UNIT-IV
Exception Handling

Introduction:

There are three types of errors in java
1.Compile-time errors
2.Run time errors
3.logical errors

COMPILE TIME ERRORS:
These are errors, which prevents the code from compiling because of error in the syntax such as missing a semicolon at the end of a statement or due to missing braces, class not found, etc. These errors will be detected by java compiler and displays the error onto the screen while compiling.
Ex:
System.out.println(“ HELLO WORLD); // ” missing etc
RUN TIME ERROR:
These errors are errors which occur when the program is running. Run time errors are not detected by the java compiler. It is the JVM which detects it while the
program is running.
Ex:
int a[]=new int[5];
System.out.println(a[5]); //the 4 is the last index. It is expressed as an exception by the JVM.

LOGICAL ERRORS:
These errors are due to the mistakes made by the programmer. It will not be detected by a compiler nor by the JVM. Errors may be due to wrong idea or concept used by a programmer while coding or by giving wrong data given input by user.
Ex:
int a,b,c;
a=sc.nextInt();
b=sc.nextInt();
c=a/b; //here if the user gives input of 0 as b value, then the JRE could not execute code and raises error by throwing an exception
The logical errors and runtime errors when raised in program, the JRE could not execute the code and terminates abruptly the function in which it could not executed code, and this case is said that the JRE has thrown an Exception. The java
provides a software mechanism called Exception Handling to let the programmer to control program execution, such that it won’t terminate abruptly.

The exception handling is supported with keywords try-catch-finally, throw and throws.
try-catch-finally : this block is used to handle the code, such that when an exception is thrown, the JRE customizes the process to be done without terminating abruptly.
The main benefit of exception handling is to debug the program, such that the program can be executed as robust program.

Q. Give an introduction and benefits of exceptions

Classification of exceptions- Checked / Unchecked Exceptions:

The java exceptions are classified into two types.

- Checked exceptions / compile time exceptions
- Unchecked exceptions / runtime exceptions

Checked exceptions: The code in java that has hardware interactivity or interactivity of other runtime environments always must be handled to achieve more perfection (Robustness). To make sure programmer to use appropriate handler, these exceptions are declared as checked / compile time exceptions. Unless these are handled, the code will not be compiled.

Runtime Exceptions: the exceptions that are thrown by JRE during running of program, based on data given input buy user or the logic implemented in program are called runtime exceptions.

The java can handle more than 600 types of pre-defined exceptions and all these are derived from java.lang.Exception class.

The different pre-defined exceptions are:

Compile time exceptions / checked exceptions:-

- IOException : when any IO operation is performed
- FileNotFoundException: when file processing code is written
- SQLException: when database connectivity established
- InterruptedException: when the sleep() of thread class is used
- ClassNotFoundException: when a class is loaded / used into memory and the class is not existing in the path specified

Runtime exceptions / unchecked exceptions

- ArrayIndexOutOfBoundsException :when we mention the index >= length of array

StringIndexOutOfBoundsException : when we use the index >= length() of String
ArithmeticException :when division is done with ZERO
NumberFormatException : when a non-numeric value is tried to be parsed
NullPointerException : when a class reference is used prior its instantiation(allocation of memory)

Q. Write about exception handling in java
Q. explain different types of exceptions in java

Handling exceptions using try-catch-finally:

The java provides a software mechanism called Exception Handling to let the programmer to control program execution, such that it won't terminate abruptly. The exception handling is supported with keywords try-catch-finally, throw and throws.

try-catch-finally: this block is used to handle the code, such that when an exception is thrown, the JRE customizes the process to be done without terminating abruptly.
Syntax :

```
try{
code expected to throw exception
}
catch(TYPEException e){
Code that need to be executed when exception is thrown and handled
}
finally{
default block
}
```

Here the TYPEException in catch() is replaced with appropriate exception type.
We place the code that is expected to throw exception in try block. A try must have at least one catch() or finally. But a try can have any number of catches, each handling a specific type of exception.
Ex:

```
try{
int x=Integer.parseInt(sc.nextLine());
int y=Integer.parseInt(sc.nextLine());
int z=x/y;
```

```
System.out.println(z);
} catch(ArithmeticException ae){
System.out.println("don't enter zeros");
}Catch(NumberFormtException ne){
System.out.println("enter only numbers");
}
```

The finally is default block that executes in all the cases. Generally, the closing of streams, database connections etc is one with in finally block.

