21UCU403 - OBJECT ORIENTED PROGRAMMING

UNIT - 4

Java GUI, File Stream and Concurrency

ANSWER KEY

Created by:
Ajo. S. S
I B.Sc CS ‘A’

Section - 1

1. Give a note on GUI development.

Answer:

- Graphical User Interface (GUI) development is the process of creating interactive user interfaces for software applications.
- GUIs are visual representations of an application's functionality, and they allow users to interact
- Graphical User Interfaces (GUIs) are an essential aspect of software development, as they allow users to interact with the software in an intuitive and visually appealing manner.
- Java provides several libraries for developing GUI applications, including AWT, Swing, and JavaFX. Swing is a popular GUI toolkit for Java.
- Swing provides a comprehensive set of GUI components, including buttons, labels, text fields, checkboxes, radio buttons, and more.
- These components can be arranged in a variety of layouts, such as flow layout, grid layout, border layout, and box layout, to create a visually pleasing and functional GUI.
- To create a GUI application in Java, you typically start by creating a JFrame, which is a top-level container that holds all the other components in the GUI.
- You can then add components to the JFrame using various layout managers, set their properties, and attach event listeners to handle user actions.

2. List out SWING concept with an example.

Answer:

SWING is a Java user interface (UI) toolkit that provides a set of GUI (Graphical User Interface) components, such as buttons, labels, text fields, and many more, that can be used to build desktop applications.

Some of the key concepts in SWING include:

Components: SWING provides a wide range of components that can be used to build desktop applications. These components are objects that are added to a container, such as a JFrame or JPanel.

Containers: Containers are objects that hold other components. In SWING, JFrame, JDialog, and JPanel are examples of containers.

Layout Managers: Layout managers are used to determine the position and size of components within a container. Some of the common layout managers in SWING include BorderLayout, GridLayout, and FlowLayout.

Event Handling: SWING provides an event-driven programming model. Events are generated by user actions, such as clicking a button or typing text, and are handled by event listeners.

Example:

```
import javax.swing.*;
public class SwingExample {
    public static void main(String[] args) {
        // Create a JFrame
        JFrame frame = new JFrame("My Frame");
        // Create a button
        JButton button = new JButton("Click Me");
        // Add the button to the JFrame
        frame.add(button);
        // Set the size of the JFrame
        frame.setSize(300, 200);
        // Set the layout manager for the JFrame
        frame.setLayout(new BorderLayout());
        // Set the JFrame to be visible
        frame.setVisible(true);
    }
}
```

In this example, we create a JFrame and a JButton. We add the button to the JFrame and set the layout manager for the JFrame to BorderLayout. We then set the size of the JFrame and make it visible. When the user clicks the button, a "click" event will be generated and can be handled by an event listener.

3. Relate Generic Collections.

Answer:

- In Java, collections are used to group related objects together in a single unit.
- Collections can be thought of as containers for holding objects of different types.
- Generic collections in Java are collections that can hold objects of any type, as long as they are of the same type as the collection itself.
- The main advantage of using generic collections in Java is that they offer type safety at compile-time.
- This means that the compiler can check that the correct types are being used when objects are added or retrieved from the collection.
- This helps to prevent type errors and improves the overall reliability of the code.

Some of the commonly used generic collections in Java include:

ArrayList: This is an ordered collection that allows elements to be added and removed from the list. It uses an array to store the elements and is resizable.

LinkedList: This is an ordered collection that allows elements to be added and removed from the list. It uses a linked list to store the elements and is efficient when adding or removing elements from the beginning or end of the list.

HashSet: This is an unordered collection that does not allow duplicates. It uses a hash table to store the elements.

HashMap: This is an unordered collection that allows key-value pairs to be stored. It uses a hash table to store the key-value pairs and is efficient for retrieving values by their keys.

These are just a few examples of the many generic collections available in Java. By using generic collections, Java developers can write more reliable and type-safe code while taking advantage of the flexibility and convenience offered by collections.

