1. List methods of the String class and explain any two of them with examples.

Common Methods of the String Class:

1. length() - Returns the length of the string.
2. charAt(int index) - Returns the character at a specified index.
3. substring(int beginIndex, int endIndex) - Returns a substring.
4. equals(Object obj) - Compares two strings for equality.
5. equalsIgnoreCase(String anotherString) - Compares two strings, ignoring case.
6. compareTo(String anotherString) - Compares two strings lexicographically.
7. indexOf(String str) - Returns the index of the first occurrence of a substring.
8. toUpperCase() - Converts the string to uppercase.
9. toLowerCase() - Converts the string to lowercase.
10. trim() - Removes leading and trailing spaces.

Explanation of Two Methods:

1. **length() Method**:
   This method returns the number of characters in the string.
   Example:

```
2. public class Main {
3.     public static void main(String[] args) {
4.         String str = "Hello, World!";
5.         System.out.println("Length: " + str.length());
6.     }
7. }
```

Output:
Length: 13

8. **toUpperCase() Method**:
   This method converts all characters in a string to uppercase.
   Example:

```
9. public class Main {
10.     public static void main(String[] args) {
11.         String str = "hello";
12.         System.out.println("Uppercase: " + str.toUpperCase());
```

```
    }
}
```

Output:
Uppercase: HELLO

---

2. What is Exception Handling in Java? Explain with an example.

Explanation:

Exception handling in Java is a mechanism to handle runtime errors to maintain the normal flow of the program. It uses try, catch, finally, throw, and throws.

Example:

```
public class Main {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // This will throw an ArithmeticException
        } catch (ArithmeticException e) {
            System.out.println("Exception caught: " + e.getMessage());
        } finally {
            System.out.println("Finally block executed.");
        }
    }
}
```

Output:

Exception caught: / by zero

Finally block executed.

---

3. Differentiate between final, finally, and finalize.

Feature	final	finally	finalize
Definition	A keyword to declare constants, prevent inheritance, or prevent method overriding.	A block used to execute cleanup code after a try-catch.	A method in Object class for garbage collection.

Feature	final	finally	finalize
Usage	Applied to variables, methods, and classes.	Used in exception handling.	Called by JVM during garbage collection.
Example	final int MAX = 10;	finally { System.out.println("Done"); }	protected void finalize() { }

---

4. What is Run-time Polymorphism? Explain with an example.

Explanation:

Run-time polymorphism (dynamic method dispatch) occurs when a call to an overridden method is resolved at runtime. It uses method overriding in inheritance.

Example:

```
class Animal {
   void sound() {
      System.out.println("Animal makes a sound");
   }
}

class Dog extends Animal {
   @Override
   void sound() {
      System.out.println("Dog barks");
   }
}

public class Main {
   public static void main(String[] args) {
      Animal myAnimal = new Dog(); // Reference of parent, object of child
      myAnimal.sound(); // Calls Dog's method
   }
}
```

Output:
Dog barks

---

5. What is a package in Java? How can we create a subpackage and import it into another class?

Explanation:

A package in Java is a namespace that organizes classes and interfaces. Subpackages are created by nesting packages.

Creating a Package and Subpackage:

```
// File: mypackage/Parent.java

package mypackage;

public class Parent {
    public void display() {
        System.out.println("This is the parent class.");
    }
}


// File: mypackage/child/Child.java

package mypackage.child;

import mypackage.Parent;

public class Child extends Parent {
    public static void main(String[] args) {
        Child obj = new Child();
        obj.display();
    }
}
```

Running:

Compile and run from the root directory:

javac mypackage/Parent.java mypackage/child/Child.java

java mypackage.child.Child

Output:
This is the parent class.

Let's continue with the next set of questions:

---

6. What are Bitwise Operators in Java? Explain with examples.

Explanation:

Bitwise operators operate on individual bits of data. They are primarily used for low-level programming and manipulation of binary data.

Types of Bitwise Operators:

Operator	Description	Example
&	Bitwise AND	a & b
`	`	Bitwise OR
^	Bitwise XOR (exclusive OR)	a ^ b
~	Bitwise Complement (NOT)	~a
<<	Left Shift	a << 2
>>	Right Shift	a >> 2
>>>	Unsigned Right Shift	a >>> 2

Example:

```
public class Main {
   public static void main(String[] args) {
      int a = 5;  // Binary: 0101
      int b = 3;  // Binary: 0011

      System.out.println("a & b: " + (a & b));  // 0101 & 0011 = 0001 -> 1
      System.out.println("a | b: " + (a | b));  // 0101 | 0011 = 0111 -> 7
      System.out.println("a ^ b: " + (a ^ b));  // 0101 ^ 0011 = 0110 -> 6
      System.out.println("~a: " + (~a));       // ~0101 = 1010 (2's complement) -> -6
      System.out.println("a << 1: " + (a << 1));// 0101 << 1 = 1010 -> 10
      System.out.println("a >> 1: " + (a >> 1));// 0101 >> 1 = 0010 -> 2
   }
}
```

Output:

a & b: 1

a | b: 7

a ^ b: 6

~a: -6

a << 1: 10

a >> 1: 2

---

7. Differentiate between super and this keywords in Java with examples.

Feature	this	super
Definition	Refers to the current object.	Refers to the parent (superclass) object.
Usage	Used to access current class members.	Used to access superclass members.
Constructor	Can call another constructor in the same class.	Can call the superclass constructor.

