ASSIGNMENT-3

04/03/24

Q1) Differentiate b/w checked exception and unchecked exception

Checked Exception:

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time

Unchecked Exception:

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time but they are checked at runtime.

Q2) Explain all the keywords of exception handling mechanism.

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.

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.

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

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

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.

Q3) Differentiate between final, finally and finalized keyword

final	finally	finalized
1. final is the keyword and access modifier which 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 which is used to perform clean up processing just before object is garbage collected
2. Final keyword is used with the classes, methods and variables	Finally block is always related to the try and catch block in exception handling	finalize() method is used with the objects
3. Once declared, final variable becomes constant and cannot be modified.	finally block runs the important code even if exception occurs or not	finalize method performs the cleaning activities with respect to the object before its destruction.
Execution of Final method is executed only when we call it	finally block is executed as soon as the try-catch block is executed	finalize method is executed before the object is destroyed.

4 Q. Differentiate between throw and throws keyword.

Throw

1) Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.

2) Using throw keyword, we can declare both checked and unchecked exceptions. However, the throw keyword can be used to propagate checked exceptions only.

3) The throw keyword is followed by an instance of Exception to be thrown.

4) Throw is used within the method.

5) We are allowed to throw only one exception at a time i.e we cannot throw multiple exceptions.

Throws

1) 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) Type of exception using throws keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.

3) The throws keyword is followed by class names of Exception to be thrown.

4) Throws is used with the method signature.

5) We can declare multiple exceptions using throws keyword that can be thrown by the method.

5/03/24

a) Write a simple example using nested try block.

NestedTryBlock.java

```
public class NestedTryBlock{
    public static void main(String args[]){

    try{
        try{
            System.out.println("going to divide by 0");
            int b=39/0;
        }
    catch (ArithmeticException e)
        {
            System.out.println(e);
        }

    try{
        int a[] = new int[5];
            a[5]=4;
        catch{ ArrayIndexOutOfBoundsException e)
            {   System.out.println(e);
            }
        System.out.println("other statement");
    }
    catch (Exception e)
        {   System.out.println("handled the exception");
        }
    System.out.println("normal flow...");
    }
}
```

Q) Explain exception Propagation with an example.

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method. If not caught there, the exception again drops down to the previous method, and so on, until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.

Example

```
class TestExceptionPropagation1{
    void m(){
        int data = 50/0;
    }
    void n(){
        m();
    }
    void p(){
        try{
            n();
        }catch(Exception e){ }
            System.out.println("exception handled");
        }
    public static void main(String args[]){
    TestExceptionPropagation1 obj = new TestException
    Propagation1();
        obj.p();
        System.out.println("normal flow...");
    }
}
```

Q) Can we rethrow an exception in Java give an example.

Yes, you can rethrow an exception in Java. This is useful when you catch an exception in a catch block but cannot handle it within that block. By rethrowing the exception, you propagate it to a higher level in your code where it can be handled appropriately

Java example

```
try{
  //code that might throw an exception
}catch (Exception e){
    // Log the exception (or perform some other
      handling)
    throw e; // Rethrow the exception
}
```

Q. Check a given mobile number is valid or not using exception handling mechanism.

While exception handling is typically used for unexpected errors, it's not the most suitable approach for validating mobile numbers. Here's a more common way to validate mobile numbers using regular expression

Exam

```
import java.util.regex.Pattern;
public class MobileNumberValidator{
    public static boolean isValidMobileNumber(String
    mobileNumber){
String regex= "^(?:(?:\\+|0{2})91)?(?:7|8|9)
                \\d{9}$";
}}  Pattern pattern = Pattern.compile(regex);
```

11/05/24

Q. What is String constant pool?

String pool is nothing but a storage area in Java heap where string literals stores. It is also known as String Intern Pool or String Constant Pool. It is just like object allocation. By default, it is empty and privately maintained by the Java String class. Whenever we create a string the string object occupies some space in the heap memory. Creating a number of strings may increase the cost and memory too which may reduce the performance also.

When we create a string literal, the JVM first check the literal is already present in the pool, it returns a reference to the pooled instance. If the literal is not present in the pool, a new String object takes place in the string pool.

Q. Why String objects are immutable in Java?

Strings in Java are specified as immutable, as seen above because strings with the same content share (shortages) storage in a single pool to minimize creating a copy of the same value. That is to say, once a string is generated, its content cannot be changed and hence changing content will lead to the creation of new String. Otherwise, the update will affect the heap value, and then all other String references that share the same storage location

will be changed. This change has the potential to be unpredictable which is why it is not desirable.

