Correction Exceptions and Assertions - 0

D - Replace Line 14 with catch(RuntimeException e) {

In multi-catch statement, classes with multi-level hierarchical relationship can’t be used.

RuntimeException is subclass of Exception, IllegalArgumentException is indirect subclass of Exception and IllegalArgumentException is subclass of RuntimeException, hence these pairs can’t be used in multi-catch statement.

Only one option is left to replace Line 14 with ‘catch(RuntimeException e) {’.

Commenting out Line 14, Line 15 and Line 16 will resolve the compilation error but it will print the whole stack trace rather than just printing the message.

Correction Exceptions and Assertions - 1

D -

```
public class C4 implements I1 {
  public void m1() throws Exception{}
}
```

NOTE: Question is asking for “incorrect” implementation and not “correct” implementation.

According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:

1. May not declare to throw any checked exception,
2. May declare to throw the same checked exception thrown by super class / interface method,
3. May declare to throw the sub class of the exception thrown by super class / interface method,
4. Cannot declare to throw the super class of the exception thrown by super class / interface method.

Correction Exceptions and Assertions - 2

B -

```
Inner
Finally 1
Finally 2
```

System.out.println(1/0); throws ArithmeticException, handler is available in inner catch block, it executes and prints “Inner” to the console.

Once we handled the exception, no other catch block will get executed unless we re-throw the exception.

Inner finally gets executed and prints “Finally 1” to the console.

Rule is finally block always gets executed, so outer finally gets executed and prints “Finally 2” to the console.

Correction Exceptions and Assertions - 3

B - Compilation Error

Classes used in try-with-resources statement must implement java.lang.AutoCloseable or its sub interfaces such as java.io.Closeable.

As Resource1 and Resource2 don’t implement AutoCloseable interface, hence try-with-resources statement causes compilation error.

Correction Exceptions and Assertions - 4

A - On execution program terminates successfully after printing ‘HELLO’ on to the console

Even though Scanner is created in try-with-resources block, calling close() method explicitly doesn’t cause any problem.

Scanner class allows to invoke close() method multiple times. In this case, it will be called 3 times: twice because of scan.close() and once because of try-with-resources statement.

‘HELLO’ is printed on to the console and program terminates successfully.

Correction Exceptions and Assertions - 5

B - null

assert 1 == 2 : 2 == 2; => throws AssertionError and as 2 == 2 is true, hence message is set as true.

This doesn’t make any changes to cause, which is still null.

If right side of the expression is an instance of Throwable type, then cause is set to that type.

main method catches AssertionError (though you are not supposed to handle Error and its subtype) and ‘ae.getCause()’ returns null.

Correction Exceptions and Assertions - 6

D - Caught Successfully.

Even though it seems like method m() will not compile successfully, but starting with JDK 7, it is allowed to use super class reference variable in throw statement referring to sub class Exception object.

In this case, method m() throws SQLException and compiler knows that variable e (Exception type) refers to an instance of SQLException only and hence allows it.

Program executes successfully and prints ‘Caught Successfully.’ on to the console.

Correction Exceptions and Assertions - 7

C - Compilation error in Test class

close() method in AutoCloseable interface has below declaration:

void close() throws Exception;

MyResource class correctly overrides close() method.

try-with-resources statement internally invokes resource.close() method.

resource is of AutoCloseable type, so compiler checks the close() method declaration of AutoCloseable interface.

close() method in AutoCloseable interface declares to throw Exception (checked exception) and hence handle or declare rule must be followed.

As main method neither declares to throw Exception nor provides catch block for Exception type, hence try-with-resources statement causes compilation error.

Correction Exceptions and Assertions - 8

A - Compilation error

Wrong syntax for multi-catch block, only one reference variable is allowed.

Correct syntax of multi-catch statement is: ‘catch(MyException | YourException e)’

Correction Exceptions and Assertions - 9

B - SQLException

execute() method throws an instance of SQLException.

Just before finding the matching handler, Java runtime executes close() method. This method throws an instance of IOException but it gets suppressed and an instance of SQLException is thrown.

e.getMessage() prints SQLException on to the console.

NOTE: e.getSuppressed() returns Throwable [] and this helps to get all the suppressed exceptions.

