Homework 8

Q1)(D)

The code does not compile but not because of line "k1", try block must include either a catch block or a finally block.We can have both catch and finally block too if we prefer. Java uses a try statement to separate the logic that might throw an exception from the logic to handle that exception. The code in the try block is run normally. If any of the statements throw an exception that can be caught by the exception type listed in the catch block, the try block stops running and execution goes to the catch statement. If none of the statements in the try block throw an exception that can caught, the catch clause is not run. The try statement also lets us run code at the end with a finally clause regardless of whether an exception is thrown.

Lastly we should note that, having the missing catch or finally block added, there is no need to bubble up the exception to the upper levels in method call-stack. We are having the exception handled so thus no need to delegate it to someone else to handle it.

Q2)(B)

The order of the keywords of a try statement is try, catch and finally respectively. The order of the elements is not interchangeable. Either catch or finally block must be present, each coming after the try block, if both of them are present, finally coming after the catch block. Using the keywords with the wrong order will result in a compilation error.

Q3)(D)

Java has a Throwable superclass for all objects that represent exception events. Below the inheritance hierarchy, there are Exception and Error classes both extending the Throwable class, and RuntimeException class extending the Exception class. So, the correct representation of that class hierarchy is given in option D.

Q4)(A)

An error indicates serious problems that a reasonable application should not try to catch. Both Errors and Exceptions are the subclasses of java.lang.Throwable class. Errors are the conditions which cannot get recovered by any handling techniques. It surely cause termination of the program abnormally. Errors belong to unchecked type. Some of the examples of errors are out of memory error or a system crash error.

An error usually shouldn't be caught, as it indicates an abnormal condition that should never occur. We can never be sure that the application will be able to execute the next line of code. If we get an OutofMemoryError, we have no guarantee that we will be able to do

anything reliably. We should try to catch RuntimeException and checked exceptions, but not Errors.

Q5)(D)

The code does not compile. score variable is declared in try block, it's scope is limited to that block, so it does not exist and thus accessible outside of that block. Let's say that we declared it as a method local, outside of try block, then the code would print "123". try block prints and then increment, catch block does not get executed because no exception was thrown, finally block always get executed after try block or if there exists, after catch block too. Last but not least, outside of try-catch statement, one last print statement occurs, hence "123" gets printed.

Q6)(B)

RuntimeException class, Error class and all of their subclasses are unchecked. Exception class and all classes that extends it are checked exceptions. IOException is one of the checked exceptions. Other three exceptions given as options are runtime exceptions.

Q7)(A)

Throws clause is used to declare an exception, which means it works similar to the try-catch block. On the other hand throw keyword is used to throw an exception explicitly.

Throw keyword is used in the method body to throw an exception, while throws keyword is used in method signature to declare the exceptions that can occur in the statements present in the method.

Q8)(B)

The catch block for IOException must appear before the catch block for exception. A rule exists for the order of the catch blocks. Java looks at them in the order they appear. If it is impossible for one of the catch blocks to be executed, a compiler error about unreachable code occurs. This happens when a superclass is caught before a subclass. Because any thrown exception would be handled by the superclass exception, thus subclass exception would be considered unreachable code by the compiler.

Q9)(D)?

None of the above, "throw t;" statement causes a compilation error , because t variable is not defined hence cannot be resolved to a variable, resulting in a compilation error. Removing, or commenting that line code would print "AC" followed by a stack trace for a RuntimeException. Catch block does not get executed because ArrayOutOfBoundsException is subclass of RuntimeException, if it was its superclass then

catch block would get executed. So, to summarize, try block gets executed, prints 'A' then throw RuntimeException. Catch block can't handle that exception so it's passed. Finally block gets executed, printing 'C' to the screen, after that we see the RuntimeException is thrown by the main method and a stack trace is printed to the screen.

Q10)(C)

The code does not compile because of line p3. openDrawbridge declares that it throws and exception, delegating the handling up in the method-call stack. main() method should declare compatible exception with throws keyword in the method signature or handle the exception itself with a try-catch block.

Q11)(B)

Exception class and its descendants excluding RuntimeException class are checked exceptions. Checked exceptions are the exceptions that are checked at compile time. If some code within the method throws a checked exception, then the method must either handle the exception or it must specify the exception in method signature using throws keyword. All of the RuntimeException class objects and its descendants are unchecked exceptions, including NullPointerException, ArithmeticException.

Q12)(A)

The code compiles and runs without any issue. First catch block is skipped because ClassCastException is not a subclass of ArrayIndexOutOfBoundsException class. Second catch block gets executed because Exception class extends Throwable class and all other exceptions extends Exception class, so ClassCastException is indirect child of Throwable class. finally block is always executed, printing '4' and after finally block, control is given back to the main method which then prints '5' to the screen, without throwing any exception because it was handled with the second catch block.

