EXPERIMENT NO. 7

1 .What is compile time error ? What are the possible reasons of compile time error?

Ans -

A compile time error is a problem such as a syntax error or missing file reference that prevents the program from successfully compiling .The compiler produces compile time errors and usually indicates what line of the source code is causing the problem.

Possible Reasons :

✓ At the time of compilation , if program contains any syntactical mistake then Compile time error will occur.

✓ If program contains external references and if compiler not found that references then compile time error will occur.

2.What is exception?

Ans –

An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled . An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

1.A user has entered an invalid data.
2.A file that needs to be opened cannot be found.
3.A network connection has been lost in the middle of communications or the JVM has run out of memory.
4.Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

3.What is need of exception handling?

Ans –

An exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

- A user has entered an invalid data.

- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Based on these, we have three categories of Exceptions. You need to understand them to know how exception handling works in Java.

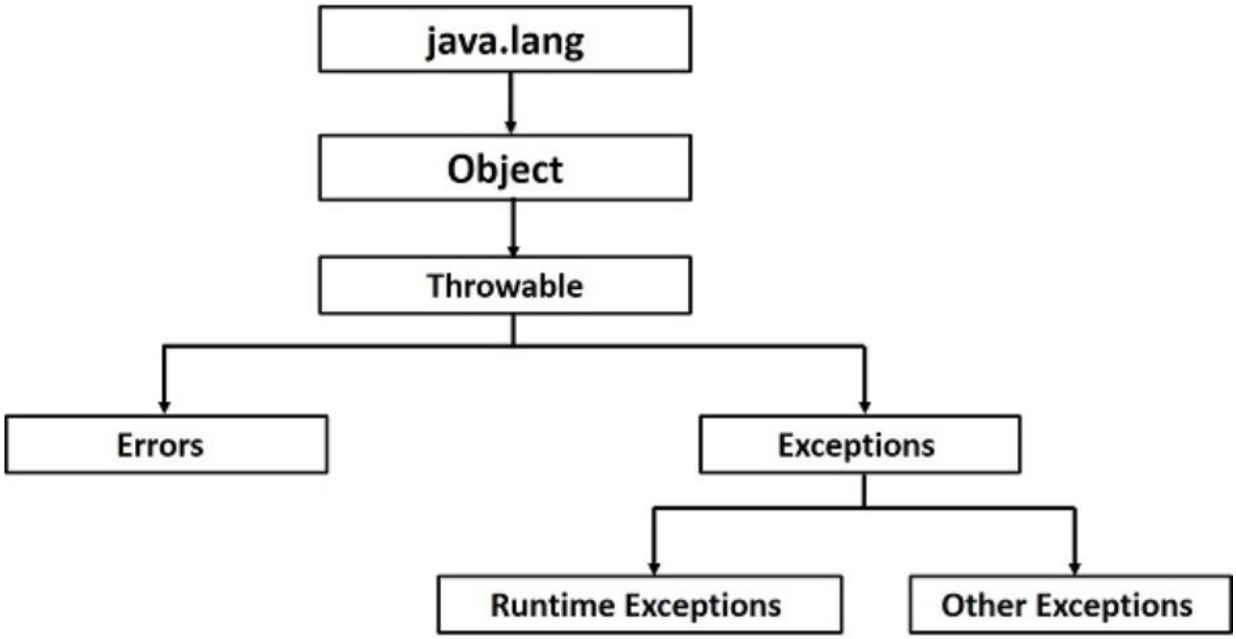

Diagram of Exception handling tree

4 . How to handle exception in java?

Ans –

Java exception handling is managed via five keywords: try, catch, throw, throws, and finally. Briefly, here is how they work. Program statements that you think can raise exceptions are contained within a try block. If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch block) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw. Any exception that is thrown out of a method must be specified as such by a throws clause.

5 .What is use of try , catch , throw ,throws and finally keyword?

Ans –

- **try block:**
  The code which might raises exception must be enclosed within try block try block must be followed by either catch block or finally If both present, it is still valid but the sequence of the try-catch-finally matters the most Otherwise, compile-time error will be thrown for invalid sequence The combination try-catch block or try-catch-finally blocks must reside inside method.
- **catch block:**
  Contains handling code for any exception raised from corresponding try block and it must be enclosed within catch block catch block takes one argument which should be of type Throwable or one of its sub-classes i.e.; class-name followed by a variable Variable contains exception information for exception raised from try block.
- **finally block:**
  finally block is used to perform clean-up activities or code clean-up like closing database connection & closing streams or file resources, etc .finally block is always associated with try-catch block With finally block, there can be 2 combinations One is try block is followed by finally block and other is try-catch-finally sequence The only other possible combination is try block followed by multiple catch block and one finally block at the end (this is case of multiple catch blocks)
- **throw clause:**
  Sometimes, programmer can also throw/raise exception explicitly at runtime on the basis of some business condition To raise such exception explicitly during program execution, we need to use throw keyword
  Syntax: throw instanceOfThrowableType;
  Generally, throw keyword is used to throw user-defined exception or custom exception
  Although, it is perfectly valid to throw pre-defined exception or already defined exception in Java like IOException, NullPointerException, ArithmeticException, InterruptedExcepting, ArrayIndexOutOfBoundsException, etc.
- **throws keyword or throws clause:**
  throws keyword is used to declare the exception that might raise during program execution whenever exception might thrown from program, then programmer doesn't necessarily need to handle that exception using try-catch block instead simply declare that exception using throws clause next to method signature But this forces or tells the caller method to handle that exception; but again caller can handle that exception using try-catch block or re-declare those exception with throws clause

