Assessment - 04

Java SE New Feature

Section I [MCQ Questions] **1 Marks Each**

1) When a developer creates an anonymous inner class, the new class is typically based on which one of the following?
   enums
   Executors
   Functional interfaces
   Static variables

2) Which is true about the parameters passed into the following lambda expression?
   (t,s) -> t.contains(s)
   Their type is inferred from the context.
   Their type is executed.
   Their type must be explicitly defined.
   Their type is undetermined.

3) Which of the following is correct about Java 8?*
   Optional curly braces - No need to use curly braces in the expression body if the body contains a single statement.
   Optional return keyword - The compiler automatically returns the value if the body has a single expression to return the value.
   Both of the above.
   None of the above.

4) Which of the following functional interface represents an operation that accepts a single input argument and returns no result?*
   BooleanSupplier
   Consumer<T>
   DoubleBinaryOperator
   DoubleConsumer

5) Which of the following functional interface represents a function that accepts a double-valued argument and produces a long-valued result?*
   DoubleToLongFunction
   DoubleUnaryOperator
   Function<T,R>
   IntBinaryOperator

6) Which of the following functional interface represents a function that accepts a long-valued argument and produces a double-valued result?*
LongToDoubleFunction
LongToIntFunction
LongUnaryOperator
ObjDoubleConsumer<T>

7) What is the purpose of the map method of stream in java 8?*
Iterate each element of the stream.
Map each element to its corresponding result.
Eliminate elements based on a criteria
Reduce the size of the stream

8) Stream takes Collections, Arrays, or I/O resources as input source.*
True
False

9) Which class from the Java Date and Time API can be used to represent a date-time without a time zone in the ISO-8601 calendar system?*
Instant
ZonedDateTime
LocalDate
LocalDateTime

10) How can you obtain the current machine's time without date information?*
LocalDateTime.now()
ZonedDateTime.now()
LocalTime.now()
Instant.now()

11) How can you change the reference type of an object?*
By calling getReference()
By casting
By declaring a new reference and assigning the object
None of The Mentioned

12) Which of the following is not a conventional abbreviation for use with generics?*
T: Table
E: Element
K: Key
V: Value

13) Which of the following objects are checked exceptions?*
All objects of type Throwable
All objects of type Exception
All objects of type Exception that are not of type RuntimeException
All objects of type Error

14) Which Statement method executes a SQL statement and returns the number of rows affected?*
stmt.execute(query);
stmt.executeUpdate(query);
stmt.executeQuery(query);
stmt.query(query);

15) Given:

```
  String[] params = {"Bob", "Smith"};
  String query = "SELECT itemCount FROM Customer " +
               "WHERE lastName='?' AND firstName='?'";
 try (PreparedStatement pStmt = con.prepareStatement(query)) {
     for (int i = 0; i < params.length; i++)
         pStmt.setObject(i, params[i]);
     ResultSet rs = pStmt.executeQuery();
     while (rs.next()) System.out.println (rs.getInt("itemCount"));
  } catch (SQLException e){ }
```

Assuming there is a valid Connection object and the SQL query will produce at least one row, what is the result?

Each itemCount value for customer Bob Smith
Compiler error
A run time error
A SQLException

16) Predict the Output of the following code:

```
 1: abstract public class Hello {
 2:   public Hello() {
 3:
 4:   }
 5:
 6:   private int hello() {
 7:     return 10;
 8:   }
 9:
10:   private static disp() {
    11:
    12:   }
    13: }
```

Predict the Output.

```
import java.io.*;
import java.util.*;
```

17) Predict the Output. A class 'Movie' that implements Comparable

```
class Movie implements Comparable<Movie>
{
    private double rating;
    private String name;
    private int year;

    // Used to sort movies by year
    public int compareTo(Movie m)
    {
        return this.year - m.year;
    }

    // Constructor
    public Movie(String nm, double rt, int yr)
    {
        this.name = nm;
        this.rating = rt;
        this.year = yr;
    }

    // Getter methods for accessing private data
    public double getRating() { return rating; }
    public String getName()   {  return name; }
    public int getYear()            {  return year;  }
}

// Driver class
class Main
{
    public static void main(String[] args)
    {
        ArrayList<Movie> list = new ArrayList<Movie>();
        list.add(new Movie("Force Awakens", 8.3, 2015));
        list.add(new Movie("Star Wars", 8.7, 1977));
        list.add(new Movie("Empire Strikes Back", 8.8, 1980));
        list.add(new Movie("Return of the Jedi", 8.4, 1983));

        Collections.sort(list);

        System.out.println("Movies after sorting : ");
        for (Movie movie: list)
        {
            System.out.println(movie.getName() + " " +
                               movie.getRating() + " " +
                               movie.getYear());
        }
    }
}
```

