This program does not compile as is. Why?

Explain the execution of each method in writing. You have to name the file 27_3.txt

Question 1:

```
private int noSystemExit()      {
        try {
            throw new Exception("1");
        } catch (Exception e)     {
            System.out.println("2");
            return 0;
        } finally     {
            System.out.println("3 finally");
            return 1;
        }
    //        return 3; // unreachable statement
    }
```

Answer 1:

1. noSystemExit() will not compile because "return 3" is an unreachable statement and finally block is always executed in any situations unless seeing System.exit() and it has a return statement. So as for return 0 in catch block even if there is no return statement in the finally block. This will cause return 3 outsider the try-catch block unreachable.

2. Execuation Order: throw new Exception("1") -> System.out.println("2"); -> return 0; -> System.out.println("3 finally"); -> return 1;

Question 2:

```
private void withSystemExit()      {
        try {
            throw new Exception("4");
        } catch (Exception e)     {
            System.out.println("5");
            System.exit(0);
        } finally     {
            System.out.println("6 finally");
        }
        System.out.println("exit(): you will not see this line");     // not exectuted
    }
```

Answer 2:

1. withSystemExit() will compille without the question.

2. Execuation Order: throw new Exception("4") -> System.out.println("5"); -> System.exit(0);

Reason: a return statement cannot prevent a finally block from execuating, except for System.exit(), which will terminate the JVM, so the finally block will not be execuated.

Question 3:

```
private int anExeption1()      {
        int[] anArray  = new int[1];
        try {
            anArray[2] = 1 / 0;
            System.out.println("inside try: 1");
            return 0;
        } catch (ArithmeticException e)     {
            anArray[2] = 0;
            System.out.println("inside catch: 2");
            return 1;
        } finally     {
            System.out.println("inside finally: 3 ");
            return 2;
        }
        // return 3;
    }
```

Answer 3:

1. anExeption1 will compille without the question.

2. Execuation Order: int[] anArray = new int[1]; -> anArray[2] = 1 / 0; -> System.out.println("inside finally: 3 "); -> return 2;

Reason: anArray[2] = 1 / 0; will throw two exception ArithmeticException caused by 1 / 0, and ArrayIndexOutOfBoundsException cause by anArray[2]. Since Java execute assign statement from right to left, so 1 / 0 will be executed first and throw an exception which will be caught by the catch block. And in the catch block, anArray[2] = 0; will throw another exception. So the following statment will not executed but the finally block will be executed anway.

Question 4:

```
private int anExeption2()      {
        int[] anArray  = new int[1];
        try {
            anArray[2] = 0;
            anArray[2] = 1 / 0;
            System.out.println("inside try: 1");
            return 0;
        } catch (ArithmeticException e)     {
            System.out.println("inside catch: 2");
            return 1;
        } finally     {
            System.out.println("inside finally: 3 ");
            return 2;
        }
        // return 3;
    }
```

Answer 4:

1.anExeption2() will compille as long as return 3 is commented out;

2. Execuation Order: int[] anArray = new int[1]; -> anArray[2] = 0; -> System.out.println("inside finally: 3 "); -> return 2;

Reason: The same reason as Question 3, and anArray[2] = 1 / 0; will throw an exception since 1 / 0. Consequently, the return value is 2;

Question 5:

```
private int noExeption()      {
       try {
          int x = 1 - 1;
          System.out.println("inside try: 1");
          return x;
       } catch (Exception e)     {
          System.out.println("inside catch: 2");
          return 1;
       } finally     {
          System.out.println("inside finally: 3 ");
          return 2;
       }
   }
```

Answer 5:

1. noExeption() will compille without the question.

2. Execuation Order: int x = 1 - 1; -> System.out.println("inside try: 1"); -> return x; -> System.out.println("inside finally: 3 "); -> return 2;

Reason: No exception thrown in the try block, so the catch block will not be executed, but the finally block will be always execuated anyway.