4. Compare I/O Streams and Object serialisation.

Answer:

In Java, I/O Streams and Object Serialization are two ways to read and write data from and to external sources, but they have different purposes and use cases.

- I/O Streams refer to the input/output streams used to read and write data from and to various sources, such as files, network sockets, and standard input/output.
- I/O Streams are used to transfer raw bytes or characters and are mainly used for low-level communication with external sources.
- In Java, there are two types of I/O Streams - byte streams and character streams.
- On the other hand, Object Serialization is a mechanism used to convert an object into a stream of bytes so that it can be stored or transferred over a network.
- Object Serialization is used to save the state of an object and recreate it later in the same or different program.
- Object Serialization allows objects to be stored in a persistent state, which can be retrieved later.
- In summary, I/O Streams are used for low-level communication with external sources, while Object Serialization is used for storing and transferring objects between programs or over a network.
- Both I/O Streams and Object Serialization are essential tools in Java programming, but they serve different purposes.

5. Draw an outline of Thread Synchronisation.

Answer:

Thread synchronization is a way to ensure that multiple threads of execution do not interfere with each other while accessing shared resources in a Java program. It is used to avoid race conditions, deadlocks, and other concurrency-related problems.

Here is an outline of thread synchronization in Java:

Shared Resources: Threads can access shared resources such as variables, objects, and files. These resources need to be protected from concurrent access by multiple threads.

Race Condition: If multiple threads access a shared resource simultaneously, they may interfere with each other's access and result in a race condition. The outcome of a race condition is unpredictable and can lead to incorrect results.

Critical Section: A critical section is a section of code that accesses a shared resource. It needs to be executed atomically, meaning that it should not be interrupted by any other thread. This can be achieved using synchronization.

Synchronized Block: In Java, synchronization can be achieved using the synchronized keyword. A synchronized block is a block of code that is executed atomically by only one thread at a time.

Monitor: A monitor is an object that is used to provide synchronization in Java. Each Java object has a monitor associated with it, which is used to synchronize access to the object's methods and variables.

Locking: When a thread enters a synchronized block, it acquires a lock on the monitor associated with the object. This lock prevents other threads from entering the synchronized block until the lock is released.

Wait and Notify: Threads can also communicate with each other using the wait and notify methods. A thread can wait for another thread to release a lock using the wait method, and it can notify another thread to wake up using the notify method.

Deadlock: Deadlock is a situation where two or more threads are waiting for each other to release locks, and none of them can proceed. It is a common problem in thread synchronization and can be avoided by careful design of the synchronization code.

Overall, thread synchronization is an essential concept in Java programming, and it ensures that threads can access shared resources in a safe and predictable manner.

6. Give a note on Java Networking.

Answer:

Java is a popular programming language that provides extensive support for networking. It provides a rich set of libraries and APIs that enable developers to create networked applications easily.

Some key features of Java Networking include:

Socket programming: Java provides a powerful and flexible API for creating network sockets. This allows developers to establish connections between different systems and transfer data over the network.

URL handling: Java provides built-in support for handling URLs, making it easy to work with web-based resources such as HTML pages, images, and other multimedia files.

RMI (Remote Method Invocation): Java also provides support for RMI, which enables distributed computing. RMI allows Java objects to invoke methods on objects residing on other systems, providing a seamless way to build distributed applications.

Java NIO (New I/O): Java NIO provides a more efficient way of performing I/O operations, allowing for faster and more scalable network applications.

Java Networking APIs: Java provides a number of networking APIs, including the Java Networking API, JavaMail API, and Java Naming and Directory Interface (JNDI), which make it easy to work with different network protocols and services.

Overall, Java Networking is a powerful and flexible tool for creating networked applications of all types, from simple client-server applications to large-scale distributed systems.

Section - 2

1. Explain Thread States and Life Cycles.

Answer:

In Java, threads are objects that represent a single flow of execution within a program. Threads can be in different states at different times, depending on what they are doing.