Example:

```
class Parent {
   String name = "Parent";

   void display() {
      System.out.println("This is the parent class.");
   }
}

class Child extends Parent {
   String name = "Child";

   void showNames() {
      System.out.println("Child name: " + this.name);  // Refers to the current class's variable
      System.out.println("Parent name: " + super.name); // Refers to the superclass's variable
   }

   void display() {
```

```
        super.display();  // Calls the superclass method
        System.out.println("This is the child class.");
    }
}

public class Main {
    public static void main(String[] args) {
        Child obj = new Child();
        obj.showNames();
        obj.display();
    }
}
```

Output:

Child name: Child

Parent name: Parent

This is the parent class.

This is the child class.

---

8. What is Constructor Overloading in Java? Explain with an example.

Explanation:

Constructor overloading in Java allows a class to have multiple constructors with different parameter lists.

Example:

```
class Person {
    String name;
    int age;

    // Default Constructor
    Person() {
        this.name = "Unknown";
        this.age = 0;
```

```
    }

    // Parameterized Constructor
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}

public class Main {
    public static void main(String[] args) {
        Person p1 = new Person();          // Calls default constructor
        Person p2 = new Person("John", 25); // Calls parameterized constructor

        p1.display();
        p2.display();
    }
}
```

Output:

Name: Unknown, Age: 0

Name: John, Age: 25

---

9. Explain various methods of the String class in Java.

Methods of the String Class:

1. **length()**: Returns the length of the string.
2. **charAt(int index)**: Returns the character at the specified index.
3. **substring(int beginIndex)**: Returns a substring starting from beginIndex.

4. **contains(CharSequence s)**: Checks if the string contains the specified sequence.
5. **replace(char oldChar, char newChar)**: Replaces all occurrences of a character.
6. **concat(String str)**: Concatenates the specified string.
7. **split(String regex)**: Splits the string based on the provided regex.
8. **toUpperCase()**: Converts the string to uppercase.
9. **trim()**: Removes leading and trailing whitespace.

Example of split and replace:

```
public class Main {
   public static void main(String[] args) {
      String str = "Hello, World!";

      // Using replace
      System.out.println(str.replace("World", "Java"));

      // Using split
      String[] words = str.split(", ");
      for (String word : words) {
         System.out.println(word);
      }
   }
}
```

Output:

```
Hello, Java!
Hello
World!
```

---

10. What is the significance of the throws keyword in Java? Explain with an example.

Explanation:

The throws keyword in Java is used to declare exceptions a method might throw. It helps the caller of the method handle the exceptions.

Example:

```
class Main {
   // Method that declares an exception
   static void divide(int a, int b) throws ArithmeticException {
      if (b == 0) {
         throw new ArithmeticException("Division by zero");
      }
      System.out.println("Result: " + (a / b));
   }

   public static void main(String[] args) {
      try {
         divide(10, 0); // This will throw an exception
      } catch (ArithmeticException e) {
         System.out.println("Exception caught: " + e.getMessage());
      }
   }
}
```

Output:

Exception caught: Division by zero

Let's continue with the next set of questions:

---

11. Explain Shift Operators in Java with examples.

Explanation:

Shift operators in Java are used to shift the bits of a number left or right. There are three types:

Operator	Name	Description
<<	Left Shift	Shifts bits to the left, filling with 0s.
>>	Right Shift	Shifts bits to the right, retaining the sign.
>>>	Unsigned Right Shift	Shifts bits to the right, filling with 0s.

Example:

```
public class Main {
   public static void main(String[] args) {
      int a = 8;  // Binary: 00001000


      System.out.println("Left Shift (a << 2): " + (a << 2));  // 00100000 = 32
      System.out.println("Right Shift (a >> 2): " + (a >> 2)); // 00000010 = 2
      System.out.println("Unsigned Right Shift (a >>> 2): " + (a >>> 2)); // Same as >> for positive numbers
   }
}
```

Output:

Left Shift (a << 2): 32

Right Shift (a >> 2): 2

Unsigned Right Shift (a >>> 2): 2

---

12. What is an Applet? Explain the graphics methods available in the Applet class.

Explanation:

An Applet is a small Java program that runs within a web browser or an applet viewer. Applets are part of the java.applet package and use the javax.swing.JApplet or java.applet.Applet class.

