Questions about Throwable

Table of Contents


```
<style>
  .imageblock > .title {
    text-align: inherit;
  }
</style>
```

General Information About Exceptions

1. What is an exception in programming in general?
2. What is an exception throwing?
3. What is an exception propagating?

Throwable

1. What is `Throwable`?
2. What is `Throwable` hierarchy?
3. What is the difference between checked and unchecked exceptions?
4. Name 5 checked `Exception`s and explain when they can occur?
5. Name 5 checked unchecked `Exception`s and explain when they can occur?
6. Name 2 `Error`s and explain when they can occur?
7. Name at least 4 constructors of `Throwable`?
8. Name and explain at least 6 methods implemented in `Throwable`?
9. Can `Error` be explicitly thrown or caught?

10. Why only `Throwable` and its subclasses can be thrown?
11. Is it possible to change the message of a caught `Throwable` object without using reflection? If yes, how would you do that?
12. Is it possible to change the stack trace of a caught `Throwable` object without using reflection? If yes, how would you do that?
13. Is it possible to change line numbers for the stack trace of a caught `Throwable` object without using reflection? If yes, how would you do that?

Key Points

Keywords

Name 5 keywords related to exceptions and explain each of them?

`finally`

For every code snippet below answer the following questions:

1. Will the `finally` block be executed?
2. Will the exception from the `try` block be:
    - caught?
    - propagated to the caller?
3. Will `finally` block throw an exception? It yes: will it be propagated to the caller?

Snippet A

```
try {
    Printer.printToConsole("Entered the 'try' block");
    throw new RuntimeException("Exception from 'try'");
} catch (RuntimeException exception) {
    Logger.error(exception);
} finally {
    Printer.printToConsole("Entered the 'finally' block");
}
```

Snippet B

```
try {
    Printer.printToConsole("Entered the 'try' block");
    throw new IndexOutOfBoundsException("Exception from 'try'");
} catch (ArithmeticException exception) {
    Logger.error(exception);
} finally {
    Printer.printToConsole("Entered the 'finally' block");
}
```

Snippet C

```
try {
    Printer.printToConsole("Entered the 'try' block");
    throw new IndexOutOfBoundsException("Exception from 'try' block");
} catch (IndexOutOfBoundsException exception) {
    Logger.error(exception);
} finally {
    Printer.printToConsole("Entered the 'finally' block");
    throw new ArithmeticException("Exception from the 'finally' block");
}
```

Snippet D

```
try {
    Printer.printToConsole("Entered the 'try' block");
    throw new IllegalStateException("Exception from the 'try' block");
} catch (IndexOutOfBoundsException exception) {
    Logger.error(exception);
} finally {
    Printer.printToConsole("Entered the 'finally' block");
    throw new ArithmeticException("Exception from the 'finally' block");
}
```

Multicatching

1. What is multicatching of exceptions and how it can be performed?
2. Will the code snippets below compile? If no, why?

Snippet A

```
try {
    System.out.println("Entered the 'try' block");
} catch (Exception exceptionOne) {
    Logger.error(exceptionOne);
} catch (Exception exceptionTwo) {
    Logger.error(exceptionTwo);
}
```

Answer

No: two same types of exceptions cannot be multicaught

Snippet B.1

```
try {
    throw new IndexOutOfBoundsException("Sample exception");
} catch (ArithmeticException exceptionOne) {
    Logger.error(exceptionOne);
} catch (StackOverflowError exceptionTwo) {
    Logger.error(exceptionTwo);
}
```

Answer

Yes

Snippet B.2

```
try {
    throw new IndexOutOfBoundsException("Sample exception");
} catch (IndexOutOfBoundsException | StackOverflowError exception) {
    Logger.error(exception);
}
```

Answer

Yes

Snippet C

```
try {
    throw new IndexOutOfBoundsException("Sample exception");
} catch (IndexOutOfBoundsException exception) {
    Logger.error(exception);
} catch (ArrayIndexOutOfBoundsException exceptionTwo) {
    Logger.error(exceptionTwo);
}
```

Answer

No: `ArrayIndexOutOfBoundsException` is narrower than `IndexOutOfBoundsException`

Snippet D.1

```
try {
    throw new IndexOutOfBoundsException("Sample exception");
} catch (IndexOutOfBoundsException | ArithmeticException exception) {
    exception.initCause(new ArithmeticException("I'm a cause exception"));
    Logger.error(exception);
}
```

Answer

Yes

Snippet D.2

```
try {
    throw new IndexOutOfBoundsException("Sample exception");
} catch (ArithmeticException exception) {
    exception = new ArrayIndexOutOfBoundsException("New exception");
    Logger.error(exception);
}
```

