ExceptionTest.method2(ExceptionTest.java:17)
        at propogation.ExceptionTest.method1(ExceptionTest.java:13)
        at propogation.ExceptionTest.main(ExceptionTest.java:8)

*/
```

Exception interview Question 21. Can a catch or finally block throw exception in java?

Answer. Yes, catch or finally block can throw checked or unchecked exception but it must be handled accordingly. Please refer this post for handling checked and unchecked exceptions in java.

Exception interview Question 22. Why shouldn't you use Exception for catching all exceptions in java?

Answer. Catching Exception rather than handling specific exception can be vulnerable to our application. Multiple catch blocks must be used to catch specific exceptions, because handling specific exception gives developer the liberty of taking appropriate action and develop robust application.

Exception interview Question 23. What is Difference between multiple catch block and multi catch syntax?

Answer. Experienced developers must know in detail about this Exception handling interview question in java

	multiple catch block	multi catch syntax
1	multiple catch blocks were introduced in prior versions of Java 7 and does not provide any automatic resource management in java.	**multi catch syntax was introduced in** java 7 for improvements in multiple exception handling which helps in **automatic resource management in java.**

2	Here is the syntax for writing **multiple catch block in java** >
	`try{`
	`  //code . . . . .`
	`}catch(IOException ex1){`
	`  //code . . . . .`
	`} catch(SQLException ex2){`
	`  //code . . . . .`
	`}`
		Here is the **multi catch syntax in java** >
		`try{`
		`  //code . . . . .`
		`}catch(IOException | SQLException ex){`
		`  //code . . . . .`
		`}`
		We could separate different exceptions using **pipe** (**|**)

3	For catching IOException and SQLException we need to write **two catch block** like this >
	`import java.io.IOException;`
	`import java.sql.SQLException;`
	
	`/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */`
	`public class ExceptionTest {`
	`  public static void main(String[] args) {`
	
	`  try{`
	`  int i=1;`
	`  if(i==1)`
	`  throw new IOException();`
	`  else`
	`  throw new SQLException();`
	`  }catch(SQLException ex){`
	`  System.out.println(ex + " handled ");`
	`  }catch(IOException ex){`
	`  System.out.println(ex + " handled ");`
	`  }`
	
	`  }`
	`}`
	`/*OUTPUT`
	
	`java.io.IOException handled`
	
	`*/`
		with the help of multi catch syntax we can catch IOException and SQLException in one catch block using **multi catch syntax** like this >
		`import java.io.IOException;`
		`import java.sql.SQLException;`
		
		`/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */`
		`public class ExceptionTest {`
		`  public static void main(String[] args) {`
		
		`  try{`
		`  int i=1;`
		`  if(i==1)`
		`  throw new IOException();`
		`  else`
		`  throw new SQLException();`
		`  }catch(IOException | SQLException ex){`
		`  System.out.println(ex + " handled ");`
		`  }`
		
		`  }`
		`}`
		
		`/*OUTPUT`
		
		`java.io.IOException handled`
		
		`*/`

4	**When multiple catch blocks** are used , first catch block could be subclass of Exception class handled in following catch blocks like this >	If **Multi catch syntax** is used to catch subclass and its superclass than compilation error will be thrown. IOException and Exception in **multi catch syntax** will cause compilation error “The exception **IOException** is already caught by the alternative **Exception**”.

IOException is subclass of Exception in java.

```
import java.io.IOException;

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ExceptionTest {
    public static void main(String[] args) {
        try{
            throw new IOException();
        }catch(IOException e){
            System.out.println("IOException handled");
        }catch(Exception e){
            System.out.println("Exception handled ");
        }
    }
}

/*OUTPUT

IOException handled

*/
```

```
import java.io.IOException;

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ExceptionTest {
    public static void main(String[] args) {

        try{
            throw new IOException();
        }catch(IOException | Exception ae){
            System.out.println("Exception handled");
        }

    }
}
```

Solution >

We must use only **Exception** to catch its subclass like this >

```
public class ExceptionTest {
    public static void main(String[] args) {

        try{
            throw new IOException();
        }catch(Exception ae){
            System.out.println("Exception handled");
        }

