Exception Handling

1. Find Output

```
public class Test
{
public static void main(String[] args) {
        int a=0;
        float b= 1.35F;
        System.out.println(b/a);
    }
}
```

Output

a) Infinity

b) Compilation error

c) Exception

d) 1.35

Ans: a

2. Find Output

```
public class Test
{
public static void main(String[] args) {
        int a=0;
        float b= 0;
        System.out.println(b/a);
    }
}
```

Output

a) NaN

b) Compilation error

c) Exception

d) Infinity

Ans: a

3. Find Output

```
public class Test
{
public static void main(String[] args) {
        int a=0;
        double b= -2.67;
        System.out.println(b/a);
    }
}
```

Output

a) -Infinity

b) Compilation error

c) Exception

d) Infinity

Ans: a

4. Find Output

```
public class Test
{
   public static void main(String[] args)
   {
      try
      {
         System.out.printf("1");
         int sum = 9 / 0;
         System.out.printf("2");
      }
      catch(ArithmeticException e)
      {
         System.out.printf("3");
      }
      catch(Exception e)
      {
         System.out.printf("4");
      }
      finally
      {
         System.out.printf("5");
      }
   }
}
```

Output

- a) 1325
  b) 1345
  c) 1342
  d) 135
- **Ans.** (d)
  Explanation: Once an exception occurs in try block, the execution passes to **corresponding** catch statement and doesn't return back to try block. Only one of the catch blocks are executed at a time. finally block is always executed whether or not the exception occurred.

5. Find Output

```
public class Test
{
   private void m1()
   {
      m2();
      System.out.printf("1");
   }
   private void m2()
   {
      m3();
      System.out.printf("2");
   }
   private void m3()
   {
      System.out.printf("3");
      try
      {
         int sum = 4/0;
         System.out.printf("4");
      }
      catch(ArithmeticException e)
      {
         System.out.printf("5");
      }


      System.out.printf("7");
   }
   public static void main(String[] args)
   {
      Test obj = new Test();
      obj.m1();
   }
}
```

Output

a) 35721
b) 354721
c) 3521
d) 35
Ans. (a)
Explanation: If an exception is handled in the catch statement, the program continues with its normal execution, after executing the catch statement corresponding to that exception. Also, when an exception occurs in the try block, the rest of the program in the try block is not executed.

6. Find Output

```
public class Test
{
   public static void main(String[] args)
   {
      try
      {
         System.out.printf("1");
         int data = 5 / 0;
      }
      catch(ArithmeticException e)
      {
         System.out.printf("2");
         System.exit(0);
      }
      finally
      {
         System.out.printf("3");
      }
      System.out.printf("4");
   }
}
```

Output

- a) 12
  b) 1234
  c) 124
  d) 123
- **Ans.** (a)
  Explanation: The only case when the code inside finally block is not executed is when System.exit(0) is called explicitly in the program. Then exit statement is called and the program is terminated without executing any further.

7. Find Output

```
public class Test {
public static void main(String[] args)
{
try
{
System.out.printf("1");
int data = 5 / 0;
}
catch(ArithmeticException e)
{
System.out.printf("2");
return;
}
finally
{
System.out.printf("3");
}
System.out.printf("4");
}}
```

Output

- a) 12
  b) 1234
  c) 124
  d) 123
- **Ans.** d

8. Find Output

```
public class Test
{
   public static void main(String[] args)
   {
      try
      {
         System.out.printf("1");
         int data = 5 / 0;
      }
      catch(ArithmeticException e)
      {
         Throwable obj = new Throwable("Sample");
         try
         {
            throw obj;
         }
         catch (Throwable e1)
         {
            System.out.printf("8");
         }
      }
      finally
      {
         System.out.printf("3");
      }
      System.out.printf("4");
   }
}
```

Output

- a) Compilation error
  b) Runtime error
  c) 1834
  d) 134
- **Ans.** (c)
  Explanation: Exceptions can be thrown in catch clause. This is done in order to change the exception type at run time. Exceptions in catch clause are thrown by creating instances of class Throwable as shown in the program.

9. Find Output

```
import java.io.EOFException;
import java.io.IOException;

public class Test
{
   public static void main(String[] args)
   {
      try
      {
         System.out.printf("1");
         int value = 10 / 0;
         throw new IOException();
      }
      catch(EOFException e)
      {
         System.out.printf("2");
      }
      catch(ArithmeticException e)
      {
         System.out.printf("3");
      }
      catch(NullPointerException e)
      {
         System.out.printf("4");
      }
      catch(IOException e)
      {
         System.out.printf("5");
      }
      catch(Exception e)
      {
         System.out.printf("6");
      }
   }
}
```

Output

a) 1346
b) 136726
c) 136
d) 13
Ans. (d)
Explanation: In multi-catch statements, the exceptions must be listed from more specific to more general. Only one catch statement which is most specific to the occurred exception is executed.