Q. Write about exception handling with try-catch-finally

Benefits of exception handling:

The when exception is thrown by code in a method, it can be handled either with a try-catch block or leave it by declaring the method as it throws specific exception.

When the method is declared with "throws", the exception is not handled. For handling the code must be tried and caught. In general the programmer displays corresponding messages to user, so that the input can be given correctly into program.

But if the exception is thrown NOT by the data given input and it is because of a logical error in code, then the root cause must be identified to repair / modify the code.

For example, if "ArrayIndexOutOfBoundsException" is thrown in a location, the root cause is not in that location, it is at the location where the array is declared. To identify the root cause, so that the program can be debugged easily, we can print the stack of the exception throws and trace out the location and make correction to the code. It is done by the method printStackrace() of the Exception class.

Ex;

```
try{
        Some code
}catch(Exception e){
        e.printStackTrace();
}
```

This displays the messages of the stack that stored the root cause of current exception, by which programmer can easily debug it.

Questions that can be framed:

Write about debugging with exception handling.

How the exception handling can be used for debugging of program,
What are the advantages of exception handling

---

usage of throw and throws keywords:

The “throw” is a keyword, used to throw an exception explicitly. The “throw” can throw a pre-defined exception or any derived class from Exception including user defined exceptions.
Ex:
throw new ArithmeticException();
The throw must be placed within a try block to handle it.

The “throws” is a keyword used along with method signature to specify that the method throws a specific exception and the programmer must provide proper handler. i.e. it is used to specify that the programmer must handle that exception with catch(), unless which the code will not be compiled.
Ex:
abc.java

```
class abc{
public int divideData( int x, int y) throws ArithmeticException{ //line 2
return x/y;
}
}
```

---

bca.java

```
class bca {
public static void main(String args[]){
abc a=new abc();
try{
System.out.println( a.divideData(5,0) ); //line 5
}catch(ArithmeticException e){
System.out.println(e);
}
}
}
```

If the line at line 5 is not placed under try block, then the program will not be compiled. Thus, the “throws” at “line 2” makes sure when the add() is called, it must be tried and caught. If the programmer doesn’t have anything to do in catch(), then he may declare that the calling method also “throws” the exception, without handling.

Ex:
bca.java

```
class bca {
public static void main(String args[]) throws ArithmeticException{
abc a=new abc();
System.out.println( a.divideData(5,0) );
}
}
```

Q. Write about throw and throws clauses in java

User defined exceptions:

We can also develop our own exceptions by extending the java.lang.Exception class.

Ex:
UnderAgeException.java

```
public class UnderAgeException extends Exception{
        public UnderAgeException(){
                super("you are under 18. Not eligible for voting");
        }
}
```

```
public class Voting{
        public static void main(String args[]){
                Scanner sc=new Scanner(System.in);
                int age=sc.nextInt();
                if(age<18)
                try{
                throw new UnderAgeException();
                }catch(UnderAgeException e){
                        System.out.println(e);
                }
                else
                        System.out.println("Eligible for voting");
        }
}
```

The keyword "throw" is used to throw an exception explicitly. As the user defined exception is derived from Exception class, it has become throwable and can be caught like a normal pre-defined exception/
The message that must be displayed is passed as argument to Exception class

constructor by calling super(), and as the toString() of Exception class returns the message as string, the Exception object can be displayed directly as System.out.println(e).

Questions that can be framed:
Write about user defined exceptions
Explain user defined exceptions with example
Explain how to throw a user defined exception

Threads

introduction, thread properties:

Thread is process. When the process needs to be divided into sub processes and executed under control of programmer, that process can be made as a thread. Generally the process management is done by operating system. When process is developed as thread, the thread control is managed by programmers. As the threads won't give burden on operating system, they are called light weight components.

A Thread can be created by extending Thread class or by implementing Runnable interface of java.lang package. He process to be placed as definition of the overridden run() method.

The threads can be either single tasking or multi tasking. That is, multiple threads make a single process done or multiple processes done. In single tasking, we override one run() method and in multi tasking, we override multiple run() methods of Runnable interface.

ThreadPriority: the Thread priority specifies how much time the thread is to occupy processor within given time. It never indicates sequence of execution. It can be specified between 1 to 10. 1 is the minimum priority and the 10 is the maximum priority. The maximum priority threads occupy the processor much time within given time. But irrespective of the priority, the execution of thread is based on the other resources using processor. So the threads are inaccurate.