    }
}
```

5	Does not provide such features.	*Features of **multi catch syntax** >* • Has **improved way of catching multiple exceptions**. • This syntax does **not looks clumsy**. • **Reduces developer efforts** of writing multiple catch blocks. • Allows us to **catch more than one exception in one catch block**. • Helps in **automatic resource management.**

For more read : *Difference between multiple catch block and multi catch syntax*

Exception interview Question 24. can a method be overloaded on basis of exceptions in java ?

Answer.

Another Exception handling interview question which will test your practical understanding of exception in java.

Yes a method be overloaded on basis of exceptions in java.

But now question which overloaded exception will be called.

Let's take an example :
***Ques**. Let's say one method handles Exception and other handles ArithmeticException. Which method will be invoked when ArithmeticException is thrown?*
Ans. Method which handles more specific exception will be called.

Program >

```
import java.io.IOException;

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com
 * Main class */
public class ExceptionTest {

    void method(Exception e){
           System.out.println(e+" caught in Exception method");
    }
    void method(ArithmeticException ae){
           System.out.println(ae+" caught in ArithmeticException method");
    }

    public static void main(String[] args) {
           ExceptionTest obj=new ExceptionTest();
           obj.method(new ArithmeticException());
           obj.method(new IOException());
    }
}

/* OUTPUT

java.lang.ArithmeticException caught in ArithmeticException method
java.io.IOException caught in Exception method

*/
```

Exception interview Question 25. Mention few exception handling best practices in java?

Answer. Experienced developers must be able to answer this Exception handling interview question in detail in java.

- Throw exceptions **when the method cannot handle the exception**, and more **importantly**, must be **handled by the caller.**

  Example -

  In Servlets - doGet() and doPost() throw ServletException or IOException in certain circumstances where the request could not be read correctly. Neither of these methods are in a position to handle the exception, but the container is (which generally results in the 404 error page in most cases).

- **Bubble the exception if the method cannot handle it.** This is a corollary of the above point, but applicable to methods that must catch the exception. If the caught exception cannot be handled correctly by the method, then it is preferable to bubble it.

- **Throw the exception right away**. If an exception scenario is encountered, then it is a good practice to throw an exception indicating the original point of failure, instead of attempting to handle the failure via error codes, until a point deemed suitable for throwing the exception. In other words, attempt to minimize mixing exception handling with error handling.

- **Either log the exception or bubble it, but don't do both.** Logging an exception often indicates that the exception stack has been completely unwound, indicating that no further bubbling of the exception has occurred. Hence, it is not recommended to do both at the same time, as it often leads to a frustrating experience in debugging.

- **Application should not try to catch Error -** Because, in most of cases recovery from an Error is almost impossible. So, application must be allowed to terminate.

  Example>
  Let's say errors like OutOfMemoryError and StackOverflowError occur and are caught then JVM might not be able to free up memory for rest of application to execute, so it will be better if application don't catch these errors and is allowed to terminate.

Read : Complete list of Exception handling best practices and guidelines for using exceptions in java

Exception interview Question 26. Difference between Final, Finally and Finalize in java?

Answer. It is another very very important exception interview question to differentiate between final, finally and finalize in java.

	final	***finally***	***finalize***
1	final can be applied to **variable**, **method** and **class** in java.	finally is a block.	finalize is a method.
2	***2.1) Final variable***	*try or try-catch block can be followed by finally block >*	finalize method is called before garbage collection by JVM,

final memberVariable
final local variable
final static variable

Final memberVariable of class must be initialized at time of declaration, once initialized final memberVariable cannot be assigned a new value.
Final variables are called **constants** in java.

```
class FinalTest {
    final int x=1;
//memberVariable/instanceVariable
}
```

If constructor is defined then final memberVariable can be initialized in constructor but once initialized cannot be assigned a new value.

```
class FinalTest {
    final int x;
//memberVariable/instanceVariable
    FinalTest() {
          x = 1; //final
memberVariable can be initialized in
constructor.
    }
}
```

Final local variable can be left uninitialized at time of declaration and can be initialized later, but once initialized cannot be assigned a new value in java.

```
class FinalTest {
   void method(){
      final int x; //uninitialized at
time of declaration
       x=1;
   }
}
```

Final static variable of class must be initialized at time of declaration or can be initialized in static block, once initialized final static variable cannot be assigned a new value.

If static block is defined then final static variable can be initialized in static block, once initialized final

try-**finally** block, or

```
try{
//Code to be enclosed in
try-finally block
}finally{
}
```

try-catch-**finally** block.

```
try{
//Code to be enclosed
in try-catch-finally
block
}catch(Exception e){
}finally{
}
```

***finally block** can can only exist if try or try-catch block is there, finally block can't be used alone in java.*

finally block is not executed in following scenarios >
finally is not executed when **System.exit** is called.
if in case **JVM crashes** because of some java.util.[Error](#).

finalize method is called for any cleanup action that may be required before garbage collection.

