Exceptions

Q1. Errors in Java Program:

a) Compile – time errors: These are syntactical errors found in the code. Like a semicolon missing.
b) Runtime Error: These errors represent inefficiency of the computer system to execute a statement.
c) Logical Errors: Flaw in the logic of the program.

Q2. What happens if main() method is written without String args[]?

The code compiles but JVM cannot run it, as it cannot see the main method with String args[].

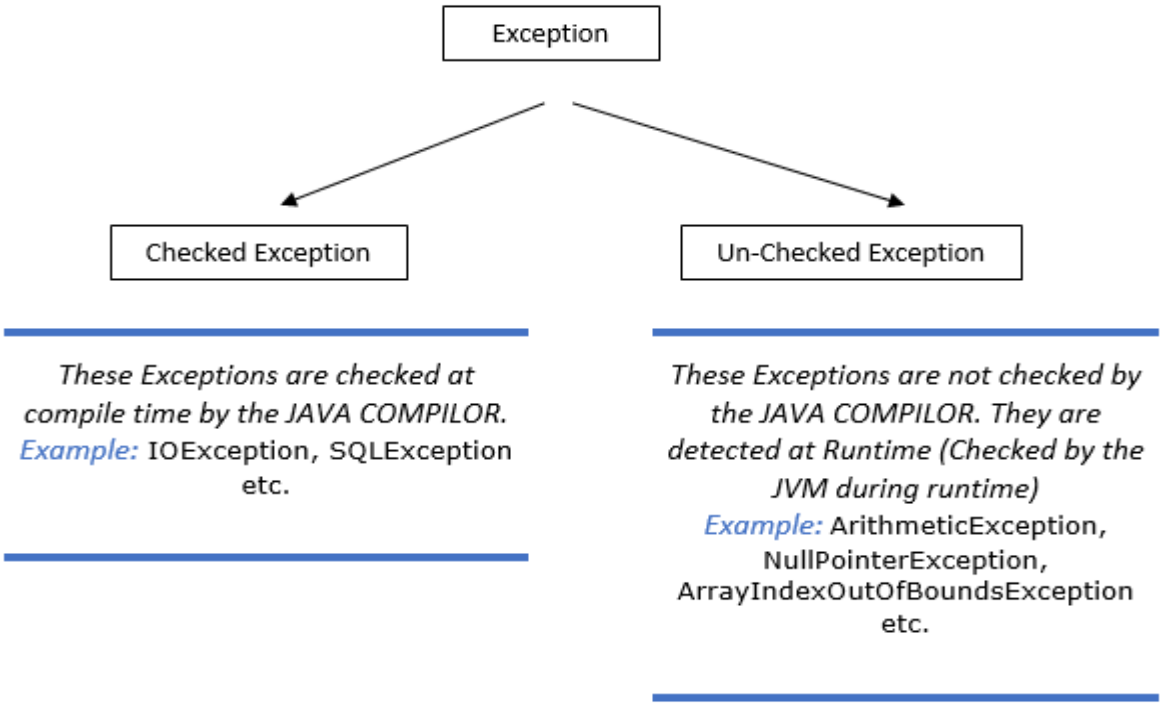
JVM-------→ Searches `public static void main(String[] args) {`

