Given:
```
public class Batman {
   int squares = 81;  //instance variable :: heap area -> squares = 81 = 82

   public static void main(String[] args) {
      new Batman().go();
   }
   void go() {
     incr(++squares);
     System.out.println(squares);//82
   }
   void incr(int squares) { squares += 10; } //local variable :: stack area ->
squares = 82 = 92
}
```

What is the result?
A. 81
B. 82
C. 91
D. 92
E. Compilation fails.
F. An exception is thrown at runtime.

Answer: B

Given:
```
public class Yippee {
public static void main(String [] args) {
      for(int x = 1; x < args.length; x++) {
            System.out.print(args[x] + " ");
      }
  }
}
```
and two separate command line invocations:
java Yippee
java Yippee 1 2 3 4 ========> String[] args = {"1","2","3","4"}
What is the result?
A. No output is produced.
   1 2 3
B. No output is produced.
   2 3 4
C. No output is produced.
   1 2 3 4
D. An exception is thrown at runtime.
   1 2 3
E. An exception is thrown at runtime.
   2 3 4
F. An exception is thrown at runtime.
   1 2 3 4

Answer: B

Given:
```
public void go() {
   String o = "";
   z:
```

```
     for(int x = 0; x < 3; x++) {
      for(int y = 0; y < 2; y++) {
            if(x==1)break;
            if(x==2 && y==1) break z;
            o = o + x + y;
      }
     }
      System.out.println(o);
}
```
What is the result when the go() method is invoked?
A. 00
B. 0001
C. 000120
D. 00012021
E. Compilation fails.
F. An exception is thrown at runtime.

Answer:: C

```
1. public class A{
2.
3. private int counter = 0;
4.
5. public static int getInstanceCount() {
6.    return counter;
7. }
8.
9.  public A() {
10.    counter++;
11. }
12.
13.}
```
A. Compilation of class A fails.
B. Line 28 prints the value 3 to System.out.
C. Line 28 prints the value 1 to System.out.
D. A runtime error occurs when line 25 executes.
E. Compilation fails because of an error on line 28.

Answer: A

Given:
```
 public class Barn {
   public static void main(String[] args) {
     new Barn().go("hi", 1);
     new Barn().go("hi", "world", 2);
  }
  public void go(String... y, int x) {
     System.out.print(y[y.length - 1] + " ");
   }
 }
```
What is the result?
A. hi hi
B. hi world
C. world world
D. Compilation fails.
E. An exception is thrown at runtime.

Answer: D

Given:

```
public static void main(String[] args) {
      try {
            args = null;
            args[0] = "test";
            System.out.println(args[0]);
      }catch (Exception ex) {
            System.out.println("Exception");
      }catch (NullPointerException npe) {
            System.out.println("NullPointerException");
      }
}
```

What is the result?

A. test
B. Exception
C. Compilation fails.
D. NullPointerException

Answer:: C

Q>

Given:

```
1. public class Breaker2 {
2.    static String o = "";
3.
4.    public static void main(String[] args) {
5.          z: for (int x = 2; x < 7; x++) {
6.                if (x == 3)
7.                      continue;
8.                if (x == 5)
9.                      break z;
10.               o = o + x;
11.    }
12.         System.out.println(o);
13.     }
14.}
15.
```

What is the result?

A. 2
B. 24
C. 234
D. 246
E. 2346
F. Compilation fails

Q>

Given:

```
1. public class Venus {
2.          public static void main(String[] args) {
3.                      int[] x = { 1, 2, 3 };
4.                      int y[] = { 4, 5, 6 };
5.                      new Venus().go(x, y);
6.          }
7.
8.          void go(int[]... z) {
```

```
9.                for (int[] a : z)
10.                     System.out.print(a[0]);
11.         }
12.}
```

What is the result?
A. 1
B. 12
C. 14
D. 123
E. Compilation fails.
F. An exception is thrown at runtime.