Graphics Methods in Applet:

1. **drawString(String str, int x, int y)**: Draws a string at specified coordinates.
2. **drawLine(int x1, int y1, int x2, int y2)**: Draws a line between two points.
3. **drawRect(int x, int y, int width, int height)**: Draws a rectangle.
4. **fillRect(int x, int y, int width, int height)**: Draws a filled rectangle.
5. **drawOval(int x, int y, int width, int height)**: Draws an oval.
6. **fillOval(int x, int y, int width, int height)**: Draws a filled oval.

Example:

```
import java.applet.Applet;
import java.awt.Graphics;


public class MyApplet extends Applet {
   public void paint(Graphics g) {
```

```
        g.drawString("Hello, Applet!", 50, 50);
        g.drawRect(100, 100, 50, 50);
        g.fillOval(200, 100, 50, 50);
    }
}
```

To run the applet, use an applet viewer or embed it in HTML:

```
<applet code="MyApplet.class" width="300" height="300"></applet>
```

---

13. What are static methods, static variables, and static blocks in Java? Explain with examples.

Explanation:

1. **Static Method**: A method that belongs to the class rather than an instance of the class.
    - Called using the class name.
    - Cannot access non-static variables directly.
2. **Static Variable**: A variable that belongs to the class, shared by all objects.
    - Retains its value across all instances.
3. **Static Block**: A block of code that executes when the class is loaded.
    - Used to initialize static variables.

Example:

```
public class Main {
    static int count; // Static variable

    static {
        count = 10; // Static block initializes static variable
        System.out.println("Static block executed.");
    }

    static void displayCount() { // Static method
        System.out.println("Count: " + count);
    }

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

```
        displayCount(); // Calling the static method
    }
}
```

Output:

Static block executed.

Count: 10

---

14. Differentiate between method overloading and method overriding in Java with examples.

Feature	Method Overloading	Method Overriding
Definition	Same method name with different parameters.	Subclass provides a specific implementation of a superclass method.
Compile/Runtime	Resolved at compile-time.	Resolved at runtime.
Return Type	Can vary.	Must be the same or covariant.
Inheritance	Not required.	Requires inheritance.

Example of Overloading:

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println(calc.add(10, 20));
        System.out.println(calc.add(10.5, 20.5));
    }
```

```
}
```

Example of Overriding:

```
class Animal {
   void sound() {
      System.out.println("Animal makes a sound.");
   }
}

class Dog extends Animal {
   @Override
   void sound() {
      System.out.println("Dog barks.");
   }
}

public class Main {
   public static void main(String[] args) {
      Animal myDog = new Dog();
      myDog.sound(); // Calls Dog's overridden method
   }
}
```

Output:

Dog barks.

---

15. Explain any four methods of the StringBuffer class with examples.

Methods of StringBuffer:

1. **append(String str)**: Adds the specified string to the existing string buffer.
2. **insert(int offset, String str)**: Inserts a string at the specified position.
3. **replace(int start, int end, String str)**: Replaces a part of the string.
4. **reverse()**: Reverses the content of the string buffer.

Example:

```
public class Main {
   public static void main(String[] args) {
      StringBuffer sb = new StringBuffer("Hello");

      // Append
      sb.append(" World");
      System.out.println("After append: " + sb);

      // Insert
      sb.insert(6, "Java ");
      System.out.println("After insert: " + sb);

      // Replace
      sb.replace(6, 10, "Program");
      System.out.println("After replace: " + sb);

      // Reverse
      sb.reverse();
      System.out.println("After reverse: " + sb);
   }
}
```

Output:

After append: Hello World

After insert: Hello Java World

After replace: Hello Program World

After reverse: dlroW margorP olleH

Let's proceed with the next set of answers.

---

16. Write a short note on the Thread Life Cycle in Java.

Explanation:

The thread life cycle represents the various states a thread undergoes during its execution in Java. These are:

State	Description
New	The thread is created but not started yet (new Thread()).
Runnable	The thread is ready to run but waiting for CPU time (start()).
Running	The thread is actively running on the CPU.
Blocked/Waiting	The thread is waiting for resources or a signal to resume execution (e.g., waiting for I/O).
Terminated	The thread finishes its execution and moves to the terminated state.

