Core JAVA Interview Question

1. **What is Java?**
   Java is a high-level, object-oriented, platform-independent programming language developed by James Gosling and his team at Sun Microsystems (later acquired by Oracle Corporation). It was released in 1995 and is widely used for building various types of applications, including web, mobile, enterprise, and desktop applications.
2. **Briefly describe the history of Java.**
   The history of Java dates back to the early 1990s when James Gosling and his team at Sun Microsystems started working on a project called "Oak." The project was later renamed Java. Here are the key milestones in the history of Java:
   - 1991: The project "Oak" begins at Sun Microsystems.
   - 1995: Java 1.0 is released to the public.
   - 1997: Java 1.1 is released, introducing inner classes and JIT (Just-In-Time) compiler support.
   - 2004: Java 5 (Java 1.5) is released, introducing major updates like Generics, Annotations, and Enumerations.
   - 2011: Oracle Corporation acquires Sun Microsystems, becoming the official maintainer of Java.
   - 2014: Java 8 is released with significant updates, including Lambda expressions and Streams API.
   - 2017: Java 9 is released with the introduction of modules and JShell.
   - 2018: Java 10 and Java 11 are released with various enhancements.
   -
3. **What are the features of Java that make it a popular programming language?**
   Java offers several features that make it a popular programming language:
   - Platform Independence: Java code is compiled into platform-independent bytecode, which can run on any system with a compatible JVM.

- Object-Oriented: Java follows an object-oriented paradigm, allowing the use of classes and objects to structure code.
- Garbage Collection: Java automatically manages memory through garbage collection, freeing developers from manual memory management.
- Multi-threading: Java supports multi-threading, allowing developers to create concurrent applications.
- Security: Java provides a secure runtime environment with features like class loaders and a security manager.

4. **Compare C++ and Java in terms of their differences and similarities.**
   C++ and Java are both object-oriented programming languages, but they have some differences:
   - Memory Management: Java uses automatic garbage collection, while C++ requires manual memory management using pointers.
   - Platform Independence: Java is platform-independent, but C++ is platform-dependent.
   - Pointers: Java does not have pointers, whereas C++ supports them.
   - Virtual Functions: In Java, all functions are virtual by default, whereas in C++, virtual functions need to be explicitly defined.
5. **Write a simple "Hello Java" program that prints "Hello, World!" to the console.**
   A simple "Hello Java" program in Java:

```
javaCopy code
public class HelloJava {
    public static void main(String[] args) {
        System.out.println("Hello, Java!");
    }
}
```

6. **What is Program Internal?**
   The internal structure of a Java program is organized into classes and packages. A package is a group of related classes, and each Java file can contain one public

class, which shares the same name as the file. The main class with the `main` method is the entry point of the program.

7. **How to set the path for Java?**
   To set the path for Java in your system, follow these steps:
   - Find the location of your JDK (Java Development Kit) installation.
   - Set the `JAVA_HOME` environment variable to point to the JDK installation directory.
   - Append the `bin` directory of the JDK to the system's `PATH` environment variable.

8. **Differentiate between JDK, JRE, and JVM**
   - JDK (Java Development Kit): It is a software development kit that includes all the tools necessary for developing, debugging, and monitoring Java applications.
   - JRE (Java Runtime Environment): It is an environment that allows you to run Java applications. It includes the JVM and the necessary libraries and other files required to execute Java applications but does not contain development tools.
   - JVM (Java Virtual Machine): It is a virtual machine that executes Java byte code. It abstracts the hardware and operating system details, providing platform independence for Java programs.

9. **Describe the role and purpose of the Java Virtual Machine (JVM).**
   Answer: The JVM is a critical component of the Java platform. Its primary role is to execute Java byte code, which is generated after compiling Java source code. The JVM abstracts the underlying hardware and operating system, providing a consistent runtime environment for Java applications across different platforms. It also handles memory management, garbage collection, and ensures security by running Java programs in a controlled sandbox environment.