The following are the different thread states in Java:

New: A thread is in the "new" state when it has been created but has not yet started running.

Runnable: A thread is in the "runnable" state when it is ready to run but is waiting for the JVM to allocate processor time. In this state, the thread can be executing or waiting for its turn to execute.

Blocked: A thread is in the "blocked" state when it is waiting for a lock to be released. A thread can enter the blocked state if it tries to access a resource that is already locked by another thread.

Waiting: A thread is in the "waiting" state when it is waiting for a specific condition to occur. For example, a thread might be waiting for input from the user or waiting for another thread to complete its task.

Timed Waiting: A thread is in the "timed waiting" state when it is waiting for a specific period of time. This can occur, for example, when a thread is sleeping or waiting for a time-out to occur.

Terminated: A thread is in the "terminated" state when it has completed its task and has either exited or been stopped.

The life cycle of a thread in Java is as follows:

New: When a thread is created, it is in the "new" state.

Runnable: When the start() method is called on a thread object, it enters the "runnable" state.

Running: When the thread scheduler selects the thread to run, it enters the "running" state.

Blocked, Waiting or Timed Waiting: A running thread can enter the blocked, waiting or timed waiting state depending on the conditions it encounters while executing.

Terminated: When a thread has completed its task or has been stopped, it enters the "terminated" state.

Java provides various methods to change the state of a thread, for example, sleep(), join(), wait() and notify(). It is important to manage thread states properly to ensure efficient use of resources and avoid issues like deadlocks and race conditions.

2. Discuss on Thread synchronisation.

Answer:

- Thread synchronization is an important concept in multi-threaded programming, particularly in Java.
- It is the process of managing the access of multiple threads to shared resources, such as variables, objects, and files, in order to prevent errors and maintain consistency.
- Thread synchronization is a way to ensure that multiple threads of execution do not interfere with each other while accessing shared resources in a Java program.
- It is used to avoid race conditions, deadlocks, and other concurrency-related problems.

Here is an outline of thread synchronization in Java:

Shared Resources: Threads can access shared resources such as variables, objects, and files. These resources need to be protected from concurrent access by multiple threads.

Race Condition: If multiple threads access a shared resource simultaneously, they may interfere with each other's access and result in a race condition. The outcome of a race condition is unpredictable and can lead to incorrect results.

Critical Section: A critical section is a section of code that accesses a shared resource. It needs to be executed atomically, meaning that it should not be interrupted by any other thread. This can be achieved using synchronization.

Synchronized Block: In Java, synchronization can be achieved using the synchronized keyword. A synchronized block is a block of code that is executed atomically by only one thread at a time.

Monitor: A monitor is an object that is used to provide synchronization in Java. Each Java object has a monitor associated with it, which is used to synchronize access to the object's methods and variables.

Locking: When a thread enters a synchronized block, it acquires a lock on the monitor associated with the object. This lock prevents other threads from entering the synchronized block until the lock is released.

Wait and Notify: Threads can also communicate with each other using the wait and notify methods. A thread can wait for another thread to release a lock using the wait method, and it can notify another thread to wake up using the notify method.

Deadlock: Deadlock is a situation where two or more threads are waiting for each other to release locks, and none of them can proceed. It is a common problem in thread synchronization and can be avoided by careful design of the synchronization code.

Overall, thread synchronization is critical for ensuring the correct and consistent behavior of multi-threaded Java applications. Careful consideration should be given to the appropriate synchronization mechanism for a given scenario to ensure optimal performance and correctness.

3. Show the use of Java Networking.

Answer:

Java Networking refers to the set of libraries and APIs available in the Java programming language that allow for the development of networked applications.

Here are some common uses of Java Networking:

Developing client-server applications

- Java Networking is often used for developing client-server applications where multiple clients connect to a server to access a shared resource or service.
- For example, an online chat application that allows multiple users to connect to a server to chat with each other.