```
/** Copyright (c),
AnkitMittal
JavaMadeSoEasy.com */


@Override
protected void finalize() throws
Throwable {
 try {

     System.out.println("in
  finalize() method, "
                                  +

  "doing cleanup activity");

 } catch (Throwable throwable) {
      throw throwable;
 }
}
```

finalize() method is defined **in java.lang.Object**

static variable cannot be assigned a new value.

```
class FinalTest {
  final static int x; //static
variable
  static{ //static block
          x=1;
  }
}
```

2.2) Final method

Final method cannot be overridden, any attempt to do so will cause compilation error.

```
5  class Superclass {
6      final void method(){} //final method
7  }
8
9  class Subclass extends Superclass{
10     void method(){} //Final method cannot be overridden
11 }
```

Runtime polymorphism is not applicable on final methods because they cannot be inherited.

2.3) Final class

Final class cannot be extended, any attempt to do so will cause compilation error.

```
5  final class Superclass {  //final method
6  }
7
8  class Subclass extends Superclass{ //final class cannot be extended
9  }
```

3	-	finally block can only exist if try or try-catch block is there, finally block can't be used alone in java.	We can *force early garbage collection in java* by using following methods >
			`System.gc();`
			`Runtime.getRuntime().gc();`
			`System.runFinalization();`
			`Runtime.getRuntime().runFinalization();`

4	-	finally is always executed irrespective of exception thrown in java.	If any uncaught exception is thrown inside finalize method - **exception is ignored, thread is terminated and object is discarded.** **Note :** Any exception thrown by the finalize method causes the finalization of this object to be halted, but is otherwise ignored.

5	-	Currently executing thread calls finally method in java.	JVM does not guarantee which ***daemon*** thread will invoke the finalize method for an object.
6	final is a keyword in java.	finally Is a keyword in java.	finalize is not a keyword in java.

For more detail read : **Final, Finally and Finalize - difference and similarity in java**

Another Exception interview Question. **What are the differences between** between ClassNotFoundException and NoClassDefFoundError in java ?

Answer.

	ClassNotFoundException	***NoClassDefFoundError***
1	ClassNotFoundException is Checked (compile time) **Exception** in java.	NoClassDefFoundError is a Error in java. Error and its subclasses are regarded as **unchecked** exceptions in java.
2	Here is the **hierarchy** of java.lang.ClassNotFoundException - -java.lang.Object -java.lang.Throwable -java.lang.Exception - java.lang.ReflectiveOperationException -java.lang.ClassNotFoundException	Here is the **hierarchy** of java.lang.NoClassDefFoundError - -java.lang.Object -java.lang.Throwable -java.lang.Error -java.lang.LinkageError -java.lang.NoClassDefFoundError
3	**ClassNotFoundException** is thrown when JVM tries to class from classpath but it does not find that class.	**NoClassDefFoundError** is thrown when JVM tries to load class which > • **was NOT** available at **runtime** but • **was** available at **compile** time.
	`ExceptionInInitializerError` has got nothing to do with **ClassNotFoundException**.	You must ensure that class does not throws `java.lang.ExceptionInInitializerError` because that is likely to be followed by **NoClassDefFoundError**.

For more read differences between between ClassNotFoundException and NoClassDefFoundError in java

Another very important Exception interview **Question**. **What are the most important frequently occurring Exception and Errors which you faced in java?**

Answer. Most common and frequently occurring **checked (compile time)** and Errors in java >

- **FileNotFoundException** in java
- **SQLException** in java
- What is java.lang.**InterruptedException** in java
- when java.lang.**ClassNotFoundException** occurs in java

Most common and frequently occurring unchecked **(runtime)** in java.

- What is java.lang.**NullPointerException** in java, when it occurs,how to handle, avoid and fix it
- **NumberFormatException** in java
- **IndexOutOfBoundsException** in java
- When java.lang.**ArrayIndexOutOfBoundsException** occurs in java
- When java.lang.**StringIndexOutOfBoundsException** occurs in java
- **java.lang.ArithmeticException** in java - Divide number by zero
- When dividing by zero does not throw **ArithmeticException** in java
- When java.lang.**IllegalStateException** occurs in java
- when java.lang.**IllegalMonitorStateException** is thrown in java
- Solve java.lang.**UnsupportedOperationException** in java

Most common and frequently occurring **Errors** in java >

- **OutOfMemoryError** in java
- When java.lang.**StackOverflowError** occurs in java
- Solve java.lang.**ExceptionInInitializerError** in java
- How to solve java.lang.**NoClassDefFoundError** in java