`}`

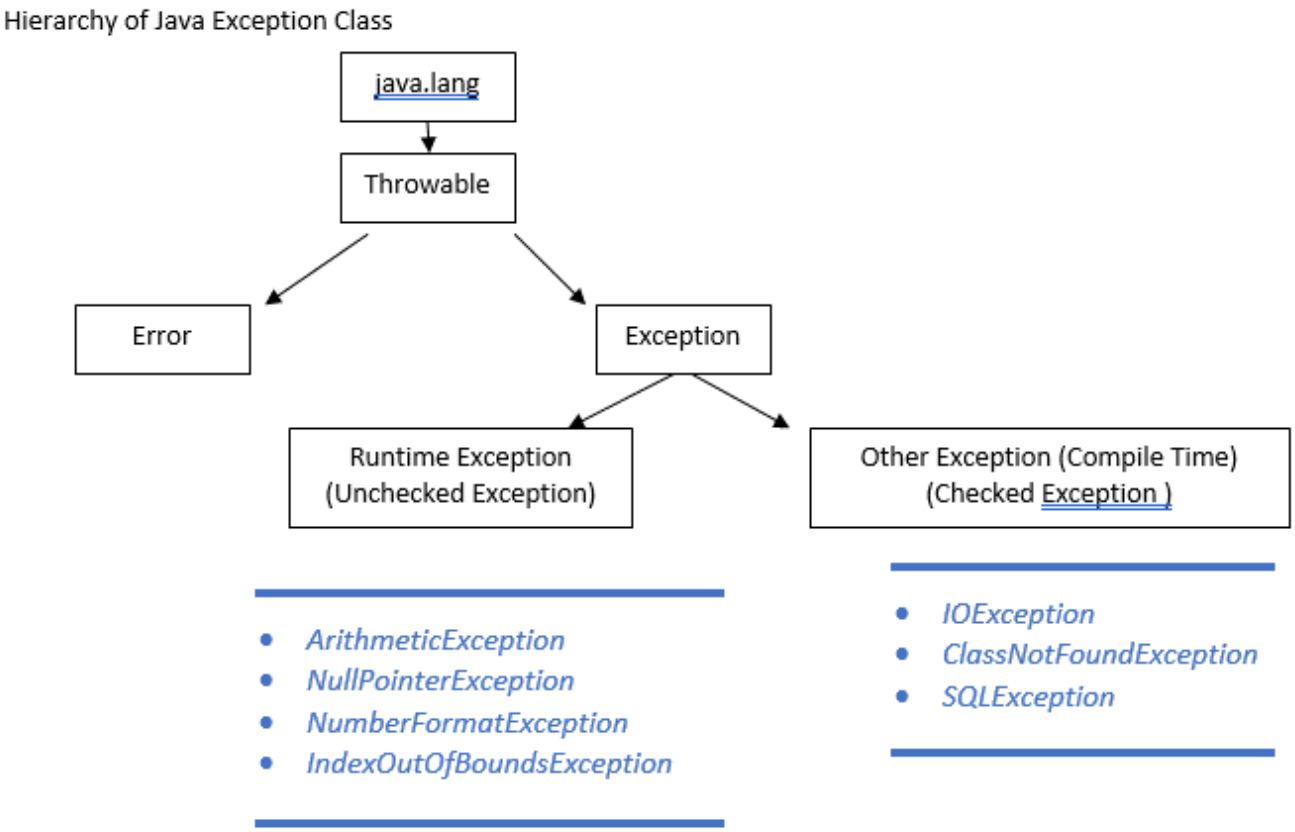
Q3. What is Exception? What is Exception handling in java? Types of Exception?

Exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Exception handling in java is one of the powerful mechanism to handle the runtime errors in the program so that the normal flow of the application can be maintained



Q4. Hierarchy of Java Exception Class



Q5. What is Throwable?

Throwable is a class that represents all Errors and Exceptions which may occur in Java

Q6. Which is the super class for all exceptions?

Exceptions is the superclass of all exceptions in java.

Q7. What is the difference between an exception and an error?

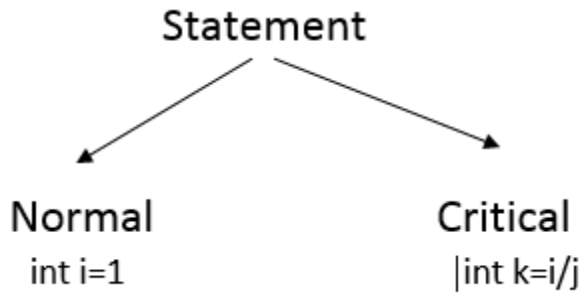
An Exception is an error which can be handled. It means when an exception happens, then programmer can do something to avoid any harm.

But an error is an error that cannot be handled, it happens, and the programmer cannot do anything.

Ex: OutOfMemoryError, VirtualMachineError, AssertionError etc.

Exception: We can handle that

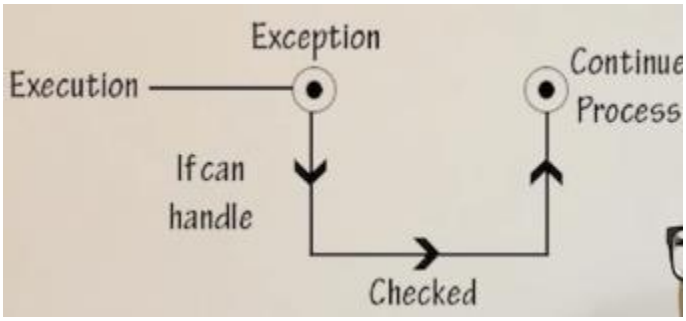
Error: We cannot handle that. Example, JVM not working properly or RAM out of Memory



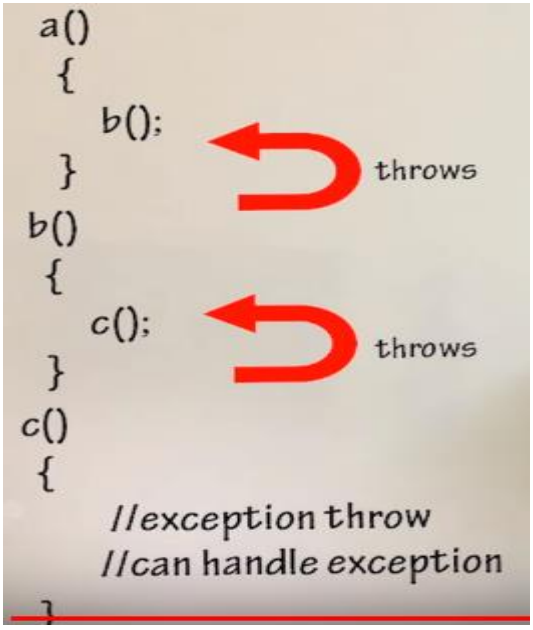
For Critical statement always use a try block, when exception comes within a try block it is catched by the catch block.