Q13)(C)

The try statement lets run code at the end with a finally clause regardless of whether an exception is thrown. There are two paths through code with both a catch and a finally. If an exception is thrown, the finally block is run after the catch block. If no exception is thrown, the finally block is run after the try block completes. Finally clauses can throw exceptions, thus there can be unreachable code, meaning not every line is guaranteed to be executed.

The curly braces are required for the try and catch blocks. try statements are like methods in that the curly braces are required even if there is only one statement inside the code blocks. if statements and loops are special in this respect as they allow you to omit the curly braces.

A catch or finally block can have any valid Java code in it, including throwing exceptions.

Q14)(C)

The code does not compile. If it is impossible for one of the catch blocks to be executed, a compiler error about unreachable code occurs. This happens when a superclass is caught before a subclass.

We see that in the question FileNotFoundException is already handled by the catch block for IOException which comes before the catch block for FileNotFoundException. So, the catch block for FileNotFoundException is unreachable.

Q15)(C)

Finalize is a method inherited from java.lang.Object, invoked just before the object is garbage collected. It has nothing to with exception handling. A try statement requires a catch block or a finally block at least. They can be both present optionally.

Q16)(B)

It's not necessary for an application to terminate if it throws an application. It is true that some exceptions can be avoided programmatically, for example null checks can be done to avoid NullPointerExceptions. It is also true that properly handled exceptions may lead to recovery from unexpected problems. It is true that exceptions are often used when thing go wrong or deviate from the expected path. Option B is certainly wrong, because a method that throws an exception can handle the exception to avoid termination.

Q17)(D)

The code does not compile due to an syntax error concerning the catch block. Catch block uses parentheses instead of brackets, this is a syntax error, hence the code does not compile. But let's assume it is corrected, then the code would print '4' because the travel method invoked with boat object return 4. It does not throw an exception, it just declares that it may.

Transport class is irrelevant here. If the Boat class was to extend the Transport class, it would have been an invalid override of the function travel(), because the overridden function can not throw a broader type exception.

Q18)(B)

Overriding method is not allowed to throw a broader type exception or a new exception. This violates the rules for checked exceptions. It is allowed for RuntimeExceptions though, because they are not checked exceptions.

Q19)(D)

Mentioned classes in the question are located in the java.lang package, but since the compiler imports that package by default there is no need to explicitly import java.lang package and their classes.

Q20)(C)

The code does not compile because of line g3. Catch block misses parentheses and exception type inside it which indicates the type of the exception to be handled. Having corrected that code would compile without any issue. Overriding method can omit the exception declared in the overridden method. That does not violate the checked exception mechanism.

Q21)(B)

A program should handle or declare checked exceptions but should never handle java.lang.Error. Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keyword, otherwise the program will give a compilation error. Checked exceptions are usually intended to signal recoverable errors, which are not caused by a programmer error (like a file not being there, or a network connectivity problem). RuntimeExceptions are usually intended to signal non-recoverable errors. They don't force the caller to handle or declare them. And many of them indeed signal programming error (like NullPointerException). It is a general belief that checking by compiler should be done for the exceptions whose occurrence is outside programmer's control. Unchecked exceptions are "not supposed to happen", so the compiler doesn't force you to declare them.

An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.

Q22)(B)

The code does not compile because of line q2. The code throws an checked exception at line q2, but there is no try-catch block handling it or compatible exception type declaration in method signature. ClassCastException is an unchecked exception so it does not need handling nor declaration, but we should also note that even if that exception would be an checked exception because the finally block throws another exception it would mask the thrown exception in the catch block. Lastly, main method does not have to declare or handle the declared exception in the openDrawBridge method because it declares a RuntimeException, RuntimeExceptions does not need to be handled nor declared.

Q23)(A)

If an exception matches two or more catch blocks, the first one that matches is executed. But we should be careful not to cause any unreachable code. That happens when a catch block which handles the superclass exception comes before the subclass catch block.

Q24)(C)

The code does not compile because of an unhandled checked exception. Although compute method throws an RuntimeException it declares an checked exception so the main method must either declare the exception or handle it. There is a try-catch statement in the main method but it does not catch the declared Exception, nor the main method declares that it throws Exception, so it remains unhandled.

Q25)(D)

All three options are possible with the given code. If the list length is less than 10 than the code would throw an ArrayIndexOutOfBounds exception. If the list is null than the method would throw a NullPointerException. Lastly, if the list is being tried to be assigned to an incompatible type with an explicit cast than the code would throw and ClassCastException.

