CP 215 Lab 8: Java Exception Handling Study Notes

Source used: **CP 215 Lab8 (1).pdf** + code in **QuestionOne** to **QuestionSeven** folders. These notes focus on what each question demonstrates in Java exception handling.

1) Catching Subclass Exceptions with a Superclass

Catch

What the question asks

Create `ExceptionA`, `ExceptionB extends ExceptionA`, and `ExceptionC extends ExceptionB`. Show that `catch(ExceptionA e)` catches both `ExceptionB` and `ExceptionC`.

What your code demonstrates

- Polymorphism in exception handling: a superclass catch handles all its subclass exception objects.
- You can reduce repeated catch blocks when exceptions are related by inheritance.

Sample

```
try {
```

```
    throw new ExceptionC("This is ExceptionC.");

} catch (ExceptionA e) {

    System.out.println(e.getMessage());

}

// Output: This is ExceptionC.
```

Exam point

If exception type `X` extends `Y`, then `catch(Y e)` can catch `X`.

2) Catching Different Exceptions with `catch (Exception e)`

What the question asks

Throw `ExceptionA`, `ExceptionB`, `NullPointerException`, and `IOException`, then catch all with `Exception`.

What your code demonstrates

- `Exception` is a common parent for many checked and unchecked exceptions.

•A general catch works, but specific catches are better when handling logic differs.

Sample

```
try {

    throw new IOException("IOException was
thrown.");

} catch (Exception e) {

    System.out.println("Caught with Exception: " +
e);

}
```

Exam point

`catch(Exception e)` is broad; use it carefully because it can hide type-specific handling.

3) Order of Catch Blocks

What the question asks

Show that catch block order matters: superclass catch before subclass catch causes compile-time error.

What your code demonstrates

•Correct order: subclass first, superclass later.

- In your code, `catch(NumberFormatException e)` appears before `catch(Exception e)`.

Sample

```
try {

    Integer.parseInt("CP215"); //
NumberFormatException

} catch (NumberFormatException e) {

    System.out.println("Specific catch first");
```

```
} catch (Exception e) {

    System.out.println("General catch second");

}
```

Exam point

Write catch blocks from most specific to most general to avoid unreachable-code compile errors.

4) Constructor Failure

What the question asks

Show a constructor throwing an exception and passing failure info to an exception handler.

What your code demonstrates

- Constructors can throw exceptions using `throws`.
- Object creation with `new` should be in a try block when constructor may fail.

Sample

```
class SomeClass {

    SomeClass() throws Exception {

        throw new Exception("Constructor
failed.");

    }

}


try {

    new SomeClass();

} catch (Exception e) {

    System.out.println(e.getMessage());

}
```

Exam point

Construction is not guaranteed to succeed; constructor exceptions are handled like method exceptions.

5) Rethrowing Exceptions

What the question asks

`someMethod2()` throws, `someMethod()` catches and rethrows, `main()` catches and prints stack trace.

What your code demonstrates

- An exception can be caught for logging and rethrown for higher-level handling.
- The stack trace still shows where the exception originated.

Sample

```
static void someMethod() throws Exception {

    try {
```

```
            someMethod2();

        } catch (Exception e) {

            throw e; // rethrow same exception

        }

    }
```

Exam point

Use rethrowing when a lower method cannot fully recover but should report context upward.

6) Exceptions Can Escape to Outer Scope

What the question asks

Show that a method with its own try block does not need to catch every possible exception.

What your code demonstrates

- Inner method catches only `NullPointerException`.

- `ArithmeticException` from division by zero is not caught there and propagates to `main`.

Sample

```
void methodWithOwnTry() {

    try {

        int x = 10 / 0; // ArithmeticException

    } catch (NullPointerException e) {

        // does not handle ArithmeticException

    }

}



// Outer scope catches ArithmeticException.
```

Exam point

Exception handling can be split across scopes; unhandled exceptions move up the call stack.

7) Exception Propagation

What the question asks

`propagator1()` throws `ArithmeticException`, `propagator2()` catches and rethrows, `main()` handles and prints stack trace.

What your code demonstrates

•Runtime exceptions propagate automatically if not handled.

•Intermediate methods may catch, log, and rethrow without swallowing the error.

Sample

```
static void propagator2() {

    try {

        propagator1();

    } catch (ArithmeticException e) {
```

```
        throw e;

    }

}
```

Exam point

Stack traces are key for debugging propagation paths across multiple method calls.

Quick Revision Summary

•Superclass catches subclass exceptions.

- General `catch(Exception e)` catches many types but can be too broad.

•Catch order must be specific to general.

•Constructors can throw exceptions.

•Rethrowing preserves flow to higher-level handlers.

•Unhandled inner-scope exceptions move to outer scopes.

- Propagation + stack trace explain where failure started and how it moved.