Example:

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

public class Main {
   public static void main(String[] args) {
      Thread t1 = new MyThread(); // New
      t1.start();              // Runnable
   }
}
```

---

17. Explain the concept of deadlock in Java with examples.

Explanation:

A **deadlock** occurs when two or more threads block each other indefinitely while waiting for a resource.

Example:

```
class Resource {
   synchronized void method1(Resource r2) {
      System.out.println(Thread.currentThread().getName() + " is executing method1");
```

```
        try { Thread.sleep(100); } catch (InterruptedException e) {}
        r2.method2();
    }

    synchronized void method2() {
        System.out.println(Thread.currentThread().getName() + " is executing method2");
    }
}

public class DeadlockExample {
    public static void main(String[] args) {
        Resource r1 = new Resource();
        Resource r2 = new Resource();

        Thread t1 = new Thread(() -> r1.method1(r2), "Thread-1");
        Thread t2 = new Thread(() -> r2.method1(r1), "Thread-2");

        t1.start();
        t2.start();
    }
}
```

Output: The threads block each other, causing a deadlock.

---

18. Compare the features of Java and C++.

Feature	Java	C++
Platform Independence	Platform-independent (via JVM).	Platform-dependent.
Memory Management	Automatic garbage collection.	Manual memory management using pointers.
Multiple Inheritance	Not supported directly, achieved through interfaces.	Supported with classes.

Feature	Java	C++
Pointers	Not exposed directly.	Fully supported.
Compiler/Interpreter	Compiled into bytecode and interpreted by JVM.	Compiled into machine code.
Ease of Use	Simple and beginner-friendly.	Complex due to features like pointers.
Security	More secure (no explicit pointers).	Less secure (exposes low-level details).

---

19. What is inheritance in Java? Explain the use of the super keyword in inheritance.

Explanation:

Inheritance is a mechanism where a class (child) inherits the properties and methods of another class (parent). It promotes code reuse and establishes a parent-child relationship.

Types of Inheritance:

1. Single
2. Multilevel
3. Hierarchical

super Keyword:

1. **Access Parent Class Constructor**:
    - super() calls the parent class constructor.
2. **Access Parent Class Methods**:
    - Calls methods of the parent class.
3. **Access Parent Class Variables**:
    - Accesses variables from the parent class.

Example:

```
class Parent {
  int num = 10;

  Parent() {
    System.out.println("Parent constructor");
  }
```

```
    void display() {
        System.out.println("Parent method");
    }
}

class Child extends Parent {
    int num = 20;

    Child() {
        super(); // Calls Parent constructor
        System.out.println("Child constructor");
    }

    void show() {
        System.out.println("Child num: " + num);
        System.out.println("Parent num: " + super.num); // Access Parent variable
        super.display(); // Call Parent method
    }
}

public class Main {
    public static void main(String[] args) {
        Child c = new Child();
        c.show();
    }
}
```

Output:

Parent constructor

Child constructor

Child num: 20

Parent num: 10

Parent method

---

20. What is Exception Handling in Java?

Explanation:

Exception Handling is a mechanism to handle runtime errors in Java, ensuring the application continues to run without abrupt termination.

Key Concepts:

1. **try**: Encapsulates the code that might throw an exception.
2. **catch**: Handles the exception.
3. **finally**: Executes code regardless of exception occurrence.
4. **throw**: Used to explicitly throw an exception.
5. **throws**: Declares exceptions in a method signature.

Example:

```
public class Main {
   public static void main(String[] args) {
      try {
         int result = 10 / 0;
      } catch (ArithmeticException e) {
         System.out.println("Cannot divide by zero.");
      } finally {
         System.out.println("Execution completed.");
      }
   }
}
```

Output:

Cannot divide by zero.

Execution completed.

Let's proceed with the next set of answers.

---

21. What are the uses of static and final keywords in Java?

static Keyword:

The static keyword can be used with variables, methods, blocks, and nested classes.

1. **Static Variable**: Shared among all instances of a class.
2. class Example {
3. static int count = 0; // Static variable
4. Example() {
5. count++;
6. }
7. }
8. public class Main {
9. public static void main(String[] args) {
10. Example e1 = new Example();
11. Example e2 = new Example();
12. System.out.println(Example.count); // Output: 2
13. }
14. }
15. **Static Method**: Can be called without creating an object of the class.
16. class Example {
17. static void display() {
18. System.out.println("Static method called.");
19. }
20. }
21. public class Main {
22. public static void main(String[] args) {
23. Example.display(); // Directly calling without creating an object.
24. }
25. }
26. **Static Block**: Executes once when the class is loaded into memory.
27. class Example {
28. static {
29. System.out.println("Static block executed.");

```
30.     }
31. }
```

---

final Keyword:

The final keyword is used to restrict changes.

1. **Final Variable**: Value cannot be changed after initialization.
2. final int MAX = 100;
3. **Final Method**: Cannot be overridden by subclasses.
4. class Parent {
5. final void show() {
6. System.out.println("Final method.");
7. }
8. }
9. **Final Class**: Cannot be inherited.
10. final class Example {}

---

22. What is the Java Interpreter?

The Java interpreter is part of the Java Runtime Environment (JRE) responsible for executing bytecode generated by the Java compiler.

1. **Function**: Translates bytecode into machine code at runtime.
2. **Components**:
    - **JVM**: Executes the bytecode.
    - **Just-In-Time (JIT) Compiler**: Optimizes frequently executed code.

---

23. Explain Thread Synchronization in Java with examples.

Explanation:

Thread synchronization ensures that shared resources are accessed by only one thread at a time to avoid data inconsistency.

Synchronized Methods:

Locks the entire method.

