Q1. What is an object-oriented paradigm?

A)The Object-oriented paradigm is a programming paradigm that organizes software design and development around the concept of “objects”.Objects represent real world entities or concepts and encapsulate data and the operations that can be performed on the data.

The key principles of the object oriented paradigm include:

1.Objects:

Objects are instances of classes and represent tangible entities in the program.They encapsulate data(attributes or properties)and behaviors(methods or functions)related to that entity.

2.Classes:

A class is a blueprint or template for creating objects.It defines the properties and behaviors that its objects will have.Objects are instances of classes , and each object created from a class is unique but shares the same structure and behavior defined by a class.

3.Encapsulation:

Encapsulation is the bundling of data and methods that operate on that data within a single unit,the object.It helps hide the internal details of an object and expose only what is necessary for the outside world,promoting information hiding and reducing complexity.

4.Inheritance:

Inheritance is a mechanism that allows a class(subclass or derived class)to inherit properties and behaviors from another class(superclass or base class).This promotes code reuse,as common functionality can be defined in a base class and inherited by multiple subclasses.

5.Polymorphism:

Polymorphism allows objects to be treated as instances of their parent class,even if they are actually instances of a subclass.This concept enables flexibility in programming and is expressed through method overloading and method overriding.

6.Abstraction:

Abstraction involves simplifying complex systems by modeling classes based on the essential properties and behaviors relevant to the problem domain.It allows developers to focus on relevant details while ignoring unnecessary complexities.

Object -oriented programming(OOP)languages,such as Java,C++,Python,and many others,are designed to support the object-oriented paradigm.OOP facilitates modular and scalable software development by providing a clear and organized way to structure code.The use of objects ,classes,inheritance,and other OOP principles helps improve code readability,reusability,and maintainability,making it easier to design and manage complex software systems.

Q2. Can we modify the throws clause of the superclass method while overriding it in the subclass?

In Java,when you override a method in a subclass,you must adhere to certain rules,and one of those rules is related to the throws clause.The general rule is that the overridden method in the subclass cannot declare more checked exceptions than the overridden method in the superclass.However , the overriding method can choose not to throw any exceptions or throw fewer exceptions(including subclasses of the exceptions thrown by the superclass method).

Here are the rules regarding the throws clause while overriding a method in a subclass:

1.No new checked exceptions:

If the superclass method throws a checked exception,the overriding method in the subclass can either:

- **Not throw any exception**
- **Throw the same exception or its subclass.**

In other words, the overriding method cannot introduce new checked exceptions that are not present in the throws clause of the superclass method.

2.Unchecked exceptions:

The overriding method in the subclass can throw any unchecked exception ,regardless of whether the superclass method declares such exceptions.

Here is an example to illustrate these rules:

```
class Superclass {
    // Superclass method with 'throws' clause
    void exampleMethod() throws Exception {
        // method implementation
    }
}

class Subclass extends Superclass {
    // Overriding method without 'throws' clause
    // This is allowed because the overridden method in the superclass does not throw any unchecked exceptions.
    @Override
    void exampleMethod() {
        // overridden method implementation
    }

    // Overriding method with a broader 'throws' clause
    // This is allowed because it includes the same exception declared in the superclass method.
```

```
    @Override
    void exampleMethod() throws Exception, AnotherException {
        // overridden method implementation
    }

    // Overriding method with 'throws' clause that is not allowed
    // This will result in a compile-time error because it introduces a new checked exception.
    // @Override
    // void exampleMethod() throws IOException {
    //     // overridden method implementation
    // }
}
```

In the example above,the Subclass overrides the exampleMethod from the Superclass.It follows the rules regarding the throws clause,demonstrating that the overridden method can throw the same exceptions or their subclasses but cannot introduce new checked exceptions.

Q3. Differentiate between process and thread?

Process:

1.Definition:

- A process is an independent program that runs in its own memory space and performs a specific task.
- It is the execution of a program,which includes its code , data, and system resources.

2.Isolation:

Processes are isolated from each other.Each process has its own memory space,and one process cannot directly access the memory of another process.

3.Communication:

Inter-process communication(IPC) is required for processes to communicate with each other.Common mechanisms include pipes,sockets,and message passing.

4.Resources Overhead:
Processes have higher resource overhead compared to threads because they have separate memory spaces and require additional resources for communication and synchronization.

5.Creation Time:

Creating a new process is generally more time-consuming and resources-intensive compared to creating a new thread.

6.Fault Tolerance:

Processes are more robust in terms of fault tolerance.If one process fails,it does not necessarily affect others.

7.Scalability:

Processes are heavier in terms of resource consumption,making them less suitable for scenarios where a large number of lightweight tasks need to be performed concurrently.

Thread:

Definition:

- A thread is a lightweight,smaller unit of a process that runs within the context of a process.
- Threads share the same code and data space but have their own execution stack and register values.

Isolation:

- Threads within the same process share the same memory space and resources.
- They can directly access the memory of other threads within the same process.

Communication:

- Threads can communicate more easily since they share the same memory space.
- However,developers must be cautious about potential data race conditions and use synchronization mechanisms.

Resource Overhead:

- Threads have lower resource overhead compared to processes because they share the same memory space and resources.Communication between threads is more efficient.

Creation Time:

- Creating a new thread is generally faster and requires less overhead compared to creating a new process.

Fault Tolerance:

- If one thread in a process fails(e.g.,encounters an unhandled exception),it can potentially affect other threads within the same process.

Scalability:

- Threads are more suitable for scenarios where a large number of lightweight tasks need to be performed concurrently because they have lower resource overhead.

