Java-unit-2

18 May 2025 00:09

Unit-2: Exception handling & MultiThreading

Q.1 What is an Exception? Explain the types of exceptions?

In Java, an exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
These exceptions can occur for various reasons, such as invalid user input, file not found, or division by zero. When an exception occurs, it is typically represented by an object of a subclass of the java.lang.Exception class.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Types of Java Exceptions

1. Checked Exception
2. Unchecked Exception
3. Error

1. Checked Exceptions

Checked exceptions are the exceptions that are checked at compile-time. This means that the compiler verifies that the code handles these exceptions either by catching them or declaring them in the method signature using the throws keyword. Examples of checked exceptions include:

IOException: An exception is thrown when an input/output operation fails, such as when reading from or writing to a file.
SQLException: It is thrown when an error occurs while accessing a database.
ParseException: Indicates a problem while parsing a string into another data type, such as parsing a date.
ClassNotFoundException: It is thrown when an application tries to load a class through its string name using methods like Class.forName(), but the class with the specified name cannot be found in the classpath.

2. Unchecked Exceptions (Runtime Exceptions)

Unchecked exceptions, also known as runtime exceptions, are not checked at compile-time. These exceptions usually occur due to programming errors, such as logic errors or incorrect assumptions in the code. They do not need to be declared in the method signature using the throws keyword, making it optional to handle them. Examples of unchecked exceptions include:
NullPointerException: It is thrown when trying to access or call a method on an object reference that is null.
ArrayIndexOutOfBoundsException: It occurs when we try to access an array element with an invalid index.
ArithmeticException: It is thrown when an arithmetic operation fails, such as division by zero.
IllegalArgumentException: It indicates that a method has been passed an illegal or inappropriate argument.

3. Errors

Errors represent exceptional conditions that are not expected to be caught under normal circumstances. They are typically caused by issues outside the control of the application, such as system failures or resource exhaustion. Errors are not meant to be caught or handled by application code.
Examples of errors include:
OutOfMemoryError: It occurs when the Java Virtual Machine (JVM) cannot allocate enough memory for the application.
StackOverflowError: It is thrown when the stack memory is exhausted due to excessive recursion.
NoClassDefFoundError: It indicates that the JVM cannot find the definition of a class that was available at compile-time.

==========

Q.2 How to handle exception? Explain the suitable example?

Java provides five keywords that are used to handle the exception. The following table describes each.

1. try
2. Catch
3. Finally
4. Throw
5. Throws

1. Try:
   The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone.
   The try block must be followed by either catch or finally.

2. Catch
   The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

2. Finally
   The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

3. Throw

   The "throw" keyword is used to throw an exception.

4. Throws
   The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

   Let's see an example of Java Exception Handling in which we are using a try-catch statement to handle the exception.

```
//Java Program to understand the use of exception handling in Java
public class Main{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

The try-catch Block

One of the primary mechanisms for handling exceptions in Java is the try-catch block. The try block contains the code that may throw an exception, and the catch block is used to handle the exception if it occurs

Handling Multiple Exceptions

You can handle multiple types of exceptions by providing multiple catch blocks, each catching a different type of exception.

```
try {
    // Code that may throw an exception
} catch (IOException e) {
    // Handle IOException
} catch (NumberFormatException e) {
    // Handle NumberFormatException
} catch (Exception e) {
    // Handle any other exceptions
}
```

The finally Block

In addition to try and catch, Java also provides a finally block, which allows you to execute cleanup code, such as closing resources, regardless of whether an exception occurs or not. The finally block is typically used to release resources that were acquired in the try block.