(Q) List the different comparison techniques in Java?

There are three ways to compare string in Java.

(1) By using equals() Method.
(2) By String == Operator
(3) By CompareTo() Method.

(1) By using equals() Method.

The String class equals() method compares the original content of the string. It compares values of string for equality. String class provides the following two methods:

(2) By using == operator compares references not values.

(3) String compareTo() Method demonstrates the use of == operator used for comparing two String objects. The String class CompareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

(Q) Difference between String and StringBuffer.

String	StringBuffer
1) The string class is immutable	1) The StringBuffer class is mutable
2) String is slow and consumes more memory when we Concatenate too many strings because every time it creates new instance	2) StringBuffer is fast and consumes less memory when we concatenate strings.

2) String class override the equals() method of object class. so you can compare the contents of two strings by equals() method	2) StringBuffer class doesn't Override the equals() method of object class.
4) String class is slower while performing concatenation operation	4) StringBuffer class is faster while performing concatenation operation.
5) String class uses String constant pool.	5) StringBuffer uses Heap memory.

Q How to perform two tasks by two threads?

There are two common ways to perform two tasks by two threads in most programming languages

(1) Using Separate Runnable Objects: Define two separate classes that implement the Runnable interface (Java) or similar concept in other languages

(2) Extending the Thread class:

- Create two separate classes that extend the Thread class or similar concept in other languages.

(3) Thread synchronization: If your tasks involve shared data, you might need to implement synchronization mechanisms

(4) Thread Safety: Make sure your code within the threads is thread-safe, meaning it can be accessed by multiple threads without issues.

Q. How to perform multithreading by anonymous class?

```
public class Main{
    public static void main(String[] args){
        Thread thread = new Thread(new Runnable(){

    public void run(){
        for(int i=0; i<5; i++){
    System.out.println("Thread : " + Thread.currentThread());

try{
    Thread.sleep(1000);
    }catch(InterruptedException e){
        e.printStackTrace();

        }
    }
}
});

thread.start();

for(int i=0; i<5; i++){
    System.out.println("Main Thread : " + Thread.currentThread());

    try{
     Thread.sleep(1000);
}catch(InterruptException e){
            e.printStackTrace();

    }
}
}
```

Q) What is the Thread Scheduler and what is the Difference between preemptive scheduling and time slicing?

The thread scheduler is a part of the operating system responsible for allocating CPU time to threads. Its primary job is to decide which thread should run next based on priority, time and ensuring fairness and preventing monopolization. Time slicing, a form of preemptive scheduling, divides CPU time into intervals, ensuring all threads receive execution time. It prevents starvation and enhances responsiveness in multitasking environments by limiting each thread's maximum CPU occupancy. Both approaches manage thread execution to optimize system performance and fairness.

Q) What happens if we start a thread twice?

- Illegal State: When you call the start() method on a thread object for the first time, the thread transitions to a running state and begins executing the code within its run() method.
- Second Attempt: If you try to call start() again on the same thread object, the program recognizes that the thread is already running or has finished execution.

This throws an exception, typically IllegalThreadStateException or similar, indicating the illegal attempt to start an already started thread.

Q. What happens if we call the run() method instead of start() method?

Calling run() directly bypasses thread creation. Instead of spawning a new thread, the code defined in run() executes on the current thread like a regular method call.

- No concurrency: The tasks won't run simultaneously. They'll execute one after another in the current thread's sequence.
- No thread management: You won't benefit from features like thread scheduling or context switching.

Q. What is the purpose of join method.

The join method in threading is used for synchronization between threads.

- Join makes the current thread wait until another thread finishes its execution. This ensures the program doesn't proceed until a specific task (in the other thread) is complete. It's helpful for tasks that rely on the output or completion of another thread.

Q. Why JVM terminates the daemon thread if no user threads are remaining?

JVM terminates daemon threads when no user threads are left because daemon threads exist to serve user threads. Here's why:

- Supportive role: Daemon threads are background helpers that provide services & perform maintenance tasks for user threads. (the main program and other critical tasks)
- No purpose alone: Once user threads finish, there's no need for the JVM to keep daemon threads running as they wouldn't have anything to support.
- Resource Management: Terminating daemon threads frees up resources for the system.

Q. What is the shutdown hook?

A shutdown hook in Java is a special mechanism that lets you run some code just before the JVM terminates. It's like a last-minute chore before the shutdown. You can register code to be executed using Runtime.getRuntime().addShutdownHook(thread). This is useful for tasks like closing database connections, saving application state, or cleaning up resources.