Q26)(B)

The common cause for a stack overflow is a bad recursive call. Typically, this is caused when your recursive functions doesn't have the correct termination condition, so it ends up calling itself forever. When a function call is invoked by a Java application, a stack frame is allocated on the call stack. The stack frame contains the parameters of the invoked method, its local parameters, and the return address of the method. The return address denotes the execution point from which, the program execution shall continue after the invoked method returns. If there is no space for a new stack frame then, the StackOverflowError is thrown by the Java Virtual Machine.

The java.lang.NullPointerException is thrown when a reference variable is accessed (or de-referenced) and is not pointing to any object. This error can be resolved by using a try-catch block or an if-else condition to check if a reference variable is null before dereferencing it.

NoClassDefFoundError is caused when there is a class file that your code depends on and it is present at compile time but not found at runtime. Look for differences in your build time and runtime classpaths.

Q27)(C)

Checked exceptions often used to force a caller to handle or declare its exceptions, without declaring or handling checked exceptions code won't compile, by doing that we also notify the caller of potential types of problems. We are also giving caller a chance to recover from an exception by making the caller obligated to either declare or handle the exception. Option C is the correct option since there may not be a way to handle the exception other than to terminate the application.

Q)28(D)

This code does not compile because of an syntax error. Catch and finally block cannot be interchangeably placed. If there exists a catch block, finally block must come after catch block. If we were to reverse the order of the keywords than the code would compile without any issue and print "Finished!Joy Hopper". Notice that the try block does not print anything, it just returns the concatenated string. Finally block runs and prints "Finished!" and then the method finishes its execution, resulting in control flow returned to the caller which is the main method. Main method then use the returned string from the getFullName() method and passes it as parameter to print method.

Q29)(A)

A try-catch statement may not have a finally block defined, but if it does, there can only be at most one finally block. If a finally block exists, a try-catch statement is free to not define any catch block. There is no limitation on the number of catch blocks we can have.

Q30)(D)

The code compiles but throws an exception at runtime. Notice that the instance variable count in the superclass Duck is initialized with default value of 0. Than the Overrider method getDuckies tries to divide instance variable age with inherited variable count, which is zero unless assigned otherwise. In the main method primitive 5 is passed to constructor when creating a new Ducklings object, than the getDuckies method is invoked, which ends up dividing 5 to 0, throwing an AritmaticException in process.

Q31)(B)

If both catch block and finally block throws and exception, exception from the finally block is bubbled up to the caller, the exception thrown from the catch block is forgotten about. Option C is wrong because only one exception can be thrown.

Q32)(A)

The code does not compile because of line m1. Mistyped keyword causes a compilation error, “throw” is used instead of “throws” keyword. If the mistyped keyword corrected code would compile without any issue. Note that roar() method is not overridden but instead overloaded. Overloaded methods are not polymorphic methods of course. Because roar() method is invoked with integer value 2, it certainly throws IllegalArgumentException, printing a stack trace along.

Q33)(A)

Java ClassCastException is thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance. This is what exactly is happening here. exception variable points to an object of type Exception, next line tries to cast to a RuntimeException type, which is a subclass of Exception.

Q34)(C)

All objects within the Java hierarchy extend from the Throwable superclass. Only instances (or an inherited subclass) of Throwable are indirectly thrown by the Java Virtual Machine, or can be directly thrown via a throw statement. Every possible built-in exception class extends directly (or subclasses) from Throwable superclass. The Java exception hierarchy was built around two distinct categories: Errors and Exceptions.

Exception and Error class extends Throwable class. RuntimeException class extends Exception class. So, Throwable sits on the top of the hierarchy.

Q35)(B)

First of, both given parameters should be valid non-null String objects or else the code would throw an NullPointerException. In both cases, finally will be executed first.

Considering both parameters are non-null String objects, only the second (II) is an possible output. First one is not possible because, because getAdress() method returns the concatenated string. First, “Posted:” is printed, then the control is returned to the main method with the returned, concatenated string and then the returned string is printed.

Q36)(A)

ClassCastException is a subtype of RuntimeException. These two exception types can not be declared in any order. The catch block for ClassCastException must appear before the catch block for RuntimeException. If not the catch block for superclass type runs, which means there is no way for the second catch block to ever run. Java correctly tells us there is an unreachable catch block.

Q37)(C)

Option A is a completely irrelevant option. Option B tells us there is something wrong with the code, maybe a syntax error etc. hence we are obligated to correct it to be able to throw exceptions in the first place. A method finishes sooner than expected will most likely not considered to be a problem. A caller passes invalid data to a method can be considered as a programming error. There exists a rule which considered to be a good practice says: Use checked exceptions for recoverable conditions and runtime exceptions for programming errors (Effective Java). Option C is the correct answer since RuntimeException IllegalArgumentsException seems to be a good way to use in that situation.