```
//Java Program to illustrate the use of Arithmetic Exception in Java
public class Main {
    public static void main(String[] args) {
        int dividend = 10;
        int divisor = 0;
        try {
            int result = dividend / divisor; // Division by zero
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed.");
            // Additional error handling code can be added here
        }
    }
}
```

==========

Q.6 Differentiate the following

a. Throw and throws
b. final, finally, finalize

The throw and throws is the concept of exception handling where the throw keyword throw the exception explicitly from a method or a block of code whereas the throws keyword is used in signature of the method.

Sr. no.	Basis of Differences	throw	throws
1.	Definition	Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
2.	Uses	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.
3.	Syntax	The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
4.	Declaration	The keyword throw is used within the method.	The keyword throws is used with the method signature.
5.	Internal Implementation	We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

final Vs. finally Vs. finalize() in Java

A list of differences between final, finally and finalize is given below:

Aspect	final	finally	finalize()
Definition	final is the keyword and access modifier that is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java that is used to perform cleanup processing just before an object is garbage collected.
Applicable to	The final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	The finalize() method is used with the objects.
Functionality	(1) Once declared, a final variable becomes a constant and cannot be modified. (2) The final method cannot be overridden by sub class. (3) The final class cannot be inherited.	(1) finally block runs the important code even if an exception occurs or not. (2) The finally block cleans up all the resources used in a try block	The finalize() method performs the cleaning activities with respect to the object before its destruction.
Execution	The final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. Its execution is not dependant on the exception.	The finalize() method is executed just before the object is destroyed.
Inheritance /Overriding	Prevents method overriding and class inheritance.	Not related to inheritance or overriding.	Can be overridden from the Object class.
Use Case	To create constants, prevent subclassing or method overriding.	To ensure resource cleanup, like closing files, streams, etc., in exception handling.	To perform cleanup activities like memory/resource release before the object is collected.
Control	Gives compile-time control to avoid unintended changes.	Provides runtime control to handle cleanup post-exception or normal flow.	Invoked by JVM (Java Virtual Machine); the programmer cannot call it directly for cleanup.
Example	final int x=10;	Finally{closeResource	protected void finalize

==========

Q.5 Explain the following

a. Multiple catch block
b. catch all catch block
c. Multi catch block.
d. Nested try statement

In Java, **multiple catch blocks** allow us to handle different types of exceptions separately. It is beneficial when a single try block contains code that may throw different types of exceptions.

Therefore, a try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if we have to perform various tasks at the occurrence of other exceptions, use the **multi-catch block**.

Alternatively, we can separate each exception with | (pipe) symbol in a single catch block.

```
try {
    //code that may throw exception
}
catch (Exception1 | Exception2 | Exception3 | Exception4 e) {
   // Common handling logic for multiple exceptions
```

```
}
or
        try {
            //code that may throw exception
           }
            catch(Exception1 e1)  {
               //handle exception code
              }
            catch(Exception2 e2)   {
               //handle exception code
              }
            catch(Exception3 e3)   {
               //handle exception code
              }
           catch (Exception e) {
            // Handle any other exceptions (General exception block)
}
```

Nested Try Block in Java

In Java, using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.
For example, the inner try block can be used to handle ArrayIndexOutOfBoundsException, while the outer try block can handle the ArithmeticException (division by zero).

```
try {
    // Outer try block
    try {
        // Inner try block
    } catch (ExceptionType1 e1) {
        // Inner catch block
    }
} catch (ExceptionType2 e2) {
    // Outer catch block
}
```

Multi-catch block Allows two or more exceptions to be caught by the same catch clause. Used when two or more catch blocks has the same code. Each multi catch parameter is implicitly final. So it can't be assigned a new value.

Example_Program:

```
//To demonstrate multi-catch block
class demo8
{
 public static void main(String[] args)
 {
 int a[]={10,20,30,40,50,60};
 try
 {
 int c=a[10]/0;
 // int c=a[1]/0;
 System.out.println("result= "+c);
 }
 catch(ArithmeticException | ArrayIndexOutOfBoundsException ae)
 {
 System.out.println("Error : "+ae);
 }
 finally
 {
 System.out.println("this will execute");
 }
 }
}
```

Q.4 Explain uncaught Exception?

Uncaught Exceptions

In Java, exceptions are events that disrupt the normal flow of a program. Exceptions can be categorized into two main types: checked exceptions and unchecked exceptions. Checked exceptions must be explicitly handled by the programmer, usually through the use of try-catch blocks. Unchecked exceptions, on the other hand, are exceptions that are not checked at compile time and can occur during the execution of a program.

Uncaught exceptions specifically refer to exceptions that are not caught by any catch block within a program. When an uncaught exception occurs, the Java Virtual Machine (JVM) terminates the program's execution and prints a stack trace, providing valuable information about the cause and location of the exception.

==========

Q.8 What is thread, explain thread model and thread lifestyle?

What is the thread?

A thread is a lightweight subprocess. It is a separate path of execution because each thread runs in a different stack frame. A process may contain multiple threads. Threads share the process resources, but still, they execute independently.

What is multithreading?

Multithreading is a process of executing multiple threads simultaneously. Multithreading is used to obtain the multitasking. It consumes less memory and gives the fast and efficient performance. Its main advantages are:

- Threads share the same address space.
- A thread is lightweight process.
- The cost of communication between two processes is low.

Explain lifecycle of a Thread?

A thread can have one of the following states during execution:

New: In this state, a Thread class object is created using a new operator, but the thread is not alive. Thread does nott start until we call the start() method.

Runnable: In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selected by the thread scheduler.

Running: In this state, the thread scheduler picks the thread from the ready state, and starts running.

Waiting/Blocked: In this state, a thread is not running but still alive, or it is waiting for the other thread to finish.

Dead/Terminated: A thread is in terminated or dead state, if it satisfies any of the following condition:

- If it exits normally. It happens when the code of the thread has been entirely executed by the program.
- Because there occurred some unusual erroneous event, like a segmentation fault or an unhandled exception

==========

Q.9 Differentiate between on thread based multitasking and process based multitasking with realtime example?

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

==========

Q.10 What are different ways of creating thread. Explain with Example?

There are the following two ways to create a thread:

- By Extending Thread Class
- By Implementing Runnable Interface

Thread Class

The simplest way to create a thread in Java is by extending the Thread class and overriding its run() method. Thread class provide constructors and

methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Constructors of Thread Class

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

By Implementing Runnable Interface

Another approach to creating threads in Java is by implementing the Runnable interface. The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run(). This approach is preferred when we want to separate the task from the thread itself, promoting better encapsulation and flexibility.

public void run(): is used to perform action for a thread.

Starting a Thread

The **start() method** of the Thread class is used to start a newly created thread. It performs the following tasks:

- A new thread starts (with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

1) Creating Thread by Extending Thread Class

File Name: Multi.java

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

2) Java Thread Example by implementing Runnable interface

FileName: Multi3.java

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

==========

Q.11 What is synchronization: Explain Synchronized method and Synchronized block?.

Synchronization in Java is a critical concept in concurrent programming that ensures multiple threads can interact with shared resources safely. In a nutshell, synchronization prevents race conditions, where the outcome of operations depends on the timing of thread execution

Synchronization in Java tackles these problems through the capacity of a single thread to have exclusive access to either a synchronized block of code or a synchronized method associated with an object in question at a time. There are two primary mechanisms for synchronization in Java: synchronized methods and synchronized blocks.

Synchronized Methods

In Java, you can declare entire methods as synchronized which prevent multiple threads from accessing the method simultaneously. With this, synchronization becomes a simpler process because the mechanism is applied to all invocations of the synchronized method automatically.

Example: Synchronized Counter

```
class SynchronizedCounter {
    private int count = 0;
    public synchronized void increment() {
        count++;
    }
    public synchronized int getCount() {
        return count;
    }
}
```

With this modification, concurrent calls to increment() or getCount() will be synchronized, preventing race conditions.

Synchronized Blocks

Synchronized block provides exclusive access to shared resources, and only one thread is allowed to execute it in the same time frame. It's structured as follows:

```
synchronized (object) {
    // Synchronized code block
}
```

==========

Q.12 Explain interthread communication with producer consumer problem?

Inter-thread Communication in Java

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()
- 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.

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.

==========

Q.13 Explain the following

a. thread priority.
b. wait(),join(),yield()

Thread Priority in Java

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler schedules the threads according to their priority (known as preemptive scheduling).

public final int getPriority(): The java.lang.Thread.getPriority() method returns the priority of the given thread.
public final void setPriority(int newPriority): The java.lang.Thread.setPriority() method updates or assign the priority of the thread to newPriority. The method throws IllegalArgumentException if the value newPriority goes out of the range, which is 1 (minimum) to 10 (maximum).

```
public static int MIN_PRIORITY
public static int NORM_PRIORITY
public static int MAX_PRIORITY
```

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

==========

Q.14 what is String ? Explain String methods.

In Java, string is basically an object that represents sequence of char values. An array of characters works as a string in Java. For example:

```
char[] ch={'t','p','o','i','n','t'};
String s=new String(ch);
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it.

The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

Java String Class Methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

No.	Method	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	static String format(String format, Object... args)	It returns a formatted string.
4	static String format(Locale l, String format, Object... args)	It returns formatted string with given locale.
5	String substring(int beginIndex)	It returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
10	boolean equals(Object another)	It checks the equality of string with the given object.
11	boolean isEmpty()	It checks if string is empty.
12	String concat(String str)	It concatenates the specified string.
13	String replace(char old, char new)	It replaces all occurrences of the specified char value.
14	String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
16	String[] split(String regex)	It returns a split string matching regex.
17	String[] split(String regex, int limit)	It returns a split string matching regex and limit.
18	String intern()	It returns an interned string.
19	int indexOf(int ch)	It returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
21	int indexOf(String substring)	It returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
23	String toLowerCase()	It returns a string in lowercase.
24	String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
25	String toUpperCase()	It returns a string in uppercase.
26	String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
27	String trim()	It removes beginning and ending spaces of this string.
28	static String valueOf(int value)	It converts given type into string. It is an overloaded method.

==========

Q.15 what is StringBuilder & String Buffer - Explain its methods.

StringBuffer Class in Java

The Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as the String class except it is mutable, i.e., it can be changed.

StringBuffer()	It creates an empty String buffer with the initial capacity of 16.
StringBuffer(String str)	It creates a String buffer with the specified string..
StringBuffer(int capacity)	It creates an empty String buffer with the specified capacity as length.

append(String s)	It is used to append the specified string to this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double), etc.

insert(int offset, String s)	It is used to insert the specified string into this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double), etc

replace(int startIndex, int endIndex, String str)	It is used to replace the string from the specified startIndex and endIndex.

delete(int startIndex, int endIndex)	It is used to delete the string from the specified startIndex and endIndex.

reverse()	is used to reverse the string

capacity()	It is used to return the current capacity.

ensureCapacity(int minimumCapacity)	It is used to ensure the capacity is at least equal to the given minimum.

charAt(int index)	It is used to return the character at the specified position

length()	It is used to return the length of the string, i.e. total number of characters.

substring(int beginIndex)	It is used to return the substring from the specified beginIndex

substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

From <https://www.tpointtech.com/stringbuffer-in-java>

```
class Main {
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

```
class Main {
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

```
class Main {
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

```
class Main {
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

```
class Main {
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

StringBuilder Class in Java

Java StringBuilder class is used to create mutable (modifiable) String. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Method	Description
public StringBuilder append(String s)	It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	It is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	It is used to reverse the string.
public int capacity()	It is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	It is used to return the character at the specified position.
public int length()	It is used to return the length of the string i.e. total number of characters.
public String substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.