10. **What are variables in Java? How are they declared and used?**
    Answer: Variables in Java are containers used to store data. They are declared with a data type and can be assigned values that match their data type. Variables can be of primitive data types (e.g., int, double, char) or reference data types (e.g., objects).

```
javaCopy code
public class VariablesExample {
    public static void main(String[] args) {
        // Primitive data types
        int age = 30;
        double salary = 45000.75;
        char gender = 'M';

        // Reference data type
        String name = "John Doe";

        // Using variables
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Gender: " + gender);
        System.out.println("Salary: $" + salary);
    }
}
```

11. **Discuss the various data types available in Java with examples.**
    Answer: Java supports two categories of data types:

- Primitive Data Types: byte, short, int, long, float, double, char, boolean.
- Reference Data Types: objects, arrays, and classes.

```
javaCopy code
public class DataTypesExample {
    public static void main(String[] args) {
        // Primitive data types
        int age = 30;
        double salary = 45000.75;
        char gender = 'M';
        boolean isEmployed = true;

        // Reference data type
        String name = "John Doe";
        int[] numbers = {1, 2, 3, 4, 5};

        // Using data types
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Gender: " + gender);
        System.out.println("Salary: $" + salary);
        System.out.println("Employed: " + isEmployed);
        System.out.println("Numbers: " + Arrays.toString(numbers));
    }
```

```
}
```

12. **How does Java handle Unicode characters? Why is it important?**
    Answer: Java uses Unicode to handle character encoding, allowing it to support characters from various writing systems. Unicode is important as it ensures proper representation of characters from different languages and helps in creating internationalized applications that can work across different locales.
13. **Explain different types of operators in Java (arithmetic, logical, bitwise) with examples.**
    Java supports different types of operators:

- **Arithmetic Operators:** + (addition), - (subtraction), * (multiplication), / (division), % (modulus).
- **Logical Operators**: && (logical AND), || (logical OR), ! (logical NOT).
- **Bitwise Operators**: & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT).
- **Comparison Operators**: == (equals), != (not equals), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to).
- **Assignment Operators:** = (assignment), += (add and assign), -= (subtract and assign), *= (multiply and assign), /= (divide and assign), %= (modulus and assign).

```
javaCopy code
public class OperatorsExample {
    public static void main(String[] args) {
        int a = 10, b = 5;

        // Arithmetic Operators
        int sum = a + b;
        int difference = a - b;
        int product = a * b;
        int division = a / b;
        int modulus = a % b;

        // Logical Operators
        boolean result1 = (a > b) && (a < 20);
        boolean result2 = (a == 10) || (b == 5);
        boolean result3 = !(a > 5);

        // Bitwise Operators
```

```
        int bitwiseAnd = a & b;
        int bitwiseOr = a | b;
        int bitwiseXor = a ^ b;
        int bitwiseNotA = ~a;

        // Comparison Operators
        boolean isEqual = a == b;
        boolean isNotEqual = a != b;
        boolean isGreater = a > b;
        boolean isLess = a < b;
        boolean isGreaterOrEqual = a >= b;
        boolean isLessOrEqual = a <= b;

        // Assignment Operators
        int num = 10;
        num += 5; // num = num + 5;
        num -= 3; // num = num - 3;
        num *= 2; // num = num * 2;
        num /= 4; // num = num / 4;
        num %= 3; // num = num % 3;
    }
}
```

14. **List some important keywords in Java and their significance.**
    Keywords are reserved words in Java that have predefined meanings. They cannot be used as identifiers (e.g., variable names) in the code.

Some important keywords in Java:

- `class`: Used to declare a class.
- `public`, `private`, `protected`: Access modifiers to control access to classes, fields, and methods.
- `static`: Used to create class-level variables and methods.
- `void`: Indicates that a method does not return a value.
- `final`: Used to declare constants or make classes, methods, or variables unchangeable.
- `new`: Used to create instances of classes (objects).
- `this`: Refers to the current object within a class.
- `super`: Refers to the superclass or parent class of the current class.
- `if`, `else`: Used for conditional branching.

