Exercise 6: Object-Oriented Programming and Exception Handling

# Exercise 6.1: Object-Oriented Programming

1. In this Java program, inheritance is demonstrated by using extends from class C1 to C2. This allows C2 to use the method m defined in C1 to and alter it to have a different function.

class A1 {}

class B1 {}

class A2 {}

class B2 {}

class C1 {

public B1 m(A1 a) {

return new B1();

}

}

class C2 extends C1 {

@Override

public B2 m(A2 a) {

return new B2();

}

}

1. Covariance:
   1. Java, C++, C#
   2. Wikipedia
   3. Covariance allows a type to be substituted with its subtype, preserving the ordering of types (e.g., List<Cat> can be used where List<Animal> is expected)
   4. Advantage: The specific type can be helpful to reuse code.
2. Contravariance:
   1. Java, C++, Python
   2. Wikipedia
   3. Contravariance allows a type to be substituted with its supertype, reversing the ordering of types (e.g., Action<Animal> can be used where Action<Cat> is expected)
   4. Advantage: The allowance of a more general type can let the program have more flexible handling.

# Exercise 6.2: Exception Handling

Which parts of the try statement are executed, in what order, and what happens immediately after?

1. The try block completes normally, and the finally block completes normally.

public class FinallyExample {

public static void main(String[] args) {

System.out.print("1");

try {

throw new Exception();

} catch (Exception e) {

System.out.print("2");

} finally {

System.out.print("3");

}

}

}

// Output: 123

The FinallyExample function starts with printing out “1, “ and then immediately goes into a try statement that runs a new Exception(). This then skips to the next catch statement and runs the code within. In this case, it prints “2, “. From there, it continues to the finally statement and prints “3”.

1. The try block stops by throwing an exception that is caught, the relevant catch block completes normally, and the finally block stops by throwing an exception.

public class FinallyExample {

public static void main(String[] args) {

try {

System.out.print("Try ");

throw new RuntimeException("Try Exception");

} catch (Exception e) {

System.out.print("Catch ");

throw new RuntimeException("Catch Exception");

} finally {

System.out.print("Finally");

throw new RuntimeException("Finally Exception");

}

}

}

// Try Catch FinallyException in thread "main" java.lang.RuntimeException: Finally Exception at FinallyExample.main(FinallyExample.java:11)

The Try and Catch statements are executed as normally. The “3” is printed in the finally statement but then the runtime exception is flagged and the program errors. Even though the Try and Catch have runtime exception throws, only the Finally statement is printed because it is the last runtime exception and thus overrides the previous throws.

1. The try block stops by throwing an exception that is caught, the relevant catch block stops by executing an explicit return statement, and the finally block stops by executing an explicit return statement, returning a different value.

public class FinallyExample {

public static void main(String[] args) {

System.out.println(" = " + testMethod());

}

public static int testMethod() {

try {

System.out.print("Try ");

throw new RuntimeException("Try Exception");

} catch (Exception e) {

System.out.print("Catch ");

return 1; // Explicit return statement in catch block

} finally {

System.out.print("Finally");

return 2; // Explicit return statement in finally block

}

}

}

// Try Catch Finally = 2

The program seemed to run normally, The finally statement will always run after a try catch and so its lines will oftentimes override previous versions. In this case, the return from catch was overridden by finally and returned 2. This is a similar interaction as the previous example.