18) Predict the Output.

```
class Base
{
                int x;

                Base(int _x)
                {
                        x = _x;
                }
}

class Derived extends Base
{
                int y;

                Derived(int  _x,  int  _y)
                {
                        super(_x);
                        y = _y;
                }

                void display()
                {
                        System.out.println(" X  = " + x  + " Y =  " + y);
                }
}

public class Main
{
```

```
                public static void main(String args[])
                {
                        Derived d = new Derived(10,20);
                        d.display();
                }
}
```

19) Predict Output.

```
abstract class Base
{
     public void fun()
     {
             System.out.println(" Base Fun() Called ");
     }
}

class Derived
{
     public void fun()
     {
             System.out.println(" Derived Fun() Called ");
     }
}

class Main
{
     public static void main(String args[])
     {
             Base b = new Derived();
             b.fun();
     }
  }
```

20) A Class that represents use-defined exception

```
class MyException extends Exception
{
   public MyException(String s)
   {
      // Call constructor of parent Exception
      super(s);
   }
}
```

```
// A Class that uses above MyException
public class Main
{
   // Driver Program
   public static void main(String args[])
   {
      try
      {
         // Throw an object of user defined exception
         throw new MyException("Java");
      }
      catch (MyException ex)
      {
         System.out.println("Caught");

         // Print the message from MyException object
         System.out.println(ex.getMessage());
      }
   }
}
```

Section II [Descriptive Questions] 5 Marks Each

1) Explain the Usage of Lambda Expression, Different Functional Interfaces*

LAMBDA EXPRESSION:
The term lambda means calculus which is a theoretical framework for describing functions and evaluation. They are implemented when and wherever needed.
They are similar to anonymous inner classes. Both lambdas and anonymous inner classes are defined for passing them as arguments. They are the implementation of whole class with several methods and not just a single method.
eg: (int x, int y) -> x+y,   s->s.contains("berry");
USES:
1) No need of explicit datatype
2) Can be used as arguments, variables
3) Used of grouping classes specific to some logic

FUNCTIONAL INTERFACE:
Lambda expressions rely on functional Interfaces. A functional interface is an interface with a single abstract method, which can be implemented by a lambda expression. Java provides many built-in functional interfaces in the java.util.function package.
1) Consumer: An expression that consumes, perform some operation and never return anything.

eg: Consumer<String> print = s -> System.out.println(s);
2) Predicate: An expression that returns boolean.
eg: Predicate<Integer> isEven = x -> x % 2 == 0;
3) Function: An expression the accepts one input argument and produces a result. Used for transforming data. T to U
eg: Function<String, Integer> length = s -> s.length();
4) Supplier: Provides an instance of T
eg: Supplier<Double> suplier = () -> new Sales.supply();

2) Importance of Date Time API in Java 8, What is Fluent API. And Write a Sample Program to Illustrate LocalDateTime Functionality.

For development of applications, we often need to represent date and time locally. And representing the dates in strings may cause a number of operations to perform like using string.split() method and validate like if that is a leap year, if the month has 30 or 31 or 28 days and so on...
Date and time API an improved version of java.util.date and java.util.Calender
as they don't provide thread safety, less methods to use, less time zone functionality and performance. And hence java came up with Date time API that calculates on:
1) Currents date and time
2) Differences in date time like seconds, milli secs, hours etc
3) time and date respected to time zones
4) number of days in a month, locale calendar functionality.
5) Time duration in minutes or in periods like years, months

```
public class LocalDateTimeExample {
   public static void main(String[] args) {
      LocalDateTime now = LocalDateTime.now();
      System.out.println("Current DateTime: " + now);

      LocalDateTime specificDateTime = LocalDateTime.of(2024, Month.MAY, 19, 14, 09, 
52);
      System.out.println("Specific DateTime: " + specificDateTime);
   }
}
```

OUTPUT:
```
Current DateTime: 2025-05-19 T 17:09:45.123
Specific DateTime: 2024-05-19 T 14:09:52
```

A Fluent API is a programming style that allows method chaining and readable code that flows like natural language. It typically returns the current object or a related object from each method call, enabling multiple method calls in a single statement.
eg: myBday = Year.of(2001).atMonth(JUNE).atDay(22);

3) Explain the Importance of Modular Programming in Java

Before modularity came JARs were used for reuse of code by extending classes or implementing interfaces. But the typical use of JAR was to pack applications and library. So, if a class wanted to be imported, we just add the dependency JAR to make it available on a different project. And here comes the problem that a JAR can have multiple duplicate classes, and some application may need 100s of JAR files which made it unethical use of JARs and hence came up with Modularity.

