1. Consider below code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
        boolean flag = false;
        do {
            if(flag = !flag) { //Line n1
                System.out.print(1); //Line n2
                continue; //Line n3
            }
            System.out.print(2); //Line n4
        } while(flag); //Line n5
    }
}
```

What will be the result of compiling and executing Test class?
A. 1
B. 2
C. 12
D. 21
E. 212
F. 121
G. 112
H. 221
I. CompilationError

Output :: C

2. Consider below code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
        for(int x = 10, y = 11, z = 12; y > x && z > y; y++, z -= 2) {
            System.out.println(x + y + z);
        }
    }
}
```

What will be the result of compiling and executing Test class?
A. 33
B. 32
C. 34
D. 33
   32
E. CompilationError

Answer: A

3.
Consider below code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
        int i = 0;
        for(System.out.print(i++); i < 2; System.out.print(i++)) {
            System.out.print(i);
        }
    }
}
```

What will be the result of compiling and executing Test class?
A. 112
B. 012
C. 011
D. 12

E. 01
F. CompilationError

Answer: C

4.
Consider below code of Test.java file:
```
public class Test {
    public static void main(String[] args) {
        outer: for(int i = 0; i < 3; System.out.print(i)) {
                        i++;
            inner: for(int j = 0; j < 3; System.out.print(j)) {
                  if(i > ++j) {
                              break outer;
                        }
            }
        }
    }
}
```
What will be the result of compiling and executing Test class?
A. CompilationError
B. Program terminates succesfully but nothing printed on the console
C. Program terminates succesfully after printing 1 on to the console
D. Program terminates succesfully after printing 12 on to the console
E. Program terminates succesfully after printing 123 on to the console
F. Program terminates succesfully after printing 1231 on to the console
G. Program terminates succesfully after printing 121 on to the console
H. Program terminates succesfully after printing 0120 on to the console

Answer: F

5.
What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        char [][] arr = {
                {'A', 'B', 'C'},
                {'D', 'E', 'F'},
                {'G', 'H', 'I'}
        };

        for(int i = 0; i < arr.length; i++) {
            for(int j = 0; j < arr[i].length; j++) {
                System.out.print(arr[i][1]);
            }
            System.out.println();
        }
    }
}
```
```
A. ABC
   DEF
   GHI
B. BBB
   EEE
   HHH
C. AAA
   DDD
   GGG
```

```
D. CCC
   FFF
   III

Answer: B

6.
What will be the result of compiling and executing Test class?
public class Test {
    public static void main(String[] args) {
        String msg = "Hello";
        boolean [] flag = new boolean[1];
        if(flag[0]) {
            msg = "Welcome";
        }
        System.out.println(msg);
    }
}
A. Hello
B. Welcome
C. ArrayIndexOutOfBoundsException
D. NullPointerException

Answer: A


7.
What will be the result of compiling and executing Test class?
public class Test {
    public static void main(String[] args) {
        String msg = "Hello";
        Boolean [] flag = new Boolean[1];
        if(flag[0]) {
            msg = "Welcome";
        }
        System.out.println(msg);
    }
}
A. Hello
B. Welcome
C. ArrayIndexOutOfBoundsException
D. NullPointerException

Answer: D

Rules of Overriding w.r.t Exception Handling
++++++++++++++++++++++++++++++++++++++++++++
  CheckedException   :: Exceptions which are checked by the compiler for smooth
execution of the program at the runtime is called
                        "CheckedExceptions".
                        eg:: SQLException,IOException,.....

  UncheckedException :: Exceptions which are not checked by the compiler are called
"UnCheckedExceptions".
                   eg:: RuntimeException and its child class, Error and its Child
class are "UncheckedExceptions".
```

Rules of Overriding when exception is involved

While Overriding if the child class method throws any checked exception compulsorily the parent class method should throw the same checked exception or its parent otherwise we will get Compile Time Error.
      There are no restrictions on UncheckedException.

eg#1.

```
class Parent{
      public void methodOne();
}
class Child extends Parent{
      public void methodOne() throws Exception{}
}
error: methodOne() in Child cannot override methodOne() in Parent
       public void methodOne() throws Exception{}
       overridden method does not throw Exception
```

Rules w.r.t Overriding

1.

```
parent: public void methodOne() throws Exception{}
child : public void methodOne()
```

Output:: valid[parent throwing an Exception, Child need not throw any Exception]

2.

```
parent: public void methodOne(){}
child : public void methodOne() throws Exception{}
```

Output:: invalid[Child throwing CheckedException,Compulsorily parent should throw the SameCheckedException or its Parent]

3.

```
parent: public void methodOne()throws Exception{}
child : public void methodOne()throws Exception{}
```

Output:: valid[Child throwing CheckedException,Compulsorily parent should throw the SameCheckedException or its Parent]

4.

```
parent: public void methodOne()throws IOException{}
child : public void methodOne()throws IOException{}
```

Output:: valid[Child throwing CheckedException,Compulsorily parent should throw the SameCheckedException or its Parent]

5.

```
parent: public void methodOne()throws IOException{}
child : public void methodOne()throws FileNotFoundException,EOFException{}
```

Output:: Valid[Child throwing CheckedException,Compulsorily parent should throw the SameCheckedException or its Parent]

6.

```
parent: public void methodOne()throws IOException{}
```

```
child : public void methodOne()throws FileNotFoundException,InterruptedException{}
```

Output:: invalid[Child throwing CheckedException,Compulsorily parent should throw the SameCheckedException or its Parent]

7.
```
parent: public void methodOne()throws IOException{}
child : public void methodOne()throws FileNotFoundException,ArithmeticException{}
```

Output:: Valid[Child throwing CheckedException,Compulsorily parent should throw the SameCheckedException or its Parent]
The rule is not applicable for UnCheckedException.