Q What is Garbage collection?

Garbage collection is a form of automatic memory management in Computer science. It frees programmers from manually keeping track of memory used by objects in their programs.

- Memory allocation: When a program runs, it creates objects that occupy memory space.
- Unused objects: Over time, some objects become unnecessary as the program progresses.

Automatic reclaiming: Garbage collection identifies these unused objects and reclaims the memory they occupy.

- Program benefit: This frees the programmer from manually deleting objects and reduces the risk of memory leaks.

Q. What is the purpose of finalize() method?

The finalize() method in Java, inherited from the Object class, serves a specific purpose in garbage collection:

- Last chance cleanup: It's designed as a safety net to perform essential cleanup tasks on object before they are permanently removed by the 'garbage collector.
- Resource management: Typical use cases involve closing external resources like files, network connection, or database connection associated with the object.
- Not guaranteed: It's important to note that the Java Virtual Machine doesn't guarantee that finalize() will always be called. So, it shouldn't be relied upon for critical cleanup tasks.

Q. What does the gc() method?

The gc() method in Java is a suggestion to the JVM to run the garbage collector. It

- Request, not a guarantee: Calling gc() doesn't guarantee immediate garbage collection. The JVM might decide to collect later based on its own optimization strategies.
- Request on potential performance impact: Running garbage collection can cause temporary pauses in your program. It's generally recommended to avoid relying on gc() for memory management and let the JVM handle it automatically.

Q. What is Synchronization and why use synchronization.

Synchronization in programming ensures coordinated access to shared resources by multiple threads. Imagine two chefs working on the same dish. They need to take turns using the same tools to avoid a mess.

- Data consistency: When multiple threads access and modify the same data concurrently, it can lead to inconsistencies. Synchronization prevents this by making sure only one thread modifies the data at a time.

Race conditions: Without synchronization, threads can enter a race condition, where the

Out-come depends on unpredictable timing. Synchronization avoids this by ensuring predictable order of execution for critical sections of code.

Q. What is the difference between synchronized method and synchronized block?

Both synchronized methods and synchronized blocks achieve synchronization, but they differ in scope and flexibility.

Synchronized method: Locks the entire object. No other thread can call any synchronized method on the same object while the current method is executing.

Synchronized block: Locks only the specific object mentioned within the block. other synchronized methods on the same object can still be called by other threads as long as they don't access the locked object within the block.

- Flexibility:
  Synchronized method: Less flexible, locks everything related to the object.
- Synchronized block: More flexible, allows you to synchronize only the critical section of code that needs it.

Q. What are two ways to perform static synchronization?

In Java, there isn't exactly two ways to perform static synchronization, but there are two different ways

To achieve synchronization using the synchronized keyword that can be applied in a static context:

1) Static synchronized methods: Declare a method as synchronized and static. This synchronizes on the class object itself, meaning only one thread can execute this specific method at a time, regardless of how many objects of the class exist.

2) Synchronized blocks on the class object: within a method, use a synchronized block that explicitly synchronizes on the class object (className.class). This allows for more granular control within a method, synchronizing only a specific code section that interacts with shared data.

Q. What is deadlock and when it can occur?

Deadlock in Java is a severe concurrency issue that occurs when two or more threads become permanently blocked, waiting for each other to release resources. Imagine two people holding onto each other's coats, each needing the other's coat to proceed.

1. Mutual exclusion: Each thread needs exclusive access to one or more resources.

2. Hold and wait: One thread holds a resource while waiting for another resource held by another thread.

3. No preemption: Resources cannot be forcibly taken away from a thread.

4. Circular wait: There's a circular chain of dependencies between threads waiting for resources from each other.

Q. What is inter-thread - Communication & Cooperation

Inter-thread communication, also known as cooperation, refers to mechanisms that allow threads to exchange information & coordinate their execution in a program. It's essential for multithreaded programs to function effectively.

Here's why inter-thread communication is important:

- Sharing data: Threads can exchange data to complete tasks efficiently.
- Synchronization: It helps coordinate access to shared resources, preventing data corruption.
- Signaling: Threads can notify each other when tasks are complete or conditions change.

Common methods for inter-thread communication include

- Shared objects: Threads can access and modify the same object, but with proper synchronization to avoid conflicts.
- Wait and notify: Threads can wait for a specific condition to be met before proceeding, notified by another thread.
- Concurrent data structures: Thread-safe data structures allow for safe access and modification of data by multiple threads.