Summary:

In summary,a process is an independent program with its own memory space,whereas a thread is a smaller unit of a process that shares the same memory space with other threads in the same process.Processes are more isolated,have higher resource overhead,and require inter-process communication,while threads share resources,have lower overhead,and communicate through shared memory.The choice between processes and threads depends on the specific requirements of the application.

Q4. What is JDBC?

A)
JDBC stands for Java Database connectivity.
It is a Java-based API(Application Programming Interface)that provides a standard interface for connecting and interacting with relational databases.JDBC enables Java applications to interact with databases,execute SQL queries,and manage database connections.It provides a set of classes and interfaces that form the foundation for database access in Java applications.

Key components and features of JDBC include:

1.Driver Manager:

The DriverManager class is a part of JDBC and is responsible for managing a list of database drivers.It helps in establishing a connection to a database by selecting an appropriate driver from the available list.

2.JDBC Drivers:

JDBC drivers are platform-specific implementations that enable java applications to communicate with different types of databases.There are four types of JDBC drivers:
Type 1(JDBC-ODBC bridge),Type 2(Native-API driver),Type 3 (Network Protocol driver),and Type 4(Thin driver).

3.Connection Interface:

The connection Interface represents a connection to a database.It provides methods for creating statements,committing transactions,and managing the connection properties.

4.Statement Interface:

The statement Interface is used for executing SQL queries and updates.There are three types of statements:
Statement (for simple SQL statements),
Prepared Statements(for precompiled SQLstatements with input parameters)
And CallableStatements(for calling Stored procedures).

5.Resultset Interface:

The resultset interface represents the result set of a query.It provides methods to retrieve and manipulate data retrieved from the database.

6.SQLException:

SQL Exception is an exception class in JDBC that handles database-related exceptions.It provides information about errors that occur during database operations.

7.Datasource Interface:

The Datasource Interface is an alternative to the DriverManager for establishing database connections.It provides a more flexible and efficient way to manage database connections,especially in enterprise applications.

Here's a simple example of JDBC code to establish a connection ,execute a query ,and retrieve data from a database.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JDBCExample {
  public static void main(String[] args) {
    try {
      // Step 1: Load the JDBC driver
      Class.forName("com.mysql.cj.jdbc.Driver");

      // Step 2: Establish a connection to the database
      String url = "jdbc:mysql://localhost:3306/mydatabase";
```

```
            String username = "user";
            String password = "password";
            Connection connection = DriverManager.getConnection(url, username, password);

            // Step 3: Create a statement
            Statement statement = connection.createStatement();

            // Step 4: Execute a query
            String query = "SELECT * FROM mytable";
            ResultSet resultSet = statement.executeQuery(query);

            // Step 5: Process the result set
            while (resultSet.next()) {
                // Retrieve data from the result set
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                System.out.println("ID: " + id + ", Name: " + name);
            }

            // Step 6: Close resources
            resultSet.close();
            statement.close();
            connection.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

This example demonstrates the basic steps involved in using JDBC to connect to a database ,execute a query,and process the resultset.Note that error handling and resource management are crucial aspects of real world JDBC applications.

Q5. What is the Collection framework in Java?

A)The collection framework in java provides a set of classes and interfaces for managing and manipulating groups of objects.It forms a unified architecture for representing and manipulating collection of objects,making it easier to work with data structures like lists,sets,maps and queues.The primary goal of the Collection framework is to provide a standardized way to handle collections of objects,promoting code reuse,readability and efficiency.

Key components of the Java Collection framework include:

1.Interfaces:

- **Collection:**The root interface of the Collection framework.It defines the basic methods that all collection classes must support ,such as ‘add’,’remove’,’size’,’isEmpty’,and more.

- **List**:Represents an ordered collection of elements where elements can be accessed by their index.Implementations include ArrayList,LinkedList,and Vector.

- **Set**:Represents an unordered collection of unique elements,Implementations include HashSet,TreeSet,and LinkedHashSet.

- **Queue**:Extends the collection interface and represents a collection designed for holding elements prior to processing.Implementations include linkedList,PriorityQueue, and others.

- **Map**:Represents a collection of key-value pairs.Implementations include HashMap,TreeMap,LinkedHashMap,and others.

2.Classes:

- **ArrayList**:Implements the List interface and uses a dynamic array to store elements.It allows fast random access to elements.

- **LinkedList**:Implements the List interface and uses a doubly - linked list to store elements.It provides efficient insertion and deletion operations.

- **HashSet**:Implements the Set interface and uses a hash table to store unique elements.It does not guarantee the order of elements.

- **TreeSet**:Implements the Set interface and stores elements in a sorted tree structure.Elements are ordered based on their natural ordering or a specified comparator.

- **HashMap**:Implements the Map interface and uses a hash table to store key-value pairs.it provides fast retrieval and insertion of key-value pairs.

- **TreeMap**:Implements the Map interface and stores key-value pairs in a sorted tree structure.Keys are ordered based on their natural ordering or a specified comparator.

3.Utilities:

- Collections:A utility class providing static methods to operate on collections,such as sorting,shuffling,searching , and more.

4.Iterator:

- The Iterator interface provides a way to iterate over the elements of a collection.All collection classes implement this interface,allowing for a consistent way to traverse through elements.

The Collection framework simplifies the process of working with collections,providing a standard set of interfaces and classes that can be used interchangeably.It plays a crucial role in java programming,facilitating the creation,manipulation,and iteration of collections in a consistent and efficient manner.