Q>
Given:

```
1. public class Target {
2.    private int i = 0;
3.    public int addOne() {
4.          return ++i;
5.    }
6. }
```

And:

```
1. public class Client {
2.    public static void main(String[] args){
3.          System.out.println(new Target().addOne());
4.    }
5. }
```

Which change can you make to Target without affecting Client?
A. Line 4 of class Target can be changed to return i++;
B. Line 2 of class Target can be changed to private int i = 1;
C. Line 3 of class Target can be changed to private int addOne(){
D. Line 2 of class Target can be changed to private Integer i = 0;

Q>Choose the best option based on this program:

```
class Shape {
            public Shape() {
                  System.out.println("Shape constructor");
            }
            public class Color {
                  public Color() {
                        System.out.println("Color constructor");
                  }
            }
}
class TestColor {
      public static void main(String []args) {
            Shape.Color black = new Shape().Color(); // #1
      }
}
```

A. Compiler error: the method Color() is undefined for the type Shape
B. Compiler error: invalid inner class
C. Works fine: Shape constructor, Color constructor
D. Works fine: Color constructor, Shape constructor

Q>

```
class Vehicle
{
      int x;
      Vehicle(){
```

```
            this(10);
      }
      Vehicle(int x){
            this.x = x;
      }
}
class Car extends Vehicle
{
      int y;
      Car(){
            super();
            this(20);
      }
      Car(int y){
            this.y= y;
      }
      public String toString(){
            return super.x + " " + super.y;
      }
}
public class Test {
      public static void main(String[] args) {
            Vehicle y= new Car();
            System.out.println(y);
      }
}
```

Predict the answer
A. 10:20
B. 0:20
C. Compilation fails at line n1
D. Compilation fails at line n2

Control flow in try catch finally:
==========

```
try{
      statement-1
      statement-2
      statement-3
}catch(Exception e){
      statement-4
}finally{
      statement-5
}
        statement-6
```

Case1: If there is no exception.
           Output:: 1,2,3,5,6 :: normal termination

Case2: If an exception is raised at statement-2 and corresponding catch block is matched
         Output:: 1,4,5,6 :: normal termination

Case3: If an exception is raised at statement2 and corresponding catch block is not matched
         Output:: 1,5 :: abnormal termination

Case4: If an exception is raised at statement4

Output:: Abnormal termination by executing finally block

Case5: If an exception is raised at statement5
Output:: Abnormal termination by executing finally block

Control flow in Nested try-catch-finally
==========

```
try
{
      stmt-1
      stmt-2
      stmt-3

      try{
          stmt-4;
          stmt-5;
          stmt-6;
      }catch(X e){
            stmt-7;
      }finally{
          stmt-8;
      }
      stmt-9;
}
catch(Y e)
{
      stmt-10;
}
finally
{
      stmt-11;
}
    stmt-12;
```

Case1: If there is no exception :: 1,2,3,4,5,6,8,9,11,12

Case2: If an exception is raised at statement2 and corresponding catch block is matched
1,10,11,12 :: normal termination

Case3: If an exception is raised at statement2 and corresponding catch block is not matched.
1,11 :: abnormal termination

Case4: If an exception is raised at statement5 and corresponding inner catch block is matched.
1,2,3,4,7,8,9,11,12 :: normal termination

Case5: If an exception is raised at statement5 and inner catch has not matched but outer catch block is matched.
1,2,3,4,8,10,11,12 :: normal termination

Case6: If an exception is raised at statement5 and both inner catch and outer catch block is not matched.
1,2,3,4,8,11 :: abnormal termination.

Case11: If an exception is raised at statement 9 and corresponding catch block is matched.
Output:: 1,2,3,4,5,6,8,10,11,12 :: normal termination

Case12: If an exception is raised at statement 9 and corresponding catch block is not matched.
Output:: 1,2,3,4,5,6,8,11 :: abnormal termination

Case13: If an exception is raised at statement 10.
Output:: finally block executed with abnormal termination

Case14: If an exception is raised at statement 11 or 12.
Output:: abnormal termination.

Case7: If an exception is raised at statement7 and corresponding catch block is matched.
Output:: 1,2,3,...,8,10,11,12 -> normal termination

Case8: If an exception is raised at statement7 and corresponding catch block is not matched.
Output:: 1,2,3,4,.... 8,11 -> abnormal termination

Case9: If an exception is raised at statement8 and corresponding catch block is matched.
Output:: 1,2,3,4,5,6,...,10,11,12

Case10:If an exception is raised at statement8 and corresponding catch block is not matched.
Output:: 1,2,3,.....,11 -> abnormal termination.