10. Find Output

```
class Test
{
   public static void main(String[] args)
   {
      try
      {
         System.out.println(1/0);
      }
      catch(ArithmeticException e)
      {
         System.out.println(e.getMessage());
      }
   }
}
```

Output

1. java.lang.ArithmeticExcetion
2. / by zero
3. java.lang.ArithmeticExcetion:/ by zero
4. ArithmeticExcetion

The answer is option (2)

Explanation: In the above program, we are calling getMessage() method to print the exception information. We know that getMessage() method will always be printed as the description of the exception which is / by zero.

11. Find Output

```
class Test
{
   public static void main(String[] args)
   {
      try
      {
         System.out.println(1/0);
      }
      catch(ArithmeticException e)
      {
         System.out.println(e);
      }
   }
}
```

Output

1. java.lang.ArithmeticExcetion
2. / by zero
3. java.lang.ArithmeticExcetion:/ by zero
4. ArithmeticExcetion

The answer is option 3

12. Find Output

```
class Test
{
   public static void main(String[] args)
   {
      try
      {
         System.out.println(1/0);
      }
      catch(ArithmeticException e)
      {
         System.out.println("hi");
      }
      catch(Exception e)
      {
         System.out.println("Welcome");
      }
   }
}
```

Output

1. hi
2. No Output
3. Compile-time error
4. Welcome

Ans: 1

13. Find Output

```
class Test
{
   public static void main(String[] args)
   {
      try
      {
         System.out.println(1/0);
      }
      catch(Exception e)
      {
         System.out.println("Welcome");
      }
      catch(ArithmeticException e)
      {
         System.out.println("hi");
      }

   }
}
```

Output

1. hi
2. No Output
3. Compile-time error
4. Welcome

Ans: 3

14. Find Output

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println(1/0);
        }
    }
}
```

Output

1. Run-time Exception
2. Compile-time error
3. No Output
4. Infinity

- The answer is option (2)
- **Explanation:** In the above program, we are declaring a try block without any catch or finally block. We have to always declare try with catch or finally block because single try block is invalid. That’s Why it will give compile time error saying error: ‘try’ without ‘catch’, ‘finally’ or resource declarations.

15. Find Output

```
class Test
{
   public static void main(String[] args)
   {
      try
      {
         System.out.println(1/0);
      }
      System.out.println("Test");
      catch(ArithmeticException e)
      {
         System.out.println("Welcome");
      }
   }
}
```

Output

a) Compile-time error
b) Test
c) Welcome
d) Test Welcome

ans: a

Explanation: In the above program, we are declaring a try block and also a catch block but both are separated by a single line which will cause compile time error:

16. Find Output

```
class Main {
   public static void main(String args[]) {
      try {
         throw 10;
      }
      catch(int e) {
         System.out.println("Got the  Exception " + e);
      }
  }
}
```

Output

(A) Got the Exception 10
(B) Got the Exception 0
(C) Compiler Error
d) exception

Answer: (C)

Explanation: In Java only throwable objects (Throwable objects are instances of any subclass of the Throwable class) can be thrown as exception. So basic data type can no be thrown at all. ”throw 10” will raises compilation error.

17. Find Output

```
class Main {
   public static void main(String args[]) {
      try {
throw new ArithmeticException("10");
}
catch(ArithmeticException e) {
System.out.println(e);
}
  }
}
```

Output

(A) java.lang.ArithmeticException: 10

(B) Got the Exception 0

(C) Compiler Error

d) exception

Answer: (A)

Explanation: In Java only throwable objects (Throwable objects are instances of any subclass of the Throwable class) can be thrown as exception. So basic data type can no be thrown at all. ”throw 10” will raises compilation error.

18. Find Output

```
class Test extends Exception { }

class Main {
   public static void main(String args[]) {
      try {
         throw new Test();
      }
      catch(Test t) {
         System.out.print("Got the Test Exception  ");
      }
      finally {
         System.out.println("Inside finally block ");
      }
  }
}
```

Output

(A) Got the Test Exception Inside finally block
(B) Got the Test Exception
c) Inside finally block
(D) Compiler Error

Answer: (A)

Explanation: In Java, the finally is always executed after the try-catch block. This block can be used to do the common cleanup work.

19. Find Output

