ASSESSMENT - 03

SECTION - I

Q1)
B. Association defines a relationship between classes of objects which allows one object instance to delegate responsibilities to another object on its behalf.

Q2)
D. class L extends G and implements M

Q3)
D. class Order { private Customer customer; }

Q4)
B. Class

Q5)
E. Encapsulation

Q6)
J. Attributes

Q7)
A. package, import, class

Q8)
A. String args []
B. String [] args

Q9)
A. number_1
B. number_a
C. $1234

Q10)
D. synchronized
E. throws

Q11)
A. A static method may be invoked before even a single instance of the class is constructed.
B. A static method cannot access non-static methods of the class.
C. Abstract modifier can appear before a class or a method but not before a variable.

Q12)
B. public abstract void method();

Q13)
B. An abstract class can be extended.
C. A subclass of a non-abstract superclass can be abstract.
D. A subclass can override a concrete method in a superclass to declare it abstract.
E. An abstract class can be used as a data type.

Q14)
A. garbage collection cannot be forced.

Q15)
B. x[0] is null.
D. x[24] is 0.0

Q16)
C. x=13, a=7, b=8

Q17)
E. To read/write data from/to a file

Q18)
C. No methods.

Q19)
B. Class.forName("oracle.jdbc.driver.OracleDriver")

Q20)
Type 1: JDBC-ODBC bridge driver
Type 2: Native-API Driver(Partially Java) - IBM
Type 3: Network Protocol Driver(Fully Java) - Multi Tier
Type 4: Thin Driver (pure Java)

SECTION - II

Q1)
Null Pointer Exception

Q2)
Not Same

Q3)
Total 2 Objects Created

Q4)
Compiler Error (missing + operation in System.out.println();)

Q5)
Copy Constructor Called
[10.0 + 15.0 i]

Q6)
Compiler Error (BufferedReader)
Try Catch Block missing

Q7)
 Base Class Constructor Callled
 Derived Class Constructor Callled

Q8)
X = 10 Y = 20

Q9)
Compiler Error (Absence of inheritance)

Q10)
Compiler Error (Wrong Main Class() declaration)
Compiler Error (Class with name [outer] not available)

SECTION - III

Q1)
ZIP FILE - (Srigowri N_Assessment_01)

Q2)
Member Inner class: A member inner class is defined at the member level of the outer class. It has the following characteristics:

1) Can access all members of the outer class including private members
2) Can be declared with any access modifier.
3) Cannot contain static members if the inner class itself is not static
4) Can have inheritance

```
class OuterClass {

    private int outer = 10;

    public class MemberInnerClass {

        private int inner = 20;
        public void display() {
            System.out.println("Outer class variable: " + outer);
            System.out.println("Inner class variable: " + inner);
            outerMethod();
        }
    }

    private void outerMethod() {
        System.out.println("Outer method called");
    }

    public void createInner() {
        MemberInnerClass inner = new MemberInnerClass();
        inner.display();
    }
}

public class Main {

    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        OuterClass.MemberInnerClass inner = outer.new MemberInnerClass();
        inner.display();
        System.out.println("\nUsing Inner method:");
        outer.createInner();
    }
}
```

Q3)
Constructor: Special method used to initialize objects when they are created. They are called when an object is created using the new keyword. Java supports several types of constructors:
Types of Constructors:
1. Default Constructor: No parameters, provides default values. In the absence of constructors JVM implicitly creates a default constructor. But in the presence of a parameterized constructor, creating an object with zero parameters will throw a compiler error.

```
class Student {
        Student() {
                System.out.println("Student object created");
        }
}
```

2. Parameterized Constructor: Accepts parameters to initialize fields. The Name, Type of the data and the sequence placed should be the same as that of the method signature while creating the objects of that class.

```
class Student {
String name;
int id;
        Student(String name, int id) {
                this.name = name;
                this.id = id;
        }
}
```

3. Copy Constructor: Creates an object using another object of the same class. This can now be done by using the copyOf() method.

```
class Student {
        String name;
        int id;
        Student(String name, int id) {
                this.name = name;
                this.id = id;
        }
        Student(Student s) {
                this.name = s.name;
                this.id = s.id;
        }
}
```

Q4)

Customer Class:

```
public class Customer {

   private int id;
   private String name;
   private static ArrayList<Customer> customers = new ArrayList<>();

   public Customer(int id, String name) {
      this.id = id;
      this.name = name;
      customers.add(this);
   }

   public static Customer findCustomerById(int searchId) {
      for (Customer customer : customers) {
         if (customer.id == searchId) {
                return customer;
            }
        }
        return null;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "Customer [ID=" + id + ", Name=" + name + "]";
    }
}
```

Order Class:

```
import java.util.ArrayList;
public class Order {
    private int orderId;
    private Customer customer;
    private ArrayList<String> orderItems = new ArrayList<>();

    public Order(int orderId, Customer customer) {
        this.orderId = orderId;
        this.customer = customer;
    }

    public void addItem(String item) {
        orderItems.add(item);
        System.out.println("Added " + item + " to Order " + orderId);
    }

    public boolean removeItem(String item) {
        boolean removed = orderItems.remove(item);
        if (removed) {
            System.out.println("Removed " + item + " from Order " + orderId);
        } else {
            System.out.println("Item " + item + " not found in Order " + orderId);
        }
        return removed;
    }
}
```