Correction Exceptions and Assertions - 10

B - Surround Line 11 with below try-catch block:

```
try {
  ReadTheFile.print();
} catch(Exception e) { (Correc|
  e.printStackTrace();
}
```

F - Replace Line 4 with static void print() throws Exception {

This question is tricky as 2 changes are related and not independent. Let's first check the reason for compilation error. Line 5 throws a checked exception, IOException but it is not declared in the throws clause. So, print method should have throws clause for IOException or the classes in top hierarchy such as Exception or Throwable.

Based on this deduction, Line 4 can be replaced with either "static void print() throws Exception {" or "static void print() throws Throwable" but we will have to select one out of these as after replacing Line 4, Line 11 will start giving error as we are not handling the checked exception at Line 11.

This part is easy, do we have other options, which mention "Throwable"? NO. Then mark the first option as "Replace Line 4 with static void print() throws Exception {".

As, print() method throws Exception, so main method should handle Exception or its super type and not it's subtype. Two options working only with IOException can be ruled out.

Multi-catch statement "catch(IOException | Exception e)" causes compilation error as IOException and Exception are related to each other in multilevel inheritance. So you are left with only one option to pair with our 1st choice:

Surround Line 11 with below try-catch block:

```
try {
    ReadTheFile.print();
} catch(Exception e) {
    e.printStackTrace();
}
```

Correction Exceptions and Assertions - 11

C - Compilation error

Scope of writer variable is with-in the boundary of try-with-resources block.

It is not accessible inside catch block and hence ‘writer.close()’ causes compilation failure.

NOTE: PrintWriter constructor and println method don’t throw any exception but it is OK to catch Exception type.

Compiler allows to catch Exception type even if code within try block doesn’t throw any exception.

Correction Exceptions and Assertions - 12

A - NullPointerException is thrown and program ends abruptly

At runtime, first statement inside checkStatus(STATUS) method is switch(status) and as status is null, NullPointerException is thrown and program ends abruptly.

Correction Exceptions and Assertions - 13

C - AECB

AutoCloseable interface has abstract method: void close() throws Exception;

Both Resource1 and Resource2 implement the close() method correctly and main method specified handler for Exception type, hence no compilation error.

Resources are always closes, even in case of exceptions. And in case of multiple resources, these are closed in the reverse order of their declaration. So r2 is closed first and then r1. Output will have ‘EC’ together.

r1.m1(); prints ‘A’ on to the console. An exception (with message ‘B’) is thrown so close methods are invoked.

After close() methods of r2 and r1 are invoked successfully, output will be: ‘AEC’.

Exception is caught in main method and e.getMessage() returns ‘B’.

So the overall output will be; ‘AECB’.

Correction Exceptions and Assertions - 14

Correction Exceptions and Assertions - 15

A -

```
Test
Resource2
Resource1
```

Resources are closed in the reverse order of their declaration.

So r2 is closed first and then r1.

Correction Exceptions and Assertions - 16

C - Method m() causes compilation error.

If you don’t initialize variable e inside catch block using ‘e = new SQLException();’ and simply throw e, then code would compile successfully as compiler is certain that e would refer to an instance of SQLException only.

But the moment compiler finds ‘e = new SQLException();’, ‘throw e;’ causes compilation error as at runtime e may refer to any Exception type.

Correction Exceptions and Assertions - 17

B - NO

close() method of FileReader class throws IOException, which is a checked exception and hence handle or declare rule applies in this case.

As main method neither declares to throw IOException nor a catch block is available for IOException, hence code doesn’t compile.

Correction Exceptions and Assertions - 18

C - No output and program terminates successfully

Assertions are disabled by default, so assert false; statement is not executed.

No output is shown and program terminates successfully.

If above program is run with -ea option, the ‘TWO’ will be printed on to the console as AssertionError extends Error.

NOTE: It is not a good programming practice to validate user input using assertion.

Correction Exceptions and Assertions - 19

B - AE

Any RuntimeException can be thrown without any need it to be declared in throws clause of surrounding method.

‘throw (RuntimeException)e;’ doesn’t cause any compilation error.

Even though variable ‘e’ is type casted to RuntimeException but exception object is still of ArithmeticException, which is caught in main method and ‘AE’ is printed to the console.

Correction Exceptions and Assertions - 20

C - HELLO

For null resources, close() method is not called and hence NullPointerException is not thrown at runtime.