- `while`, `for`: Used for looping.
- `try`, `catch`, `finally`: Used for exception handling.

Java If-else

15. **What is the purpose of the if-else statement in Java? Provide an example of its usage.**
The if-else statement allows you to execute different code blocks based on a condition. It is used for decision-making in Java programs.

```
javaCopy code
public class IfElseExample {
    public static void main(String[] args) {
        int num = 10;

        if (num > 0) {
            System.out.println("The number is positive.");
        } else {
            System.out.println("The number is non-positive.");
        }
    }
}
```

Java Switch Statement

16. **Explain the purpose of the switch statement in Java and provide an example.**
The switch statement allows you to select one of many code blocks to be executed based on the value of an expression.

```
javaCopy code
public class SwitchExample {
    public static void main(String[] args) {
        int dayOfWeek = 2;
        String dayName;

        switch (dayOfWeek) {
            case 1:
                dayName = "Sunday";
```

```
                break;
            case 2:
                dayName = "Monday";
                break;
            case 3:
                dayName = "Tuesday";
                break;
            // Add cases for other days
            default:
                dayName = "Invalid day";
        }

        System.out.println("Today is " + dayName);
    }
}
```

Java For Loop

17. **What is the purpose of the for loop in Java? Provide an example.**
The for loop is used to execute a block of code repeatedly based on a specified condition.

```
javaCopy code
public class ForLoopExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Iteration " + i);
        }
    }
}
```

Java While Loop

18. **Explain the purpose of the while loop in Java and provide an example.**
he while loop is used to repeatedly execute a block of code as long as a given condition is true.

```
javaCopy code
public class WhileLoopExample {
```

```
    public static void main(String[] args) {
        int count = 1;
        while (count <= 5) {
            System.out.println("Count: " + count);
            count++;
        }
    }
}
```

Java Do-While Loop

19. **Describe the purpose of the do-while loop in Java. Provide an example.**
The do-while loop is used to execute a block of code at least once, and then repeatedly as long as the specified condition is true.

```
javaCopy code
public class DoWhileLoopExample {
    public static void main(String[] args) {
        int count = 1;
        do {
            System.out.println("Count: " + count);
            count++;
        } while (count <= 5);
    }
}
```

Java Break Statement

20. **What is the purpose of the break statement in Java? Provide an example of its usage.**
The break statement is used to exit from a loop or switch statement prematurely.

```
javaCopy code
public class BreakExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i == 5) {
```

```
                break;
            }
            System.out.println("Iteration " + i);
        }
    }
}
```

Java Continue Statement

21. **Explain the purpose of the continue statement in Java and provide an example.**

The continue statement is used to skip the current iteration of a loop and continue with the next iteration.

```
javaCopy code
public class ContinueExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            if (i == 3) {
                continue;
            }
            System.out.println("Iteration " + i);
        }
    }
}
```

Java Comments

22. **Describe the different types of comments in Java and their purposes.**

Java supports three types of comments:

- **Single-line comments**: Used to comment a single line of code. They start with //.
- **Multi-line comments:** Used to comment multiple lines of code. They start with /* and end with */.
- **Java doc comments**: Used to generate documentation. They start with /** and end with */.

```
javaCopy code
public class CommentsExample {
    public static void main(String[] args) {
        // This is a single-line comment.

        /*
         * This is a multi-line comment.
         * It spans multiple lines.
         */

        /**
         * This is a Javadoc comment.
         * It provides documentation for the code.
         */
    }
}
```

Java Programs

23. **Write a Java program to find the sum of all even numbers from 1 to 100.**

```
javaCopy code
public class SumOfEvenNumbers {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 1; i <= 100; i++) {
            if (i % 2 == 0) {
                sum += i;
            }
        }
        System.out.println("Sum of even numbers from 1 to 100: " + sum);
    }
}
```

24. **Write a Programme to find the Number is Palindrome or not?**

```
public class New_Beginning {
public static void main(String[] args) {
int temp,rem,rev=0;
intn=121;
        temp=n;
```

```
while(n>0){
            rem=n%10;
            rev=(rev*10)+rem;
            n=n/10;
        }
if(rev==temp){
            System.out.println("Yes it ispalendrom");}
else{
            System.out.println("Not a palendrom number");}
    }
    }
```

JAVA OOPS

25. **What is a class?**
    A class is a blueprint for creating objects. It defines the structure and behavior of objects.