```
class Counter {
  private int count = 0;
```

```
    synchronized void increment() {
        count++;
    }

    int getCount() {
        return count;
    }
}
public class Main {
    public static void main(String[] args) {
        Counter counter = new Counter();
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) counter.increment();
        });
        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) counter.increment();
        });
        t1.start();
        t2.start();
        try { t1.join(); t2.join(); } catch (InterruptedException e) {}
        System.out.println("Final count: " + counter.getCount());
    }
}
```

---

24. Explain the constructors of the String class in Java.

Constructors:

1. **Empty Constructor**: Creates an empty string.
2. String str = new String(); // Empty string
3. **String from Literal**: Initializes the string with a given value.
4. String str = new String("Hello");

5. **String from Character Array**: Converts a character array to a string.
6. char[] arr = {'H', 'e', 'l', 'l', 'o'};
7. String str = new String(arr);
8. **String from Byte Array**: Converts a byte array to a string.
9. byte[] bytes = {72, 101, 108, 108, 111};
10. String str = new String(bytes);

---

25. Explain Bitwise Operators in Java with examples.

Explanation:

Bitwise operators perform operations on bits.

Operator	Description	Example (5 = 0101, 3 = 0011)
&	Bitwise AND	5 & 3 = 0001
`	`	Bitwise OR
^	Bitwise XOR	5 ^ 3 = 0110
~	Bitwise Complement	~5 = 1010
<<	Left Shift	5 << 1 = 1010
>>	Right Shift	5 >> 1 = 0010

Example:

```
public class Main {
   public static void main(String[] args) {
      int a = 5, b = 3;
      System.out.println("a & b: " + (a & b)); // 1
      System.out.println("a | b: " + (a | b)); // 7
      System.out.println("a ^ b: " + (a ^ b)); // 6
      System.out.println("~a: " + (~a));      // -6
      System.out.println("a << 1: " + (a << 1)); // 10
   }
}
```

Let's proceed with the next set of answers.

26. What is an Abstract Class in Java? Explain with an example.

Explanation:

An abstract class in Java is a class that cannot be instantiated. It can include abstract methods (methods without a body) as well as concrete methods.

Key Points:

1. Declared using the abstract keyword.
2. Can have constructors, fields, and non-abstract methods.
3. Used to provide a base class for other classes.

Example:

```
abstract class Animal {
   abstract void sound(); // Abstract method

   void sleep() { // Concrete method
      System.out.println("Sleeping...");
   }
}

class Dog extends Animal {
   void sound() {
      System.out.println("Barks");
   }
}

public class Main {
   public static void main(String[] args) {
      Animal myDog = new Dog(); // Allowed through upcasting
      myDog.sound(); // Output: Barks
      myDog.sleep(); // Output: Sleeping...
   }
}
```

---

27. What is Narrowing Conversion in Java? Explain with an example.

Explanation:

Narrowing conversion occurs when a larger data type is converted into a smaller data type. It may lead to data loss and requires explicit casting.

Example:

```
public class Main {
   public static void main(String[] args) {
      double largeValue = 100.04;
      int smallValue = (int) largeValue; // Explicit casting
      System.out.println("Double: " + largeValue); // Output: 100.04
      System.out.println("Int: " + smallValue);    // Output: 100
   }
}
```

---

28. What is Garbage Collection in Java?

Explanation:

Garbage collection is an automatic memory management feature in Java. The JVM removes objects that are no longer in use to free up memory.

Key Points:

1. **Automatic**: No need for explicit memory deallocation.
2. **Performed by JVM**: Uses a background thread.
3. **Methods to Request Garbage Collection**:
    - System.gc()
    - Runtime.getRuntime().gc()

Example:

```
public class Main {
   public static void main(String[] args) {
      Main obj = new Main();
      obj = null; // Eligible for garbage collection
      System.gc();
   }
```

```
    @Override
    protected void finalize() {
        System.out.println("Garbage collected.");
    }
}
```

---

29. Explain Arrays in Java with examples.

Explanation:

An array is a collection of elements of the same type stored in contiguous memory locations.

Syntax:

```
dataType[] arrayName = new dataType[size];
```

Example:

```
public class Main {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40}; // Declaration and initialization
        for (int i = 0; i < arr.length; i++) {
            System.out.println("Element at index " + i + ": " + arr[i]);
        }
    }
}
```

---

30. Explain the usage of static keywords in Java.

Explanation:

The static keyword can be used for methods, variables, blocks, and nested classes.

1. Static Variable:

- Shared among all objects of a class.

```
class Example {
    static int counter = 0;
}
```

2. Static Method:

- Called without creating an object.