HELLO is printed on to the console.

Correction Exceptions and Assertions - 21

B - None of the System.out.println statements are executed

main(args) method is invoked recursively without specifying any exit condition, so this code ultimately throws java.lang.StackOverflowError. StackOverflowError is a subclass of Error type and not Exception type, hence it is not handled. Stack trace is printed to the console and program ends abruptly.

Java doesn't allow to catch specific checked exceptions if these are not thrown by the statements inside try block.

catch(java.io.FileNotFoundException ex) {} will cause compilation error in this case as main(args); will never throw FileNotFoundException. But Java allows to catch Exception type, hence catch (Exception ex) {} doesn't cause any compilation error.

Correction Exceptions and Assertions - 22

D - Program ends abruptly

Classes in Exception framework are normal java classes, hence null can be used wherever instances of Exception classes are used, so Line 10 compiles successfully.

No issues with Line 16 as method m() declares to throw SQLException and main method code correctly handles it.

Program compiles successfully but on execution, NullPointerException is thrown, stack trace is printed on to the console and program ends abruptly.

If you debug the code, you would find that internal routine for throwing null exception causes NullPointerException.

Correction Exceptions and Assertions - 23

B - Compilation error

In a multi-catch block, type of reference variable (ex in this case) is the common base class of all the Exception types mentioned, ‘MyException’ and ‘YourException’.

This means ex is of RuntimeException. Method log() is not available in RuntimeException or its super classes.

ex.log(); causes compilation failure.

Correction Exceptions and Assertions - 24

A - IOException

execute() method throws an instance of SQLException.

Just before finding the matching handler, Java runtime executes close() method. This method throws an instance of IOException but it gets suppressed and an instance of SQLException is thrown.

e.getSuppressed() returns Throwable [] whose 1st element is the instance of suppressed exception, IOException.

‘e.getSuppressed()[0].getMessage()’ prints IOException on to the console.

Correction Exceptions and Assertions - 25

B - AssertionError is thrown at runtime and program terminates abruptly

On execution, ‘args.length == 0’ returns true and ‘assert false : changeMsg(“Bye”);’ is executed.

‘changeMsg(“Bye”)’ method makes msg to refer to “Bye” and null is returned.

An instance of AssertionError is thrown with message null.

Remaining statements are ignored and program terminates abruptly.

NOTE: Though this class compiles successfully but Assertion mechanism is not implemented appropriately. There are 2 issues:

1. Assertion mechanism is used to validate the arguments of public method and
2. changeMsg(“Bye”) method causes side effect as it changes the class variable, ‘msg’.

Correction Exceptions and Assertions - 26

D - Compilation error

Method m1() throws Exception (checked) and it declares to throw it, so no issues with method m1().

But main() method neither provides catch handler nor throws clause and hence main method causes compilation error.

Handle or Declare rule should be followed for checked exception if you are not re-throwing it.

Correction Exceptions and Assertions - 27

D -

```
Inner
Finally 1
Outer
```

System.out.println(1/0); throws ArithmeticException, handler is available in inner catch block, it executes and prints “Inner” to the console.

throw e; re-throws the exception.

But before exception instance is forwarded to outer catch block, inner finally gets executed and prints “Finally 1” to the console.

In outer try-catch block, handler for ArithmeticException is available, so outer catch block gets executed and prints “Outer” to the console.

Correction Exceptions and Assertions - 28

C - Compilation error

Even though an instance of FileNotFoundException is thrown by method m1() at runtime, but method m1() declares to throw IOException.

Reference variable s is of Super type and hence for compiler call to s.m1(); is to method m1() of Super, which throws IOException.

And as IOException is checked exception hence calling code should handle it.

As calling code doesn't handle IOException or its super type, so s.m1(); causes compilation error. Even though an instance of FileNotFoundException is thrown by method m1() at runtime, but method m1() declares to throw IOException.

Reference variable s is of Super type and hence for compiler call to s.m1(); is to method m1() of Super, which throws IOException.

And as IOException is checked exception hence calling code should handle it.

As calling code doesn't handle IOException or its super type, so s.m1(); causes compilation error.

Correction Exceptions and Assertions - 29

D - Derived: m1()

NullPointerException extends RuntimeException, but there are no overriding rules related to unchecked exceptions.