Implementing network protocols

- Java Networking can be used to implement various network protocols such as HTTP, FTP, SMTP, and TCP/IP.
- These protocols allow for communication between devices on a network.

Remote method invocation (RMI)

- Java Networking can also be used for remote method invocation (RMI). RMI allows a Java program to invoke methods on remote objects running on different JVMs (Java Virtual Machines).
- This is commonly used in distributed computing and enterprise applications.

Web services

- Java Networking can be used for developing web services. Web services allow different applications to communicate with each other over a network using standard web protocols like SOAP and REST.

Networking monitoring and management

- Java Networking also provides APIs for monitoring and managing network connections, protocols, and devices.
- This can be useful for network administrators to monitor network performance and troubleshoot network issues.

Overall, Java Networking is a powerful tool that can be used to develop a wide variety of networked applications, from simple client-server applications to complex distributed systems.

4. List out the steps for Java GUI.

Answer:

Steps for creating a Java GUI:

Import the necessary packages

The first step is to import the required packages for creating a Java GUI. Some of the commonly used packages for Java GUI development are javax.swing, java.awt, and java.awt.event.

Create a frame

The next step is to create a frame that will serve as the container for the GUI components. You can create a frame using the JFrame class.

Set the size and layout of the frame

After creating the frame, you need to set its size and layout. You can set the size of the frame using the setSize() method, and you can set the layout using the setLayout() method.

Create GUI components

The next step is to create the GUI components that will be added to the frame. Some of the commonly used GUI components are JButton, JLabel, JTextField, JCheckBox, and JRadioButton.

Add the GUI components to the frame

After creating the GUI components, you need to add them to the frame. You can add the GUI components to the frame using the add() method.

Set the properties of the GUI components

After adding the GUI components to the frame, you can set their properties such as their size, location, and text using the appropriate methods.

Add event handlers

- If you want your GUI components to respond to user interactions such as button clicks, you need to add event handlers to them.
- You can add event handlers using the addActionListener() method.

Display the frame

Finally, you need to display the frame on the screen using the setVisible() method.

By following these steps, you can create a basic Java GUI.

5. Show how to make GUI Development using SWING.

Answer:

SWING is a Java-based GUI toolkit that provides a set of graphical user interface components for building desktop applications.

Here are the basic steps to create a simple GUI application using SWING:

1. Create a new Java project in your favorite IDE (Integrated Development Environment) such as Eclipse or NetBeans.

2. Import the SWING library by adding the following import statement at the top of your Java file:

```
  import javax.swing.*;
```

3. Create a new JFrame object to hold your GUI components. The JFrame is the main window of your application:

```
  JFrame frame = new JFrame("My Application");
```

4. Add the GUI components to the JFrame. Here is an example of adding a label and a button:

```
  JLabel label = new JLabel("Hello, world!");
  JButton button = new JButton("Click me!");
  frame.add(label);
  frame.add(button);
```

5. Specify the layout manager for the JFrame. The layout manager is responsible for positioning and sizing the components within the JFrame:

```
 frame.setLayout(new FlowLayout());
```

6. Set the properties of the JFrame, such as its size and visibility:

```
  frame.setSize(300, 200);
  frame.setVisible(true);
```

7. Finally, add an event listener to the button to handle user input. Here is an example of adding an ActionListener to the button:

```
  button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
          label.setText("Button clicked!");
      }
  });
```

This will change the text of the label to "Button clicked!" when the user clicks the button.

That's it! This is a basic example of how to create a GUI application using SWING. There are many more components and features available in SWING that you can explore to create more complex GUI applications.

6. Explain in detail about I/O Streams in Java.

Answer:

In Java, I/O streams are used to transfer data between a program and its external environment, such as files, network connections, and user input/output. Java provides two types of I/O streams: character streams (text) and byte streams (binary).

Character streams use Unicode character encoding, which is a standard for representing text in most of the world's writing systems. Byte streams are used for reading and writing binary data, such as images, audio, and video files.