Note:
1. If we are not entering into try block then finally block wont be executed.
2. If we are entering into try block without executing finally block we can't come out.
3. We can write try inside try, nested try-catch is possible.
4. Specific exceptions can be handled using inner try catch and generalized exceptions can be handled using outer try catch.

Note::
```
public class TestApp{
      public static void main(String... args){
            try{
                  System.out.println(10/0);
            }catch(ArithmeticException ae){
                  System.out.println(10/0);
            }finally{
                  String s=null;
                  System.out.println(s.length());
            }
      }
}
```
Default exception handler handles the most recent exception and it can handle only one exception.
RE: java.lang.NullPointerException.

Various possible cases of Exception
==========

1.try{

  }catch(X e){

  }
Output:: valid

2. try{

   }catch(X e){

   }catch(Y e){

   }
Output:: valid , X-> Child y-> Parent

3. try{

   }catch(X e){

   }catch(X e){

   }
Output:: invalid :: "Exception xxxx has already been caught"

4. try{

   }finally{

   }
Output:: valid case

5. try{

   }catch(X e){

   }finally{

   }

Output:: valid case

6. try{}

7. catch(){}

8. finally{}

      Output:: 6,7,8 invalid

9. try{}
   System.out.println("Hello");
   catch(){}
      Output:: invalid

10. try{}
    catch(X e){}
    System.out.println("hello");
    catch(Y e){}

```
  Output:: invalid
```

11.
```
try{}
catch(X e){}
System.out.println("hello");
finally{}
  Output:: invalid case
```

12.
```
try{}
finally{}
catch(X e){}
  Output:: invalid case
```

13.
```
try{}
catch(X e){}
try{}
finally{}
  Output:: valid case
```

14.
```
try{}
catch(X e){}
finally{}
finally{}
  Output:: invalid case
```

15.
```
try{}
catch(X e){
  try{}
     catch(Y e1){}
}
  Output:: valid case
```

16.
```
try{}
catch(X e){}
finally{
  try{}
    catch(Y e1){}
    finally{}

}
  Output:: Valid case
```

17.
```
try{
  try{}
}
  Output:: invalid case
```

18.
```
try
 System.out.println("hello");
catch(X e){}
  Output:: invalid case
```

19.
```
try{}
catch( X e1)
  System.out.println("hello");
  Output:: invalid case
```