So, method m1() in Derived class correctly overrides Base class method.

Rest is simple polymorphism. obj refers to an instance of Derived class and hence obj.m1(); invokes method m1() of Derived class, which prints “Derived: m1()” to the console.

Correction Exceptions and Assertions - 30

B - No output and program terminates successfully

‘java -ea:com.training…’ enables the assertion in com.training package and its sub packages. Test class is defined under ‘com.training.ocp’ package, hence assertion is enabled for Test class.

‘assert flag = true : flag = false;’ => On the left side ‘flag = true’ is a valid boolean expression, so no issues and on right side ‘flag = false’ assigns false to flag and false is returned as well, which is not void. Hence no issues with right side as well.

On execution, flag is true, hence AssertionError is not thrown.

Nothing is printed on to the console and program terminates successfully.

Correction Exceptions and Assertions - 31

B - Compilation error

Resources used in try-with-resources statement are implicitly final, which means they can’t be reassigned.

scan = null; will fail to compile as we are trying to assign null to variable scan.

Correction Exceptions and Assertions - 32

B -

```
Executing
Closing
```

close() method in AutoCloseable interface has below declaration:

void close() throws Exception;

MyResource class correctly overrides close() method.

try-with-resources statement internally invokes resource.close() method after executing resource.execute().

Output is:

```
Executing
Closing
```

Correction Exceptions and Assertions - 33

A - Compilation error

Variable r is implicitly final and hence can’t be re-initialized.

‘r = new Resource();’ causes compilation error.

Correction Exceptions and Assertions - 34

C - XZ

Even though method m1() declares to throw IOException but at runtime an instance of FileNotFoundException is thrown.

A catch handler for FileNotFoundException is available and hence X is printed on to the console.

After that finally block is executed, which prints Z to the console.

Correction Exceptions and Assertions - 35

C -

```
1
java.lang.ClassCastException
```

Line n3 throws an instance of RuntimeException. As catch(RuntimeException e) is available, hence control starts executing catch-block inside check() method.

1 is printed on to the console.

At Line n4, instance of super-class (RuntimeException) is type-casted to sub-class (ArithmeticException), hence Line n4 throws an instance of ClassCastException.

ClassCastException is a sub-class of RuntimeException, so catch-block of main method is executed and Line n1 prints the fully qualified name of ClassCastException. java.lang.ClassCastException is printed on to the console.

Correction Exceptions and Assertions - 36

C - Compilation error

NullPointerException extends RuntimeException and in multi-catch syntax we can’t specify multiple Exceptions related to each other in multilevel inheritance.

Correction Exceptions and Assertions - 37

A - Compilation error

Variable ‘e’ used in multi-catch block is implicitly final and can’t be re-initialized.

e = null; causes compilation failure.

Correction Exceptions and Assertions - 38

E - Line 11 causes compilation failure

Exception is a java class, so e = null; is a valid statement and compiles successfully.

If you comment Line 10, and simply throw e, then code would compile successfully as compiler is certain that e would refer to an instance of SQLException only.

But the moment compiler finds ‘e = null;’, ‘throw e;’ (Line 11) causes compilation error as at runtime e may refer to any Exception type.

NOTE: No issues with Line 17 as method m() declares to throw SQLException and main method code correctly handles it.

Correction Exceptions and Assertions - 39

C - MyException2

If finally block throws exception, then exception thrown by try or catch block is ignored.

In this case, method m() throws an instance of MyException2 class.

Correction Exceptions and Assertions - 40

C - Compilation Error

close() method in AutoCloseable interface has below declaration:

void close() throws Exception;

MyResource class correctly overrides close() method.

try-with-resources statement internally invokes resource.close() method after executing resource.execute().

Overriding close method declares to throw Exception (checked exception) and hence handle or declare rule must be followed.

As main method neither declares to throw Exception nor provides catch block for Exception type, hence try-with-resources statement causes compilation error.

Correction Exceptions and Assertions - 41

C - assert 1 == 2 : 1;

assert 1 == 2 : () -> “a”; => Right expression can’t specify any lambda expression.

assert 1 == 2 : return 1; => Right expression can’t use return keyword, it should just specify the value.

assert 1 == 2 : Test::get; => Right expression can’t specify any method reference syntax.

assert 1 == 2 : 1; => Legal. Left expression ‘1 == 2’ returns boolean and right expression ‘1’ is a non-void value.