Example:

```
javaCopy code
class Car {
    String make;
    String model;

    void start() {
        System.out.println("Starting the car.");
    }
}
```

26. **What is an object?**
    An object is an instance of a class. It represents a real-world entity.

Example:

```
javaCopy code
public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.make = "Toyota";
        myCar.model = "Corolla";
```

```
        myCar.start();
    }
}
```

27. **What is the difference between a class and an object?**
    A class is a blueprint, while an object is an instance created from that blueprint.

28. **What are instance variables?**
    Instance variables are non-static variables declared inside a class. Each object of the class has its own copy of these variables.

Example:

```
javaCopy code
class Student {
    String name;
    int age;
}
```

29. **What are instance methods?**
    Instance methods are non-static methods that operate on the instance variables of a class.

Example:

```
javaCopy code
class Circle {
    double radius;

    double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```

30. **What is encapsulation?**
    Encapsulation is the process of hiding internal data and methods of a class and exposing only what is necessary through access modifiers (e.g., private, public, protected).

Example:

```
javaCopy code
class BankAccount {
    private double balance;

    public void deposit(double amount) {
        balance += amount;
    }

    public double getBalance() {
        return balance;
    }
}
```

31. **What is inheritance?**
    Inheritance is the process of acquiring the properties and behaviors of one class (superclass) by another class (subclass).

Example:

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

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

32. **What is the 'super' keyword used for in Java?**
    The 'super' keyword is used to call the superclass constructor or refer to the superclass's members (methods or variables) from the subclass.

Example:

```
javaCopy code
class Animal {
    String type;

    Animal(String type) {
        this.type = type;
    }
}

class Dog extends Animal {
    String breed;

    Dog(String type, String breed) {
        super(type); // Calling the superclass constructor
        this.breed = breed;
    }
}
```

33. **What is method overriding?**
    Method overriding is the process of providing a new implementation for a method that is already defined in the superclass.

Example:

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

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

34. **Can you override a private or final method in Java?**
    No, you cannot override a private method as it is not accessible outside the class. And you cannot override a final method as it is marked as a final and cannot be modified.

35. **What is method overloading?**
    Method overloading is the process of defining multiple methods with the same name but different parameters in the same class.

Example:

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

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

36. **What is a constructor?**
    A constructor is a special method that is used to initialize the state of an object when it is created.

Example:

```
javaCopy code
class Car {
    String make;
    String model;

    Car(String make, String model) {
        this.make = make;
        this.model = model;
    }
}
```

37. **Can you have multiple constructors in a class?**
    Yes, a class can have multiple constructors as long as they have different parameters (method overloading).

Example:

```
javaCopy code
class Car {
    String make;
    String model;

    Car(String make, String model) {
        this.make = make;
        this.model = model;
    }

    Car(String make) {
        this.make = make;
    }
}
```

38. **What is the default constructor?**
    If you don't define any constructor in a class, Java provides a default constructor with no arguments.

Example:

```
javaCopy code
class MyClass {
    // Default constructor
}
```

39. **What is the purpose of the 'this' keyword?**
    The 'this' keyword is a reference to the current object. It is used to distinguish between instance variables and parameters with the same name.

Example:

```
javaCopy code
class Person {
    String name;

    Person(String name) {
        this.name = name; // 'this' is used to refer to the instance variable
    }
}
```

40. **What is the difference between shallow copy and deep copy?**
    Shallow copy creates a new object but does not duplicate the contents. Deep copy creates a new object and duplicates the contents.

41. **What is the 'instanceof' operator used for?**
    The 'instanceof' operator is used to check if an object is an instance of a particular class or implements a particular interface.

Example:

```
javaCopy code
class Animal { }
class Dog extends Animal { }

public class Main {
    public static void main(String[] args) {
        Animal animal = new Dog();
        System.out.println(animal instanceof Dog); // Output: true
        System.out.println(animal instanceof Animal); // Output: true
    }
}
```

42. **What is polymorphism?**
    Polymorphism allows a method to take on different forms based on the type of objects it operates on. It can be achieved through method overloading and method overriding.

43. **What is an abstract class?**
    An abstract class is a class that cannot be instantiated and may have abstract methods (methods without implementation). It serves as a blueprint for its subclasses.

Example:

```
javaCopy code
abstract class Shape {
    abstract void draw();
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing a circle.");
    }
```

```
}
```

44. **Can an abstract class have a constructor?**
    Yes, an abstract class can have a constructor, and it is called when an instance of a concrete subclass is created.

Example:

```
javaCopy code
abstract class Animal {
    String name;

    Animal(String name) {
        this.name = name;
    }

    abstract void sound();
}

class Dog extends Animal {
    Dog(String name) {
        super(name);
    }

    void sound() {
        System.out.println("Dog barks.");
    }
}
```

45. **What is an interface?**
    An interface is a blueprint for a class that defines a set of abstract methods that must be implemented by the classes that implement the interface.

Example:

```
javaCopy code
interface Shape {
    void draw();
}

class Circle implements Shape {
    public void draw() {
        System.out.println("Drawing a circle.");
    }
```

```
}
```

46. **What is the difference between an abstract class and an interface?**
    An abstract class can have abstract and non-abstract methods, while an interface can only have abstract methods. A class can implement multiple interfaces, but it can extend only one abstract class.

47. **Can an interface have variables?**
    Yes, an interface can have variables, but they are implicitly public, static, and final (constants).

Example:

```
javaCopy code
interface Constants {
    int MAX_COUNT = 100;
}
```

48. **What is a default method in an interface?**
    A default method is a method with a default implementation provided in the interface. It allows adding new methods to an existing interface without breaking the classes that implement it.

Example:

```
javaCopy code
interface Printer {
    default void print() {
        System.out.println("Printing...");
    }
}
```

49. **Can an interface extend another interface?**
    Yes, an interface can extend multiple interfaces.

Example:

```
javaCopy code
interface Shape {
    void draw();
}

interface Circle extends Shape {
    void radius();
}
```

50. **What is a static method in an interface?**
    A static method in an interface is a method that belongs to the interface and can be called using the interface name.

Example:

```
javaCopy code
interface MathOperation {
    static int add(int a, int b) {
        return a + b;
    }
}
```

51. **What is a nested class?**
    A nested class is a class defined inside another class. It can be static or non-static.

Example:

```
javaCopy code
class Outer {
    int x;

    class Inner {
        int y;
    }
}
```

52. **What is a static nested class?**
    A static nested class is a nested class that is marked as static. It does not require an instance of the outer class to be instantiated.

Example:

```
javaCopy code
class Outer {
    static class Nested {
        int x;
    }
}
```

53. **What is a local inner class?**
    A local inner class is a class defined inside a method. It can only be used within that method.

Example:

```
javaCopy code
class Outer {
    void display() {
        class Inner {
            void show() {
                System.out.println("Inside the local inner class.");
            }
        }

        Inner inner = new Inner();
        inner.show();
    }
}
```

54. **What is an anonymous inner class?**
    An anonymous inner class is a class defined without a name. It is usually used for implementing interfaces or extending classes.

Example (with interface implementation):

```
javaCopy code
interface Greeting {
    void greet();
}

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

```
        Greeting greeting = new Greeting() {
            public void greet() {
                System.out.println("Hello, anonymous inner class!");
            }
        };

        greeting.greet();
    }
}
```

55. **What is a final class in Java?**
    A final class is a class that cannot be subclassed (extended). It prevents inheritance.

Example:

```
javaCopy code
final class MyClass {
    // Class implementation
}
```

56. **What is a final method in Java?**
    A final method is a method that cannot be overridden by subclasses.