  Note: use of throws clause doesn't necessarily mean that program will terminate normally rather it is the information to the caller to handle for normal termination

6. What is difference between checked and unchecked exception ?

Ans –

1) Checked:
are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.
For example, consider the following Java program that opens file at location "C:\test\a.txt" and prints the first three lines of it. The program doesn't compile, because the function main() uses FileReader() and FileReader() throws a checked exception FileNotFoundException. It also uses readLine() and close() methods, and these methods also throw checked exception IOException

```
import java.io.*;
class Main {
public static void main(String[] args) {
FileReader file = new FileReader("C:\\test\\a.txt");
BufferedReader fileInput = new BufferedReader(file);
// Print first 3 lines of file "C:\test\a.txt"
for (int counter = 0; counter < 3; counter++)
System.out.println(fileInput.readLine());
fileInput.close();                          }
}
```

2) Unchecked:
Unchecked are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions. In Java exceptions under Error and RuntimeException classes are unchecked exceptions, everything else under throwable is checked. Consider the following Java program. It compiles fine, but it throws ArithmeticException when run. The compiler allows it to compile, because ArithmeticException is an unchecked exception.

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

7 . What are standard runtime exceptions in java?

Ans –

Common Runtime Errors-

1.NullPointerException:

Thrown when Java encounters a null reference when it doesn't expect one .This usually happens when you try to use something that hasn't been initialized or instantiated, or something that no longer exists because it was garbage collected or fundamentally altered.

Null pointers will occur most often when you create an instance variable and try to use it before you've actually assigned it a value. See below for an example of this occurrence.

```
public class OperateCar(){
private Car _car;
public OperateCar(){ }
public void drive(){
_car.moveForward(1); //NullPointerException
occurs
}
}
```

The above code would be fixed by initializing _carin the constructor, which

would look like this: _car = new Car();

2.ClassCastException:

Happens when your code attempts to wrongly cast an object to a particularclass. Specifically, your object is not an instance of this class, or yourobject is not a subclass of this class. For example, if you try to cast an Car to a Color

Car myCar = new Car();

Color myColor = (Color) myCar; //generates ClassCastException

3.Arithmetic Exception:

Illegal arithmetic attempt. Most likely you tried to divide by zero.

4.StackOverflowException:

More methods got called than Java memory can handle. It is most likely you have an infinite loop or a problem with your base case in recursion, which you'll learn about a little later.

Note: Sometimes it may take a while before the StackOverflowException is actually thrown. If your program is taking a really long time to run, then you might be in the middle of an infinite loop.

```
Boolean alwaysTrue = true;
while(alwaysTrue)
{ //will run infinitely causing an
error
myCar.drive();
}
```

5.IndexOutOfBoundsException:

When Java tries to access an Object in a list at an index that is not in the list's range.

```
Car[] listOfCars = new Car[5]; //array with indexes 0 through 4

for(int i = 0; i <= 5; i++){

listOfCars[i].drive(); //when i = 5, error will be

thrown

}
```

8.Who handles exception in java?

Ans –

Default Exception Handling : Whenever inside a method, if an exception has occurred, the method creates an Object known as Exception Object and hands it off to the run-time system(JVM). The exception object contains name and description of the exception, and current state of the program where exception has occurred. Creating the Exception Object and handling it to the run-time system is called throwing an Exception.There might be the list of the methods that had been

called to get to the method where exception was occurred. This ordered list of the methods is called Call Stack.Now the following procedure will happen.

The run-time system searches the call stack to find the method that contains block of code that can handle the occurred exception. The block of the code is called Exception handler.

The run-time system starts searching from the method in which exception occurred, proceeds through call stack in the reverse order in which methods were called.

If it finds appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.

If run-time system searches all the methods on call stack and couldn't have found the appropriate handler then run-time system handover the Exception Object to default exception handler , which is part of run-time system. This handler prints the exception information in the following format and terminates program abnormally.