Here's the basic syntax for creating an input stream and reading data from it:

```
import java.io.*;

public class Example {
  public static void main(String args[]) throws IOException {
    // Create an input stream from a file
    FileInputStream inputStream = new FileInputStream("input.txt");

    // Read data from the stream
    int data = inputStream.read();

    // Loop until the end of the stream is reached
    while(data != -1) {
      // Do something with the data
      System.out.print((char)data);

      // Read the next byte
      data = inputStream.read();
    }

    // Close the stream
    inputStream.close();
  }
}
```

Here's the basic syntax for creating an output stream and writing data to it:

```
import java.io.*;

public class Example {
  public static void main(String args[]) throws IOException {
    // Create an output stream to a file
    FileOutputStream outputStream = new FileOutputStream("output.txt");

    // Write data to the stream
    String message = "Hello, world!";
    byte[] messageBytes = message.getBytes();
    outputStream.write(messageBytes);
```

```
        // Close the stream
        outputStream.close();
    }
}
```

Note that both input and output streams must be closed after use to prevent resource leaks and ensure that any buffered data is flushed to the underlying stream. The IOException is thrown when an I/O error occurs, so it's important to handle it appropriately.

Section - 3

1. Construct the GUI Development using SWING.

Answer:

SWING is a Java GUI toolkit used for developing desktop applications with a graphical user interface. Here are some steps to develop a GUI application using SWING:

1. Create a new Java project in your preferred IDE (e.g., Eclipse, IntelliJ IDEA).
2. Create a new Java class that extends the JFrame class to represent the main window of your application.
3. Add Swing components to your JFrame using its content pane. Some common Swing components include JLabel, JButton, JTextArea, JList, and JTable.
4. Use layout managers to arrange the components on the JFrame. Layout managers help ensure that the components are arranged correctly regardless of the screen resolution and other factors.
5. Add event listeners to the components to handle user interactions. For example, you can add an ActionListener to a JButton to handle clicks.
6. Build and run your application to test it. You should see your GUI with the components arranged according to your layout manager and responding to user interactions.

Here's an example code snippet that creates a simple GUI application with a button:

```
import javax.swing.*;
public class MyFrame extends JFrame {
  private JButton myButton;
  public MyFrame() {
    setTitle("My GUI Application");
    setSize(400, 300);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    myButton = new JButton("Click me!");
    add(myButton);
    setVisible(true);
  }
  public static void main(String[] args) {
    MyFrame myFrame = new MyFrame();
  }
}
```

- In this example, we create a new class MyFrame that extends JFrame.
- In the constructor, we set the title, size, and close operation of the JFrame.
- We also create a new JButton and add it to the JFrame using the add() method.
- Finally, we make the JFrame visible by calling setVisible(true). When the user clicks the button, nothing happens because we haven't added an event listener yet.

To handle the button click, we can add an ActionListener to the JButton as follows:

```
myButton.addActionListener(e -> {
  JOptionPane.showMessageDialog(null, "Hello, world!");
});
```

This code creates a lambda expression that shows a simple message dialog when the button is clicked. When the user clicks the button, a message dialog with the text "Hello, world!" appears.

2. Apply the concepts of Thread States and Life Cycles.

Answer:

In Java, a thread is an independent execution path that can run concurrently with other threads. Each thread has a state and a life cycle. The state of a thread represents its current condition, while the life cycle represents the stages through which a thread passes.

The following are the different thread states in Java:

New: A thread is in the "new" state when it has been created but has not yet started running.

Runnable: A thread is in the "runnable" state when it is ready to run but is waiting for the JVM to allocate processor time. In this state, the thread can be executing or waiting for its turn to execute.

Blocked: A thread is in the "blocked" state when it is waiting for a lock to be released. A thread can enter the blocked state if it tries to access a resource that is already locked by another thread.

Waiting: A thread is in the "waiting" state when it is waiting for a specific condition to occur. For example, a thread might be waiting for input from the user or waiting for another thread to complete its task.