Q38)(C)

The code does not compile because the subclass overridden method returns an incompatible checked exception. Note that it does not matter the superclass method throws a unchecked exception. Superclass could also be not throwing any exception at all. Overridden method declaring a checked exception in this case violates the checked exception system of java. But, if the superclass method was to throw a checked exception, then it would be okay for the subclass overridden method to declare an unchecked exception (since they are not checked at compile time), or not to declare any exception at all, but vice versa is not true.

Q39)(D)

NullPointerException is a RuntimeException, which means it is an unchecked exception hence it is not necessary to handle or declare that exception. Because unchecked exceptions are not checked at compile time, when a method calls another method which may throw an unchecked exception, is not obligated to declare a compatible exception. Enclosing method may choose to handle this NullPointerException with a try-catch block or it may ignore it completely.

Q40)(D)

The code does not compile because the new keyword is forgotten when throwing RuntimeException in zipper() method's catch block.Since exceptions are objects, new keyword is required to create them. Having the new keyword added properly, the try block would throw an ClassCastException, and the catch block itself will throw RuntimeException which will replace ClassCastException. Throwed RuntimeException then will be catched by the catch block in the main main method which is an empty block. Note that the print statement in the try block comes after the method call, considering the zipper method will throw RuntimeException, this print statement will not get executed, control will be skipped to catch block. In this case, the correct answer would be Option C.

Q41)(C)

First, try block will throw ClassCastException which is an subclass of RuntimeException, second catch block will catch that exception and will throw a NullpointerException itself. Notice that, even if we have a NullPointerException thrown, finally block is going to be executed nonetheless and throw RuntimeException, replacing the preceding NullPointerException (or any other exception). Only one thrown exception can be bubbled up in the method call-stack.

Q42)(D)

Notice that in the interface Outfielder, the catchBall() method is defined with a void return type. But the signatures given in the options all have int return types in them which is not compatible with return type void. All of these signatures are invalid overrides. So none of them is correct. But, if the return types were to be void, then the correct answer would be A, because OutOfBoundsException is the exception itself thrown in the overridden method. BadCatchException and Exception classes are not compatible because the an implementer class of OutFielder interface must throw the same or a more specific type of exception. BadCatchException and Exception are superclasses of OutOfBoundsException.

Q43)(D)

The code does not compile due to an syntax error. There is no variable name declared in the catch block parentheses. If we correct that syntax error code would compile and run without error. Try block would throw IllegalArgumentException but catch block only handles Errors so it is skipped. IllegalArgumentException would then be sent to the caller and we would see IllegalArgumentException in the stack trace.

Q44)(D)

The code compiles and runs but a stack trace is printed at runtime. Try block throws an Exception but it is not caught but the catch block since it is not a RuntimeException. Finally block executes, throwing a RuntimeException object and replacing the exception thrown in the try block, sent to the caller method. Stack trace is printed as a result.

Q45)(C)

ClassCastException and IllegalArgumentException both are RuntimeException. But there is no relationship between each other. Neither is a superclass of the other. So, the catch blocks for these two exception types can be declared in any order.

Q46)(D)

The code does not compile, because Problem class tries to implement the RuntimeException class. A class should be extended not implemented, interfaces are implemented. If we correct that syntax error, then the code would print "Problem?Fixed!". Try block throws a BiggerProblem exception handled by the first catch block printing "Problem?". Lastly finally block gets executed after the catch block and prints "Fixed!". Note that the catch blocks handling exceptions are placed in the right order. BiggerProblem extends Problem so it should be handled first for not to cause unreachable code.

Q47)(D)

The code does not compile because of a syntax error. throws keyword is used instead of throw keyword. throws keyword is used to declare exceptions in method signatures, not to throw exceptions. If we change the keyword to the correct one, the correct option would be B. The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling. It allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

Q48)(D)

Usually, running out of memory is an unrecoverable state, hence would most likely result in a java.lang.Error being thrown. Two users try to register an account at the same time and a user entering his/her password incorrectly are situations that is encountered in almost every application. Option B seems like it would throw an Error, but it says in the option temporarily.

Q49)(C)

The code does not compile because of line z1. Same parameter name is used for both outer catch and inner catch. We cannot use it since we are declaring it in the same scope twice. We could declare it twice with two catch blocks, provided that the catch block are on the same level of scope. Having the error corrected, code would print "Failed", first catch block handling the IOException and then throwing an FileNotFoundException in an another inner try-catch block, and inner catch block handling the FileNotFoundException with printing "Failed".

Q50)(B)

The code does not compile because of line x1. Finally block throws an checked exception which is neither declared in the method signature nor handled with a try-catch block. Having this problem corrected with declaring Exception in methods signature with throws

keyword in method signature, try block would throw an ArrayIndexOutOfBounds error, then the catch block handling that exception since it extends RuntimeException, printing, "Awake!", followed by finally block executing, throwing Exception in the process hence printing a stack trace. In short, the correct answer would be A.