One try with multiple catch is possible.

```
try {
      int i=1/0   ;
} catch (ArithmeticException e) {
      // TODO: handle exception
} catch (Exception e) {
      // TODO: handle exception
}
```



If user can handle the Exception, then make it checked exception and if user cannot handle the Exception make it unchecked.



Example: In this program a calls b and b calls c.

If C is throwing an Exception we have actually two options.

- C can himself handle the Exception
- Or C can send the Exception to B.

Similarly, B have the two options.

C **throws** the Exception, throws means sending the request[Exception] above the Stack

Finally Block:

Java finally block is a block that is used to execute important code such as closing connection, stream etc. **Java finally block** is always executed whether exception is handled or not. **Java finally block** follows try or catch block.

```
try {
      int i=1/0   ;
}catch (ArithmeticException e) {
      System.out.println(e);
}catch (Exception e) {
      // TODO: handle exception
}finally {
      System.out.println("Bye Bye");
   }
```

Output:
```
java.lang.ArithmeticException: / by zero
Bye Bye
```

Try with Resources:

Normally, we se

```
/**
       * Try with Resource Java 1.7 onwards
       * @throws IOException
       */
      public static void main(String[] args) throws IOException {
            BufferedReader br=null;
            try {
                  String str="";
                  br=new BufferedReader(new InputStreamReader(System.in));
                  str=br.readLine();
            } catch (Exception e) {
                  System.out.println(e);
            }finally {
                  br.close();
            }
  }
```

But introducing the new concept,

```
/**
       * Try with Resource Java 1.7 onwards
       * @throws IOException
       */
      public static void main(String[] args) throws IOException {
            try(BufferedReader br=new BufferedReader(new InputStreamReader(System.in));)
            {
                  String str="";
                  str=br.readLine();
            }
      }
```

In this scenario we do not require any **catch block** or any **finally block.**
So, here we can write a try block without catch and finally.

Advantage is that, as soon as we are done with the try block the resource [In this case `BufferedReader br`] will get deallocated.
Even any exception occurs try will deallocate the resource.

Throws:

If you are lazy enough to not handle any exception.
Then at least **throws** the Exception to higher level of Stack.
throws is a keyword in java used to suppress Exception and not Handling i**t.**

Custom Exception:

```
try {
		int a=3;
		if(a<5)
			throw new ArithmeticException();
	}catch (ArithmeticException e) {
		System.out.println("Exception Occured "+"Minimum Value Of output is 5");
	}
	catch (Exception e) {
		// TODO: handle exception
  }
```

Here we are forcefully throwing an Exception by the **throw** keyword.

User defined Exception:

```
public class UserDefinedException {

	public static void main(String[] args) {
		try {
			int a=3;
			if(a<5)
				throw new MyException("Error Occured");
		}
		catch (Exception e) {
			System.out.println("Exception::"+e);
		}

	}

}

class MyException extends Exception{

	public MyException(String message) {
		super(message);// Call the Constructor of Exception Class
		// TODO Auto-generated constructor stub
	}

}
```

Q8. What is the difference between **throws** and **throw**?

Ans: **throws** clause is used when the programmer does not want to handle the exception and throw it out of a method. **throw** clause is used when a programmer wants to throw an exception explicitly and wants to handle it using catch block.

Q9. Is it possible to re-throw an exception?

Ans: Yes, we can re-through an exception from catch block to another class where it can be handled.

```
public class ReThrowExceptionDemo {

	public static void main(String[] args) {
		B b=new B();
		try {
			b.method1();
		} catch (StringIndexOutOfBoundsException e) {
			System.out.println("Exception Re-Occured :I caught the Re-thrown
Exception");
		}
	}
}

class B{
	public void method1() {
		try {
			String str="Hello";
			char ch=str.charAt(5);
		} catch (StringIndexOutOfBoundsException e1) {
			System.out.println("Exception Occured : Please check the index, it
should be within range");
			throw e1;  //Re-thrown the exception
		}
	}
   }
```

Output:

```
Exception Occured : Please check the index, it should be within range
Exception Re-Occured :I caught the Re-thrown Exception
```