```
static void show() {
   System.out.println("Static method.");
}
```

3. Static Block:

- Executes when the class is loaded.

```
static {
   System.out.println("Static block executed.");
}
```

---

31. Explain the Thread Life Cycle in detail.

Explanation:

The thread life cycle in Java consists of five stages:

1. **New**: Thread is created but not started.
2. **Runnable**: Thread is ready to run, waiting for CPU.
3. **Running**: Thread is executing.
4. **Waiting/Blocked**: Thread is waiting for resources or signals.
5. **Terminated**: Thread finishes execution.

Example:

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


public class Main {
   public static void main(String[] args) {
      Thread t1 = new MyThread();
      t1.start(); // Moves to Runnable and then Running
   }
}
```

32. Compare the features of Java and C++.

Feature	Java	C++
Platform Independence	Yes, via JVM.	No.
Memory Management	Automatic garbage collection.	Manual memory management.
Pointers	Not exposed directly.	Fully supported.
Multiple Inheritance	Achieved through interfaces.	Supported.
Ease of Use	Simplified syntax, no explicit pointers.	Complex due to pointers and templates.
Security	Highly secure.	Less secure.

33. What is a deadlock in Java? Explain in detail with examples.

Explanation:

A deadlock occurs when two threads are waiting for each other's resources indefinitely.

Example:

```
class Resource {
   synchronized void method1(Resource r2) {
      System.out.println(Thread.currentThread().getName() + " is executing method1");
      r2.method2();
   }

   synchronized void method2() {
      System.out.println(Thread.currentThread().getName() + " is executing method2");
   }
}

public class Main {
   public static void main(String[] args) {
      Resource r1 = new Resource();
      Resource r2 = new Resource();
```

```
        Thread t1 = new Thread(() -> r1.method1(r2), "Thread-1");
        Thread t2 = new Thread(() -> r2.method1(r1), "Thread-2");

        t1.start();
        t2.start();
    }
}
```

In this case, Thread-1 and Thread-2 will be stuck indefinitely.

---

34. What is an Exception in Java? Explain user-defined exceptions with an example.

Explanation:

An exception is an unexpected event that occurs during the execution of a program and disrupts its normal flow. In Java, exceptions are objects of the Throwable class.

Types of Exceptions:

1. **Checked Exceptions**: Exceptions checked at compile-time (e.g., IOException, SQLException).
2. **Unchecked Exceptions**: Exceptions that occur at runtime (e.g., ArithmeticException, NullPointerException).
3. **Errors**: Serious issues (e.g., OutOfMemoryError).

User-Defined Exceptions:

You can create custom exceptions by extending the Exception or RuntimeException class.

Example:

```
class CustomException extends Exception {
    CustomException(String message) {
        super(message);
    }
}

public class Main {
    public static void main(String[] args) {
        try {
            validateAge(15);
```

```
        } catch (CustomException e) {
            System.out.println("Caught Exception: " + e.getMessage());
        }
    }

    static void validateAge(int age) throws CustomException {
        if (age < 18) {
            throw new CustomException("Age must be 18 or above.");
        }
        System.out.println("Valid age!");
    }
}
```

---

35. What are the life cycle methods of an Applet? Explain in detail.

Explanation:

An Applet is a Java program that runs in a browser or Applet viewer. Its life cycle consists of four methods:

1. **init()**: Initializes the applet.
2. **start()**: Executes when the applet starts running.
3. **stop()**: Called when the applet is stopped or the user navigates away.
4. **destroy()**: Invoked when the applet is permanently removed from memory.

Example:

```
import java.applet.Applet;
import java.awt.Graphics;

public class MyApplet extends Applet {
    public void init() {
        System.out.println("Applet Initialized");
    }

    public void start() {
```

```
    System.out.println("Applet Started");
  }

  public void paint(Graphics g) {
    g.drawString("Welcome to Applet!", 50, 50);
  }

  public void stop() {
    System.out.println("Applet Stopped");
  }

  public void destroy() {
    System.out.println("Applet Destroyed");
  }
}
```

---

36. Explain Thread Synchronization in Java.

Explanation:

Thread synchronization ensures multiple threads do not access shared resources simultaneously, preventing data inconsistency.

Mechanisms:

1. **Synchronized Block**: Locks only a portion of code.
2. synchronized(object) {
3. // Critical section
4. }
5. **Synchronized Method**: Locks the entire method.
6. synchronized void display() {
7. // Critical section
8. }

Example:

class Table {

```
    synchronized void printTable(int n) {
        for (int i = 1; i <= 5; i++) {
            System.out.println(n * i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

class MyThread extends Thread {
    Table t;
    MyThread(Table t) {
        this.t = t;
    }

    public void run() {
        t.printTable(5);
    }
}

public class Main {
    public static void main(String[] args) {
        Table obj = new Table();
        MyThread t1 = new MyThread(obj);
        MyThread t2 = new MyThread(obj);
        t1.start();
        t2.start();
    }
```

}

---

37. Explain the StringBuffer class and its uses.

Explanation:

StringBuffer is a mutable sequence of characters used for efficient string manipulation.