20.
```
try{}
catch( NullPointerException e1){}
```

```
    finally
      System.out.println("Hello");
      Output:: invalid case

Rules::
1. Whenever we are writing try block compulsorily we should write either catch
block or finally
   try without catch and finally is invalid.
2. Whenever we are writing catch block, compulsorily try block is required.
3. Whenever we are writing finally block, compulsorily try block is required.
4. try catch and finally order is important.
5. With in try catch finally blocks, we can take try catch finally.
6. For try catch finally blocks curly braces are mandatory.

throw keyword in java
=====================
This keyword is used in java to throw the exception object manually and informing
jvm to handle the exception.
      Syntax:: throw new ArithmeticException("/ by zero");

Eg#1.
class Test{
      public static void main(String... args){
            System.out.println(10/0);
      }
}
Here the jvm will generate an Exception called "ArithmeticException", since main()
is not handling
it will handover the control to jvm, jvm will handover to DEH to dump the exception
object details
through printStackTrace().

                  vs
class Test{
      public static void main(String... args){
            throw new ArithmeticException("/by Zero");
      }
}
Here the programmer will generate ArithmeticException,and this exception object
will be delegated
to JVM, jvm will handover the control to DEH to dump the excpetion information
details through
printStackTrace().

Note:: throw keyword is mainly used to throw an customized exception not for
predefined exception.

eg::
  class Test{
      static ArithmeticException e =new ArithmeticException();
      public static void main(String... args){
            throw e;
      }
  }
Output::
   Exception in thread "main" java.lang.ArithmeticException

eg::
class Test{
```

```
      static ArithmeticException e;
      public static void main(String... args){
            throw e;
      }
  }

Output::
  Exception in thread "main" java.lang.NullPointerException.

Case2
=====
After throw statement we can't take any statement directly otherwise we will get
compile time
error saying unreachable statement.

eg#1.
class Test{
      public static void main(String... args){
            System.out.println(10/0);
            System.out.println("hello");
      }
}
Output::
Exception in thread "main" java.lang.ArithmeticException
                    VS
eg#2.
class Test{
  public static void main(String... args){
            throw new ArithmeticExeption("/ by zero");
            System.out.println("hello");
  }
}
Output::
 CompileTime error
  Unreachable statement
    System.out.println("hello");

Case3
=====
  We can use throw keyword only for Throwable types otherwise we will get compile
time error
  saying incompatible type.

eg#1.
class Test3{
      public static void main(String... args){
            throw new Test3();
      }
}
Output::
 Compile time error.
 found::Test3
 required:: java.lang.Throwable

eg#2.
public class Test3 extends RunTimeException{
      public static void main(String... args){
            throw new Test3();
      }
```

```
}
Output::
  RunTimeError: Exception in thread "main" Test3

throws statement
================
 In our program if there is a chance of raising checked exception then compulsory
we should handle
 either by try catch or by throws keyword otherwise the code won't compile.

eg#1.
import java.io.*;
class Test3{
      public static void main(String... args){
            PrintWriter pw=new PrintWriter("abc.txt");
            pw.println("Hello world");
      }
}
CE: unreported exception java.io.FileNotFoundException; must be caught or declared
to be thrown

eg#2.
class Test3{
      public static void main(String... args){
            Thread.sleep(3000);
      }
}
CE: unreported exception java.lang.InterruptedException; must be caught or declared
to be thrown

We can handle this compile time error by using the following 2 ways
1. using try catch
2. using throws keyword

1. using try catch
class Test3{
      public static void main(String... args){
            try{
                  Thread.sleep(5000);
            }catch(InterruptedException ie){}
      }
}
ouput:: compiles and succesfully running

2. using throws keyword
class Test{
      pubilc static void main(String... args) throws InterruptedException{
            Thread.sleep(5000);
      }
}
output:: compiles and succesfully running.
=> we can use throws keyword to delegate the responsibility of exception handling
to the caller
    method. Then caller method is responsible to handle the exception.

Note::
  . Hence the main objective of "throws" keyword is to delegate the responsiblity
of exception
    handling to the caller method.
```

```
  . throws keyword required only for checked exception. usage of throws keyword for
unchecked
    exception there is no use.
  . "throws" keyword required only to convince compiler.Usage of throws keyword
does not prevent
    abnormal termination of the program.

Hence recomended to use try-catch over throws keyword.

eg#1.
class Test{
      public static void main(String... args) throws InteruptedException{
            doWork();
      }
      pubilc static void doWork() throws InteruptedException{
            doMoreWork();
      }
      public static void doMoreWork() throws InteruptedException{
            Thread.sleep(5000);
      }
}

In the above code, if we remove any of the throws keyword it would result in
"CompileTimeError".

Case studies of Throwable
=========================
Case 1::
    we can use throws keyword only for Throwable types otherwise we will get
compile time error.

class Test3{
      public static void main(String... args)throws Test3{

      }
}
output:Compile Time Error,Test3 cannot be Throwable

class Test3 extends RuntimeException{
      public static void main(String... args)throws Test3{

      }
}
output:Compiles and run succesfully


Case2::
public class Test3 {
      public static void main(String... args) {
            throw new Exception();
      }
}
Output:Compile Time Error
    unreported Exception must be caught or declared to be thrown

public class Test3 {
      public static void main(String... args) {
            throw new Error();
      }
```

```
}
Output:RunTimeException
 Exception in thread "main" java.lang.Error
        at Test3.main(Test3.java:4)

Case3::
  In our program with in try block,if there is no chance of rising an exception
then we can't
  write catch block for that exception, otherwise we will get Compile Time Error
saying
   "exception XXX is never thrown in the body of corresponding try statement", but
this rule
  is applicable only for fully checked exceptions only.

eg#1.
 public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (Exception e)
            {

            }
      }
}
Output:hiee

eg#2.
public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (ArithmeticException e)
            {

            }
      }
}
Output:hiee

eg#3.
public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (java.io.FileNotFoundException e)
            {
```

```
            }
      }
}
Ouput::Compile time error(fully checked Exception)

eg#4.
public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (InteruptedException e)
            {

            }
      }
}
Ouput::Compile time error(fully checked Exception)
        exception InterruptedException is never thrown in the body of correspoding
try statement.

eg#5.
public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (Error e)
            {

            }
      }
}
Output:hiee
```