```
class Main {
   public static void main(String args[]) {
      int x = 0;
      int y = 10;
      int z = y/x;
  }
}
```

Output

(A) Compiler Error
(B) Compiles and runs fine
(C) Compiles fine but throws ArithmeticException exception

D) No output

Answer: (C)

Explanation: ArithmeticException is an unchecked exception, i.e., not checked by the compiler. So the program compiles fine

20. Find Output

```
class Base extends Exception {}
class Derived extends Base  {}

public class Main {
  public static void main(String args[]) {
   // some other stuff
   try {
      // Some monitored code
      throw new Derived();
    }
    catch(Base b)     {
      System.out.println("Caught base class exception");
    }
    catch(Derived d)  {
      System.out.println("Caught derived class exception");
    }
  }
}
```

Output

(A) Caught base class exception
(B) Caught derived class exception
(C) Compiler Error because derived is not throwable
(D) Compiler Error because base class exception is caught before derived class

Answer: (D)

21. Find Output

```
class Test
{
   public static void main (String[] args)
   {
      try
      {
         int a = 0;
         System.out.print ("a = " + a);
         int b = 20 / a;
         System.out.printl("b = " + b);
      }

      catch(ArithmeticException e)
      {
         System.out.print(“ Divide by zero error");
      }

      finally
      {
         System.out.println (“ inside the finally block");
      }
   }
}
```

Output

(A) Compile error
(B) Divide by zero error
(C) a = 0 Divide by zero error inside the finally block
(D) a = 0

Answer: (C)

Explanation: On division of 20 by 0, divide by zero exception occurs and control goes inside the catch block.
Also, the finally block is always executed whether an exception occurs or not.

22. Find Output

```
class Test
{
   public static void main(String[] args)
   {
      try
      {
         int a[]= {1, 2, 3, 4};
         for (int i = 1; i <= 4; i++)
         {
            System.out.println ("a[" + i + "]=" + a[i] + "\n");
         }
      }

      catch (Exception e)
      {
         System.out.println ("error = " + e);
      }

      catch (ArrayIndexOutOfBoundsException e)
      {
         System.out.println ("ArrayIndexOutOfBoundsException");
      }
   }
}
```

Output

(A) Compiler error
(B) Run time error
(C) ArrayIndexOutOfBoundsException
(D) Error Code is printed
Answer: (A)

Explanation: ArrayIndexOutOfBoundsException has been already caught by base class Exception. When a subclass exception is mentioned after base class exception, then error occurs.

23. Find Output

```
class Test {
   public static void main(String args[]) {
      try {
         System.out.print("Hi ");
         int c=12/0;
      }
      catch(ArithmeticException ae) {
         System.out.print("Test ");
      }
      finally {
         System.out.print(" welcome");
      }
  }
}
```

Output

a) Hi Test welcome
b) Compilation error
c) Exception
d) Hi Test

Ans: a

24. Find Output

```
class Test {
   public static void main(String args[]) {
      try {
         int a[] = new int[-5];
      }
      catch(ArithmeticException ae) {
         System.out.print("Test ");
      }
      finally {
         System.out.print("welcome ");
      }
   }
}
```

Output

a) Welcome exception message
b) Test welcome
c) Welcome
d) Test

Ans: a

25. Find Output

```
class Test {
   public static void main(String args[]) {
      try {
         String s= null;
      }
      finally {
         System.out.print("welcome ");
      }
      catch(ArithmeticException ae) {
          System.out.print("Test ");
  }
}
```

Output

a) Compilation error
b) Exception
c) Welcome Test
d) Test Welcome

Ans: a

26. Find Output

```
class exception_handling
{
public static void main(String args[])
{
try
{
System.out.print("Hello" + " " + 1 / 0);
}
catch(ArithmeticException e)
{
System.out.print("World");
}
}
}
```

Output

- a) Hello
  b) World
  c) HelloWorld
  d) Hello World

- Answer: b
- Explanation: System.ou.print() function first converts the whole parameters into a string and then prints, before “Hello” goes to output stream 1 / 0 error is encountered which is cached by catch block printing just “World”.

27. Find Output

```
class exception_handling
{
public static void main(String args[])
{
try
{
int a, b;
b = 0;
a = 5 / b;
System.out.print("A");
}
catch(ArithmeticException e)
{
System.out.print("B");
}
}
}
```

Output

- a) A
  b) B
  c) Compilation Error
  d) Runtime Error

- Answer: b

Find Output

Output