Common Methods:

1. **append()**: Adds text to the string.
2. StringBuffer sb = new StringBuffer("Hello");
3. sb.append(" World");
4. System.out.println(sb); // Output: Hello World
5. **insert()**: Inserts text at a specified index.
6. sb.insert(6, "Java ");
7. System.out.println(sb); // Output: Hello Java World
8. **delete()**: Removes text between two indices.
9. sb.delete(6, 11);
10. System.out.println(sb); // Output: Hello World
11. **reverse()**: Reverses the sequence.
12. sb.reverse();
13. System.out.println(sb); // Output: dlroW olleH

---

38. Explain the structure of JVM in detail.

Explanation:

The Java Virtual Machine (JVM) is the runtime environment for executing Java programs. Its main components are:

1. **Class Loader**: Loads .class files into memory.
2. **Memory Areas**:
    - **Method Area**: Stores class structure, constants, and static variables.
    - **Heap**: Stores objects and instance variables.
    - **Stack**: Manages method invocations and local variables.
    - **PC Register**: Stores the address of the next instruction.
    - **Native Method Stack**: Handles native (non-Java) methods.

3. **Execution Engine**:
    - **Interpreter**: Executes bytecode instructions.
    - **JIT Compiler**: Converts bytecode into native machine code for better performance.
4. **Native Interface**: Connects Java code with native libraries.
5. **Garbage Collector**: Manages memory by removing unused objects.

---

39. Explain Bitwise Operators in Java with detailed examples.

Explanation:

Bitwise operators manipulate bits of variables.

Operators:

1. **& (AND)**: Sets bit if both are 1.
2. **| (OR)**: Sets bit if at least one is 1.
3. **^ (XOR)**: Sets bit if bits are different.
4. **~ (Complement)**: Inverts bits.
5. **<< (Left Shift)**: Shifts bits to the left.
6. **>> (Right Shift)**: Shifts bits to the right.

Example:

```
public class Main {
   public static void main(String[] args) {
      int a = 5; // 0101 in binary
      int b = 3; // 0011 in binary

      System.out.println("a & b: " + (a & b)); // Output: 1
      System.out.println("a | b: " + (a | b)); // Output: 7
      System.out.println("a ^ b: " + (a ^ b)); // Output: 6
      System.out.println("~a: " + (~a));      // Output: -6
      System.out.println("a << 1: " + (a << 1)); // Output: 10
   }
}
```

---

40. What is an Interface in Java? Explain with examples.

Explanation:

An **interface** in Java is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces cannot contain instance fields or constructors. They are used to specify a set of methods that a class must implement.

- **Key Points:**
  - An interface can be implemented by a class using the implements keyword.
  - A class can implement multiple interfaces (multiple inheritance).
  - Interface methods are abstract by default (they do not have a body).

Example:

```
interface Animal {
   void sound();  // Abstract method
}


class Dog implements Animal {
   public void sound() {
      System.out.println("Dog barks");
   }
}


public class Main {
   public static void main(String[] args) {
      Animal myDog = new Dog();
      myDog.sound();  // Calling the implemented method
   }
}
```

Output:

Dog barks

---

41. What is the super keyword in Java? Explain with examples.

Explanation:

The super keyword in Java refers to the immediate parent class of the current object. It is used to:

1. Call parent class methods.

2. Access parent class variables.
3. Invoke a parent class constructor.

Example:

```
class Animal {
   String name = "Animal";

   void sound() {
      System.out.println("Animal makes a sound");
   }
}

class Dog extends Animal {
   String name = "Dog";

   void displayNames() {
      System.out.println("Child Name: " + name);
      System.out.println("Parent Name: " + super.name);  // Accessing parent class variable
   }

   void sound() {
      super.sound();  // Calling parent class method
      System.out.println("Dog barks");
   }
}

public class Main {
   public static void main(String[] args) {
      Dog dog = new Dog();
      dog.displayNames();
      dog.sound();
   }
```

```
}
```

Output:

```
Child Name: Dog
Parent Name: Animal
Animal makes a sound
Dog barks
```

---

42. Explain arrays of objects in Java with suitable examples.

Explanation:

An **array of objects** in Java is an array where each element is a reference to an object. It can store multiple objects of the same class type.