Correction Exceptions and Assertions - 42

C - Runtime Exception

Even though Scanner is created in try-with-resources block, calling close() method explicitly doesn’t cause any problem.

Scanner class allows to invoke close() method multiple times.

But once Scanner object is closed, other search operations should not be invoked. If invoked on closed scanner, IllegalStateException is thrown.

Correction Exceptions and Assertions - 43

C - Compilation error

Java doesn’t allow to catch specific checked exceptions if these are not thrown by the statements inside try block.

catch(FileNotFoundException ex) {} causes compilation error in this case as System.out.println(1); will never throw FileNotFoundException.

NOTE: Java allows to catch Exception type. catch(Exception ex) {} will never cause compilation error.

Correction Exceptions and Assertions - 44

A - Compilation error

Resources used in try-with-resources statement must be initialized.

‘try(PrintWriter writer;)’ causes compilation error as writer is not initialized in this statement.

Correction Exceptions and Assertions - 45

C - A is printed to the console, stack trace is printed and then program ends abruptly

Method m1() throws an instance of ArithmeticException and method m1() doesn’t handle it, so it forwards the exception to calling method main.

Method main doesn’t handle ArithmeticException so it forwards it to JVM, but just before that finally block is executed. This prints A on to the console.

After that JVM prints the stack trace and terminates the program abruptly.

Correction Exceptions and Assertions - 46

C - throw

catch is for catching the exception and not throwing it.

thrown is not a java keyword.

throws is used to declare the exceptions a method can throw.

To manually throw an exception, throw keyword is used. e.g., throw new Exception();

Correction Exceptions and Assertions - 47

A - Compilation error

throw ex; causes compilation error as div method doesn’t declare to throw Exception (checked) type.

Correction Exceptions and Assertions - 48

C - Exception

Method m1() throws an instance of TestException, which is a checked exception as it extends Exception class.

So in throws clause we must provide:

1. Checked exception.
2. Exception of TestException type or it's super types (Exception, Throwable), Object cannot be used in throws clause.

Correction Exceptions and Assertions - 49

A - true

assert 1 == 2 : 2 == 2; => throws AssertionError and as 2 == 2 is true, hence message is set as true.

main method catches AssertionError (though you are not supposed to handle Error and its subtype) and ‘ae.getMessage()’ returns true.

Correction Exceptions and Assertions - 50

A - AssertionError is thrown and program terminates abruptly

‘java -ea:com.training…’ enables the assertion in com.training package and its sub packages.

Test class is defined under ‘com.training.ocp’ package, hence assertion is enabled for Test class.

assert flag : flag = true; => flag is a boolean variable, so ‘assert flag’ has no issues and right expression must not be void.

‘flag = true’ assigns true to flag and true is returned as well. Hence no issues with right side as well.

On execution, flag is false, hence AssertionError is thrown and program terminates abruptly.

Correction Exceptions and Assertions - 51

B - 0

execute() method throws an instance of SQLException.

Just before finding the matching handler, Java runtime executes close() method. This method executes successfully.

An instance of SQLException is thrown. No exceptions was suppressed so ‘e.getSuppressed()’ returns Throwable [] of size 0.

‘e.getSuppressed().length’ prints 0 on to the console.

Correction Exceptions and Assertions - 52

D - HELLO

For null resources, close() method is not called and hence NullPointerException is not thrown at runtime.

HELLO is printed on to the console.

Correction Exceptions and Assertions - 53

A - class Test causes compilation error

FileNotFoundException extends IOException and hence catch block of FileNotFoundException should appear before the catch block of IOException.

Hence, class Test causes compilation error.

Correction Exceptions and Assertions - 54

C - Program ends abruptly

Classes in Exception framework are normal java classes, hence null can be used wherever instances of Exception classes are used, so Line 7 compiles successfully.

No issues with Line 12 as method m() declares to throw SQLException and main method code correctly handles it.

Program compiles successfully but on execution, NullPointerException is thrown, stack trace is printed on to the console and program ends abruptly.

If you debug the code, you would find that internal routine for throwing null exception causes NullPointerException.