Timed Waiting: A thread is in the "timed waiting" state when it is waiting for a specific period of time. This can occur, for example, when a thread is sleeping or waiting for a time-out to occur.

Terminated: A thread is in the "terminated" state when it has completed its task and has either exited or been stopped.

The life cycle of a thread in Java is as follows:

New: When a thread is created, it is in the "new" state.

Runnable: When the start() method is called on a thread object, it enters the "runnable" state.

Running: When the thread scheduler selects the thread to run, it enters the "running" state.

Blocked, Waiting or Timed Waiting: A running thread can enter the blocked, waiting or timed waiting state depending on the conditions it encounters while executing.

Terminated: When a thread has completed its task or has been stopped, it enters the "terminated" state.

Example:

Here's an example that illustrates the life cycle and states of a thread in Java.

```
public class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running.");
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread t = new MyThread();  // NEW state
        t.start();  // RUNNABLE state
        try {
            Thread.sleep(1000);  // TIMED_WAITING state
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        synchronized (t) {
            try {
                t.wait();  // WAITING state
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        t.interrupt();  // TERMINATED state
    }
}
```

In this example, we create a new thread MyThread and start it using the start() method, putting it in the RUNNABLE state. We then put the thread in a

TIMED_WAITING state for 1 second using the sleep() method. After that, we put the thread in the WAITING state by calling the wait() method, synchronized on the thread itself. Finally, we interrupt the thread, causing it to enter the TERMINATED state.

Note that in practice, thread states can change rapidly and unpredictably, and it's often difficult to reason about them in a deterministic way. However, understanding the life cycle and states of threads is essential for writing correct and efficient multithreaded programs.

3. Explain the methods of Java Networking.

Answer:

Java provides a robust set of APIs and libraries to implement networking capabilities. The Java Networking API is designed to make it easy to develop networking applications in Java.

Some of the commonly used methods for networking in Java are:

Socket Programming

- Socket programming is a core concept in Java networking. It involves creating a socket and connecting it to a server to establish a communication channel.
- The java.net package provides classes to create sockets and work with TCP/IP network protocols.
- The classes in this package include Socket, ServerSocket, DatagramSocket, and MulticastSocket.

URL Handling

- The URL (Uniform Resource Locator) class in Java is used to handle the resources available on the internet.
- It provides a convenient way to create, parse and manipulate URLs.
- The URL class supports a wide range of protocols like HTTP, FTP, and SMTP.
- With the URL class, you can open a connection to a URL, read data from it, and perform other operations on it.

Networking using HttpURLConnection

- The HttpURLConnection class is a part of the java.net package and is used to establish a connection with the server and send an HTTP request.
- It provides a convenient way to perform HTTP operations such as GET, POST, PUT, DELETE, HEAD, and OPTIONS.
- The response from the server can be read using the getInputStream() or getErrorStream() methods.

DatagramPacket and DatagramSocket

- The DatagramPacket and DatagramSocket classes are used for UDP (User Datagram Protocol) communication.
- A DatagramPacket is used to send or receive a packet of data over the network.
- The DatagramSocket class is used to establish a UDP connection with the server.

RMI (Remote Method Invocation)

- RMI is a Java-based technology that allows Java objects to invoke methods on remote Java objects over the network.
- RMI uses Java Serialization to marshal and unmarshal objects between the client and server.
- RMI also provides features like object activation, distributed garbage collection, and dynamic class loading.

NIO (Non-blocking I/O)

- NIO is a new I/O model introduced in Java 1.4. It provides an alternative to the traditional blocking I/O model.
- NIO allows for asynchronous and non-blocking I/O operations, which can improve the performance of networking applications.
- NIO uses the java.nio package, which includes classes like Selector, Channel, and Buffer.

In summary, Java provides a rich set of networking APIs that can be used to implement various types of network applications. Depending on the specific requirements of the application, different methods can be used to achieve the desired functionality.