Example:

```
class Student {
   String name;
   int rollNo;

   Student(String name, int rollNo) {
      this.name = name;
      this.rollNo = rollNo;
   }

   void display() {
      System.out.println("Name: " + name + ", Roll No: " + rollNo);
   }
}

public class Main {
   public static void main(String[] args) {
      // Array of Student objects
      Student[] students = new Student[3];
      students[0] = new Student("Alice", 1);
```

```
        students[1] = new Student("Bob", 2);
        students[2] = new Student("Charlie", 3);

        // Displaying information of each student
        for (Student student : students) {
            student.display();
        }
    }
}
```

Output:

Name: Alice, Roll No: 1

Name: Bob, Roll No: 2

Name: Charlie, Roll No: 3

---

43. What are the life cycle methods of an Applet?

Explanation:

The **life cycle** of an applet involves four primary methods that are called at different stages:

1. **init()**:
    - Called once when the applet is first loaded. Used for initialization.
2. **start()**:
    - Called each time the applet is started (or refreshed). Used for running the applet's code.
3. **stop()**:
    - Called when the applet is stopped (for example, when the user navigates away from the page). Used for halting any ongoing activities.
4. **destroy()**:
    - Called when the applet is destroyed. Used to clean up resources.

Example:

```
import java.applet.Applet;
import java.awt.Graphics;

public class MyApplet extends Applet {
```

```
    public void init() {
        System.out.println("Applet Initialized");
    }

    public void start() {
        System.out.println("Applet Started");
    }

    public void paint(Graphics g) {
        g.drawString("Hello Applet!", 50, 50);
    }

    public void stop() {
        System.out.println("Applet Stopped");
    }

    public void destroy() {
        System.out.println("Applet Destroyed");
    }
}
```

---

44. What are Access Modifiers in Java? Explain with examples.

Explanation:

Access modifiers in Java control the visibility or access level of classes, methods, and variables. The four main access modifiers are:

1. **public**: Accessible from anywhere.
2. **protected**: Accessible within the same package and by subclasses.
3. **Default (no modifier)**: Accessible only within the same package.
4. **private**: Accessible only within the same class.

Example:

```
class Demo {
```

```
    public int publicVar = 10;
    protected int protectedVar = 20;
    int defaultVar = 30;  // No modifier, package-private
    private int privateVar = 40;

    public void display() {
        System.out.println("Public: " + publicVar);
        System.out.println("Protected: " + protectedVar);
        System.out.println("Default: " + defaultVar);
        System.out.println("Private: " + privateVar);
    }
}

public class Main {
    public static void main(String[] args) {
        Demo obj = new Demo();
        obj.display();
        System.out.println("Accessing publicVar: " + obj.publicVar);
    }
}
```

---

45. What are the uses of the final keyword in Java?

Explanation:

The **final** keyword in Java has three primary uses:

1. **Final Variables**: Once assigned, their values cannot be changed.
2. **Final Methods**: These methods cannot be overridden by subclasses.
3. **Final Classes**: These classes cannot be subclassed.

Example:

```
final int MAX_VALUE = 100;  // Final variable

final class Animal {  // Final class
```

```
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {  // This will cause an error, as Animal is final
    void sound() {
        System.out.println("Dog barks");
    }
}
```

---

46. Explain Type Casting in Java with examples.

Explanation:

Type casting in Java is the process of converting one data type into another. There are two types:

1. **Implicit Casting (Widening)**: Automatically done by Java when a smaller type is assigned to a larger type.
2. **Explicit Casting (Narrowing)**: Requires manual intervention using parentheses to specify the target type.

Example:

```
// Implicit Casting (Widening)
int num = 100;
long l = num;  // int to long

// Explicit Casting (Narrowing)
double d = 10.5;
int i = (int) d;  // double to int, decimal part is lost

System.out.println("Implicit: " + l);  // Output: 100
System.out.println("Explicit: " + i);  // Output: 10
```

---

47. Explain the control statements available in Java with examples.

Explanation:

Control statements in Java are used to control the flow of execution. The main control statements are:

1. **Conditional Statements**: if, if-else, switch.
2. **Looping Statements**: for, while, do-while.
3. **Jump Statements**: break, continue, return.

Example:

```
// if-else
int age = 18;
if (age >= 18) {
   System.out.println("Adult");
} else {
   System.out.println("Minor");
}


// switch
int day = 3;
switch (day) {
   case 1: System.out.println("Monday"); break;
   case 2: System.out.println("Tuesday"); break;
   case 3: System.out.println("Wednesday"); break;
   default: System.out.println("Invalid day");
}


// for loop
for (int i = 0; i < 3; i++) {
   System.out.println("Iteration " + (i + 1));
}
```

---

48. What are the various ways to create a string in Java? Explain with examples.

Explanation:

Strings in Java can be created in multiple ways:

1. **Using String Literal**: Directly using double quotes.
2. **Using new Keyword**: Using the new String() constructor.

Example:

```
// Using String literal
String str1 = "Hello";

// Using new keyword
String str2 = new String("World");

System.out.println(str1);  // Output: Hello
System.out.println(str2);  // Output: World
```