A module is a set of packages and designed for reuse with strong Encapsulation. Modularity was introduced in JDK 9 that adds a higher level of aggregation. A module is a group of related packages or classes and resources like images, xml files. These modules also support in exporting packages and concealing packages. Modules help in reliability, maintainability and security. It is fully fledged java component. Module system is usable at all levels of application, libraries and JDK.
The advantages of Module system are Dependencies, Interfaces and encapsulation. It declares the dependency of modules and makes restriction on packages.

A module must be declared in module-info.java. Each module declaration should begin with the keyword "module". The body can be empty or may contain keywords like requires and exports.
Advantages/ Importance:
1) String Encapsulation
2) Organized code structure
3) Scalable and maintainable
4) Reduce Memory
5) Reduce compile time and thus enhances developer productivity, code quality, and application robustness by breaking down complex systems into manageable, reusable, and independently testable modules.

4) What is Generics? Why is it so important in terms of programming ?*

Generics were introduced in Java 5. Generics in Java are a language feature that allows classes, interfaces, and methods to operate on objects of various types while providing compile-time type safety by using parameterized types or generic types. It enables developers to write flexible and reusable code that can work with different data types without sacrificing type safety or requiring explicit casting everytime.

Importants of generics in Programming:
1) Code Reusability: They allow writing a single generic class or method that can handle multiple types, reducing code duplication and improving maintainability.
2) Avoid Explicit Casting: Before generics, collections stored objects as raw types requiring explicit casting when retrieving elements, which may lead to errors. Generics remove this need, making code cleaner and safer.

3) Code Readability: By specifying types explicitly, generics make code easier to understand and maintain, as the intended data types are clear.
4) Early Error Detection: The compiler performs stronger type checks, catching potential bugs early in the development cycle.
5) Performance Efficiency: Generics avoid unnecessary type conversions and reduce boilerplate code without runtime overhead due to type erasure.
6) Type Safety: Generics ensure that type mismatches are caught at compile time rather than at runtime, preventing errors like ClassCastException and making programs more robust.

5) What are Threads ? Explain Thread Life Cycle. And Classes in java.concurrency Package for Working with Threads.*

Thread is a single part of a process that is schedules for execution. Threads run concurrently. Threads share the same memory space but execute independently, enabling efficient utilization of CPU resources.

Thread Life Cycle in Java:
A thread in Java goes through several states during its lifetime. Understanding these states helps in writing effective multithreaded programs.

1) New (Created) state;
A thread is in this state when an instance of Thread is created but start() has not been called yet. The thread is not yet alive.

2) Runnable: After calling start(), the thread moves to the runnable state. The thread is ready to run and waiting for CPU scheduling. It may be running or waiting for the CPU.

3) Running: When the thread scheduler selects the thread from the runnable pool and it starts executing its run() method. This is the actual execution phase.

4) Blocked / Waiting: The thread enters this state when it is waiting for a resource or another thread's action.
For example, waiting to acquire a lock or waiting for a notification. Or occurs when methods like sleep(milliseconds), wait(milliseconds), or join(milliseconds) are called. And this can be revoked by using methods like wait(), join(), notify()... Or by using synchronized locks for locking resources until they are fully used.

5) Terminated (Dead): The thread reaches this state when it completes its execution or is terminated abruptly. Once terminated, the threads are killed and cannot be restarted. If done, throws an IllegalThreadStateException.

Concurrency package:
Contains classes that are useful in concurrent programming.

1) ExecutorService : Interfaces and implementations to manage thread pools and task execution asynchronously. It abstracts thread creation, scheduling, and lifecycle management, simplifying concurrent programming.
2)Callable: Callable<V> is a functional interface similar to Runnable, but it can return a result and throw checked exceptions. It represents a task that returns a value of type V after completion. Unlike Runnable which cannot return a result, Callable allows tasks to produce results and propagate exceptions.
3) Future: Future<V> represents the result of an asynchronous computation. It acts as a handle to retrieve the result once the task completes.