Uses of Threads:
Threads allow a program to operate more efficiently by doing multiple things at the same time. Threads can be used to perform complicated tasks in the background without interrupting the main program.

Terminating the Thread :
A Thread can be started its execution by calling the method start() of Thread class and it can be terminated by making a call to stop() method of Thread class.

Q. explain what is thread and how to use it.
Q. explain how the threads can be created and executed.
Q. explain how the single tasking and multi tasking threads are managed.

Thread class & methods and Runnable interface:

a Thread can be created from Thread class or by implementing Runnable interface

```
class MyThread extends Thread{
        public void run(){
        }
}
```

Or

```
class MyThread implements Runnable{
        public void run(){
        }
}
```

The process that need to be executed as thread must be placed as definition of the run() method. i.e. we must override the run() method when we implement Runnable or extends Thread class.

Thread class:

Constructors:

Thread()

Thread(Thread instance)

Thread(String name)

Methods:

currentThread() : this is a static method that returns the current thread instance.

sleep() : suspends program execution for specified number of milliseconds.

setName(String) : gives a name to the thread

setPriority(int) : used to change priority of thread

run() : used to perform the action/process of the thread

start() : starts execution of thread, when start() called, the JVM calls run()

isAlive() : returns true, if the thread is alive

Q. Explain about Thread class and its methods.

---

controlling thread with sleep():

when the sleep() / wait() is called the JRE "InterruptedException" , when a wrong argument (parameter value) is passed to any method of thread class then the "IllegalArgumentException" is thrown, and when working with multiple threads, and the threads reach a dead-lock situation, then "IllegalMonitorStateException" is thrown.

Ex:

```
class MT1 implements Runnable{
	Thread t;
	MT1()
	{
		t=new Thread(this);
	}

	public void run()
	{
		System.out.println("in MT1 :");
		for(int i=0;i<5;i++)
			System.out.print("hi ");
	}
}

class MT2
{
	public static void main(String a[]) throws Exception
```

```
	{
		System.out.println("started main thread:"+Thread.currentThread());
		MT1 m=new MT1();
		m.t.start();
		Thread.sleep(1000);
		System.out.println("\nMT1 is completed");
		System.out.println("Main  is completed");
	}
}
```

Output :

```
D:\bca>java MT2
started main thread :Thread[main,5,main]
in MT1 :
hi hi hi hi hi
MT1 is completed
Main  is completed
```

** if the main thread is not suspended with its execution with sleep(), then the main() thread completes its execution and then the user defined thread. Which leads to incorrect execution.

Q. Explain how to control thread with sleep method.

---

dead lock situation:
When working with multiple threads, we don't prefer controlling threads with sleep() to let other threads to occupy the processor.

In this case, if one thread has not completed its execution and still occupying processor, and if other thread has completed its sleep time and tries to occupy the processor, then the processor won't respond to any of the threads and it gets hang and the JRE throws IllegalMonitorStateException. This case is called 'DEADLOCK'.

To overcome this problem of deadlock, we control threads with the methods wait() and notify() of Object class, instead of sleep() of Thread class.

But the methods called under multiple threads must be declared as 'synchronized', unless which they misbehave (execute wrongly)

Synchronization: when multiple threads try to occupy processor, i.e. when multiple threads try to do process on one shared resource, then the process goes

uncontrolled. In this case, we declare the methods that are to be called by threads as synchronized, such that only one thread access the resource at a time and the other thread is in blocked state, and when the first thread goes to blocked state, the other uses the resource.

We can either declare the methods with “synchronized” keyword, or we can write a synchronized block in current versions of java.

Q. explain dead lock

Inter thread communication with wait(), notify() and synchronization:

interrupting threads, synchronizing threads, inter thread communication :
Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other. Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of Object class:
wait() , notify() and notifyAll()
1) wait() method
The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed. The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.
public final void wait()throws InterruptedException
It waits until object is notified.

2) notify() method
The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.
3) notifyAll() method Wakes up all threads that are waiting on this object's monitor. all these methods are defined in java.lang.Object class, because they are related to lock and object has a lock. Let's see the important differences between wait and sleep methods.

wait()	sleep()

The wait() method releases the lock.	The sleep() method doesn't release the lock.
It is a method of Object class	It is a method of Thread class
It is the non-static method	It is the static method
It should be notified by notify() or notifyAll() methods	After the specified amount of time, sleep is completed.

Q. Write about inter Thread Communication

Q. Explain the methods used for smooth running of multi-threading

---