Answer

Yes

Snippet D.3

```
try {
    throw new IndexOutOfBoundsException("Sample exception");
} catch (IndexOutOfBoundsException | ArithmeticException exception) {
    exception = new ArrayIndexOutOfBoundsException("New exception");
    Logger.error(exception);
}
```

Answer

No: in case of inline multicatching, the `exception` variable is final

Snippet E

```
try {
    throw new IndexOutOfBoundsException("Sample exception");
} catch (IndexOutOfBoundsException exception | StackOverflowError error) {
    System.err.println("Entered the 'catch' block")
}
```

Answer

No: in case of inline multicatch only one `exception` variable is possible

Snippet H

```
try {
    throw new IndexOutOfBoundsException("Sample exception");
} catch (IndexOutOfBoundsException | RuntimeException exception) {
    Logger.error(exception);
}
```

Answer

No: types in the multicatch block must be disjoint (i.e. that one of them cannot be the subclass of the another one)

try-with-resources

1. What types can be used as a resource? Can custom types be used as a resource?
2. Will the code snippets below compile? If no, why?

Snippet A

```
try (null) {
    System.out.println("Entered the 'try' block");
} catch (Exception exception) {
    Logger.error(exception);
}
```

Answer

No: it's impossible to pass `null` as a resource

Snippet B

```
try (StringBuilder stringBuilder = new StringBuilder()){
    stringBuilder.append("Some text");
} catch (Exception exception) {
    Logger.error(exception);
}
```

Answer

No: it's impossible to pass an object that don't implement `AutoCloseable` or `Closeable` as a resource

3. Is it possible to use 2 resources in the try-with-resources block?
4. What is the difference between `AutoCloseable` or `Closeable`?
5. What method must be implemented in classes that implement `AutoCloseable` or `Closeable`?
6. What is the purpose of try-with-resources construct? How does it work?
7. What is the order in which resources are closed?
8. Will the code snippets below compile? If no, why?

Snippet A

```
CustomResource customResource = new CustomResource("ONE.txt");
try (customResource = new CustomResource("ONE.txt")) {
    String text = customResource.readLine();
    System.out.println(text);
} catch (Exception exception) {
    Logger.error(exception);
}
```

Answer

No: resources must be `final`

Snippet B

```
try (CustomResource customResource = new CustomResource("ONE.txt")) {
    customResource = new CustomResource("ONE.txt");
    String text = customResource.readLine();
    System.out.println(text);
} catch (Exception exception) {
    Logger.error(exception);
}
```

Answer

No: resources must be `final`

9. Will the code snippet below compile? If no, why?

```
try (CustomResource customResource = new CustomResource("ONE.txt")) {
    String text = customResource.readLine();
    System.out.println(text);
}
// no 'catch' or 'finally' block
```

Answer

Yes

10. What is exception suppression?
11. Answer the following questions for every code snippet below:

    a. Are there any exceptions that will be suppressed? If yes:

    - What will cause suppressed exceptions?
    - What are those exceptions?
    - What exception will suppress what exception?
    - How to print a stack trace of a suppressed exception?

    b. Will the `catch` block be executed? If yes, what exception will it catch?

    c. What resources are closed after the code execution?

    d. Will the last line be executed? If no, why?

Common code for all snippets

```
/* 'close()' method of FlawedCustomResource looks like this:
@Override
public void close() throws IOException {
    throw new CloseException("Exception during resource closing occurred");
}*/

CustomResource customResourceOne = new CustomResource();
FlawedCustomResource flawedCustomResource = new FlawedCustomResource();
CustomResource customResourceTwo = new CustomResource();
```

Snippet A

```
try (customResourceOne;
     flawedCustomResource; // <- there is an exception thrown when 'close()'
method of this resource is called
     customResourceTwo) {
    customResourceOne.readLine();
    flawedCustomResource.readLine();
    customResourceTwo.readLine();
    throw new IndexOutOfBoundsException("Exception from the 'try' block");
} catch (IndexOutOfBoundsException exception) {
    Logger.error(exception);
}
System.out.println("Hi, my friend!");
```

Answer

- There is a suppressed `CloseException` thrown when autoclosing the `flawedCustomResource`
- The `IndexOutOfBoundsException` will suppress `CloseException`
- To print a stack trace of the suppressed exception, extract the suppressed exception from the caught exception and print its stack trace
- The `catch` block will be executed and will log `IndexOutOfBoundsException`
- Both `CustomResource`s will be closed. `FlawedCustomResource` will not be closed
- The last line will be executed

Snippet B

```
try (customResourceOne;
     flawedCustomResource;
     customResourceTwo) {
    customResourceOne.readLine();
    flawedCustomResource.readLine();
    customResourceTwo.readLine();
} catch (RuntimeException exception) {
    Logger.error(exception);
}
System.out.println("Hi, my friend!");
```