8.
```
parent: public void methodOne()
child : public void methodOne()throws
ArithmeticException,NullPointerException,RuntimeException{}
```

Output:: Valid[Rule is not applicable for UncheckedException].

Constructor level
+++++++++++++++++
=> Constructor level Exceptions should be of Sametype otherwise the code won't compiler
=> Rule is applicable only for CheckedExceptions not for UnCheckedExceptions.
=> Rules are separate for a method and constructor, because constructor won't participate in "inheritance" so we can't override the constructor.

1.
```
class Parent
{
      Parent() throws java.io.FileNotFoundException{

      }
}
class Child extends Parent
{
      Child() throws java.io.FileNotFoundException{

      }
}
public class Test
{
      public static void main(String[] args)throws Exception{
            Parent  p = new Child();
            System.out.println(p);
      }
}
```
Output:: Compilation succesfully.

2.
```
class Parent
{
      Parent() throws ArithmeticException{

      }
}
class Child extends Parent
{
      Child() throws RuntimeException{

      }
}
public class Test
{
      public static void main(String[] args){
            Parent  p = new Child();
            System.out.println(p);
      }
}
Output:: Compilation succesfully.
```

```
Exception handling keywords summary
===================================
1. try     => maintain risky code
2. catch   => maintain handling code
3. finally => maintain cleanup code
4. throw   => To hanover the created exception object to JVM manually
5. throws  => To delegate the Exception object from called method to caller method.

Various compile time errors in ExceptionHandling
================================================
1. Exception XXX is already caught
2. Unreported Exception XXX must be caught or declared to be thrown.
3. Exception XXX is never thrown in the body of corresponding try statement.
4. try without catch,finally
5. catch without try
6. finally without try
7. incompatible types :found xxx required:Throwable
8. unreachable code.

Note:
  In RealTime project we work with "CustomExceptions" and we use "try with
resource" to avoid resourceleak.



MultiThreading in java
++++++++++++++++++++++
Syllabus
========
1.Introduction.
2. The ways to define, instantiate and start a new Thread.
    1. By extending Thread class
    2. By implementing Runnable interface
3. Thread class constructors
4. Thread priority
5. Getting and setting name of a Thread.
6. The methods to prevent(stop) Thread execution.
      1. yield()
      2. join()
      3. sleep()
7. Synchronization.
```

8. Inter Thread communication.
9. Deadlock
10. Daemon Threads.
11. Various Conclusion
    1. To stop a Thread
    2. Suspend & resume of a thread
    3. Thread group
    4. Green Thread
    5. Thread Local
12. Life cycle of a Thread

Task

1. Any program which is under execution is called "Task/Process".

What is MultiTasking?
Executing more than one process simulataneously is refered as "MultiTasking".
The main advantage of MultiTaksing from the perspective of Os is "Utilize the CPU time effectively".
The process of shifting the control from one program to another program is called "Context Switching" which is performed by "O.s".

MulitTasking

Executing several task simultaneously is the concept of multitasking.
There are 2 types of Multitasking.
a. Process based multitasking
b. Thread based multitasking.

Process based multitasking

Executing several tasks simultaneously where each task is a seperate independent process such type of multitasking is called
"process based multitasking".
eg:: typing a java pgm
listening to a song
downloading the file from internet

Process based multitasking is best suited at "os level".

Thread based multitasking

=>Executing several tasks simulatenously where each task is a seperate independent part of the same Program,
is called "Thread based MultiTasking".

Each independent part is called "Thread".
1. This type of multitasking is best suited at "Programatic level".
The main advantages of multitasking is to reduce the response time of the system and to improve the performance.

2. The main important application areas of multithreading are
    a. To implement mulitmedia graphics
    b. To develop web application servers
    c. To develop video games

3. Java provides inbuilt support to work with threads through API called Thread,Runnable,ThreadGroup,ThreadLocal,...

```
4. To work with multithreading, java developers will code only for 10% remaining
90% java API will take care..

What is thread?
 A. Seperate flow of execution is called "Thread".
      if there is only one flow then it is called "SingleThread" programming.
      For every thread there would be a seperate job.


B. In java we can define a thread in 2 ways
    a. implementing Runnable interface
    b. extending Thread class


eg#1.
class MyThread extends Thread
{
      @Override
      public void run()
      {
            for (int i=0;i<10 ;i++ )
            {
                  System.out.println("child thread");
            }
      }
}


public class Test
{
      //JVM ---> main thread
      public static void main(String[] args)throws Exception{

            //instantiation of a thread
            MyThread t = new MyThread();

            //starting a thread
                  t.start();

            //job of main thread
            for (int i=0;i<5 ; i++)
            {
                  System.out.println("main thread");
                  Thread.sleep(1000);
            }
      }
}

Output: Exact output can't be predicted
main thread
child thread
child thread
child thread
child thread
child thread
child thread
child thread
child thread
child thread
child thread
main thread
main thread
main thread
main thread
```