Answer

- There is no suppressed exceptions
- The `catch` block will be executed and will log `CloseException`
- Both `CustomResource`s will be closed. `FlawedCustomResource` will not be closed
- The last line will be executed

Snippet C

```
try (customResourceOne;
     flawedCustomResource;
     customResourceTwo) {
    customResourceOne.readLine();
    flawedCustomResource.readLine();
    customResourceTwo.readLine();
} catch (IndexOutOfBoundsException exception) {
    Logger.error(exception);
}
System.out.println("Hi, my friend!");
```

Answer

- There is no suppressed exceptions
- The `catch` block will not be executed
- Both `CustomResource`s will be closed. `FlawedCustomResource` will not be closed
- The last line will not be executed, because during closing the `FlawedCustomResource` an unexpected exception occurred (`CloseException`) that was propagated to the caller and crashed the program

Snippet D

```
try (customResourceOne;
     flawedCustomResource;
     customResourceTwo) {
    customResourceOne.readLine();
    flawedCustomResource.readLine();
    customResourceTwo.readLine();
    throw new ArithmeticException("I'm an exception from the 'try' block");
} catch (IndexOutOfBoundsException exception) {
    Logger.error(exception);
}
System.out.println("Hi, my friend!");
```

Answer

- There is a suppressed `CloseException` thrown when autoclosing the `flawedCustomResource`
- The `ArithmeticException` will suppress `CloseException`
- To print a stack trace of the suppressed exception, extract the suppressed exception from the `ArithmeticException` and print its stack trace (to do this, the `ArithmeticException` must be handled by the caller since it isn't caught by the `catch` block here)
- The `catch` block will not be executed
- Both `CustomResource`s will be closed. `FlawedCustomResource` will not be closed
- The last line will not be executed, because during the execution of the `try` block, an unexpected exception occurred (`ArithmeticException`) that suppressed the `CloseException` from the `FlawedCustomResource` and was propagated to the caller and crashed the program

Miscellaneous

1. Will the code snippets below compile? If no, why?

Snippet A

```
try {
    System.out.println("Entered the 'try' block");
} catch (Object object) {
    Logger.error(object);
}
```

Answer

Yes, if `Object` extends `Throwable`

Snippet B.1

```
try {
    throw new IndexOutOfBoundsException("Sample exception");
} catch (CompletionException exception) {
    Logger.error(exception);
}
```

Answer

Yes

Snippet B.2

```
try {
   throw new IndexOutOfBoundsException("Sample exception");
} catch (SQLException exception) {
   Logger.error(exception);
}
```

Answer

No: if a checked exception is caught, then the code inside the `try` block must be able at least hypothetically produce the caught checked exception

2. What will happen if an exception thrown from the `catch` will be thrown?
3. If there is an exception from the `catch` will be thrown, will `finally` block be executed?
4. Will the code in the snippets below compile? If no, why? If yes, what will happen to each of the thrown exceptions?

Snippet A

```
try {
    throw new IndexOutOfBoundsException("Exception from the 'try' block");
} catch (ArithmeticException exception) {
    Logger.error(exception);
    throw new IndexOutOfBoundsException("Exception from the 'catch' block");
} finally {
    return new IndexOutOfBoundsException("Exception from the 'finally' block");
}
System.out.println("Hi, my friend!");
```

Answer

Will not compile: returning a value from the `finally` block is not allowed

Snippet B

```
try {
    throw new ArithmeticException("Exception from the 'try' block");
} catch (ArithmeticException exception) {
    Logger.error(exception);
    throw new IndexOutOfBoundsException("Exception from the 'catch' block");
} finally {
    System.out.println("Hi, my friend!");
    return;
}
System.out.println("Hi, my friend!");
```

Answer

Will not compile: the last line is unreachable

Snippet C

```
try {
    throw new ArithmeticException("Exception from the 'try' block");
} catch (ArithmeticException exception) {
    Logger.error(exception);
    throw new IndexOutOfBoundsException("Exception from the 'catch' block");
} finally {
    System.out.println("Hi, my friend!");
    return;
}
```

Answer

Will compile: the exception from the `try` block will be logged and the exception from the `catch` block will be discarded due to the `return` statement inside the `finally` block

5. Is the `try-finally` block, without the `catch` block, possible? If yes, what will happen to the exception from the `try` block?

Handling Caught Exceptions

1. Name 4 good practices regarding handling of caught exceptions?
2. Name 2 bad practices regarding handling of caught exceptions?

Checked vs Unchecked - Controversy

Provide and explain 2 arguments in favor and 2 arguments against the division on checked and unchecked exceptions?