Example:

```
javaCopy code
class Parent {
    final void display() {
        System.out.println("This method cannot be overridden.");
    }
}
```

57. **What is a final variable in Java?**
    A final variable is a variable whose value cannot be changed once assigned.

Example:

```
javaCopy code
class Circle {
```

```
    final double PI = 3.14;
}
```

58. **What is a static variable?**
    A static variable is a class-level variable that is shared among all instances of the class. It belongs to the class rather than to any specific instance.

Example:

```
javaCopy code
class Counter {
    static int count = 0;

    Counter() {
        count++;
    }
}
```

59. **What is the 'final' keyword used for in Java?**
    The 'final' keyword is used to define entities that cannot be modified. It can be used with classes, methods, and variables.
60. **What is the 'static' keyword used for in Java?**
    The 'static' keyword is used to define class-level entities (methods and variables) that belong to the class rather than any specific instance.
61. **What is the 'this' reference and when is it used?**
    The 'this' reference is used to refer to the current instance of a class. It is often used to distinguish between instance variables and method parameters with the same name.
62. **What is the purpose of the 'super' keyword in Java?**
    The 'super' keyword is used to call the superclass's constructor or refer to the superclass's members from the subclass.
63. **How do you prevent a class from being inherited in Java?**
    To prevent a class from being inherited, mark it as 'final.'
64. **How do you prevent a method from being overridden in Java?**
    To prevent a method from being overridden, mark it as 'final' in the superclass.

65. **Can a class be both 'abstract' and 'final' in Java?**
    No, a class cannot be both 'abstract' and 'final.' An abstract class is meant to be subclassed, while a final class cannot be subclassed.

66. **What is the purpose of the 'toString()' method?**
    The 'toString()' method is used to convert an object to its string representation. It is often overridden to provide a meaningful string representation of the object.

Example:

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

    @Override
    public String toString() {
        return "Person[name=" + name + ", age=" + age + "]";
    }
}
```

67. **What is the 'equals()' method used for?**
    The 'equals()' method is used to compare the content of two objects for equality. It is often overridden to provide a custom comparison logic.

Example:

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

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Person person = (Person) obj;
        return age == person.age && Objects.equals(name, person.name);
    }
}
```

68. **What is the 'hashCode()' method used for?**
    The 'hashCode()' method is used to generate a unique integer value for an object. It is used in hash-based collections like HashMap to determine the bucket where the object should be stored.

Example:

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

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

69. **What is the purpose of the 'clone()' method?**
    The 'clone()' method is used to create a copy of an object. To use it, the class must implement the 'Cloneable' interface.

Example:

```
javaCopy code
class MyClass implements Cloneable {
    // Class implementation

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

70. **What is the 'instance initializer block' used for?**
    The instance initializer block is used to initialize instance variables when an object is created. It is executed before the constructor.

Example:

```
javaCopy code
class MyClass {
    int x;

    {
        x = 10; // Instance initializer block
    }
}
```

71. **What is the 'static initializer block' used for?**
    The static initializer block is used to initialize static variables when the class is loaded. It is executed only once.

Example:

```
javaCopy code
class MyClass {
    static int x;

    static {
        x = 5; // Static initializer block
    }
}
```

72. **How do you achieve multiple inheritances in Java?**
    Java does not support multiple inheritance through classes (i.e., a class cannot extend multiple classes). However, it can be achieved through interfaces.

73. **What is method hiding in Java?**
    Method hiding occurs when a subclass defines a static method with the same signature as a static method in its superclass. The subclass's method hides the superclass's method.

Example:

```
javaCopy code
class Parent {
    static void display() {
        System.out.println("Parent's static method.");
    }
}
```

```
class Child extends Parent {
    static void display() {
        System.out.println("Child's static method.");
    }
}
```

74. **What is an inner class and what are its advantages?**
    An inner class is a class defined inside another class. It has access to the members of the outer class, including private members. Inner classes are often used for logical grouping and encapsulation.

Example:

```
javaCopy code
class Outer {
    private int x;

    class Inner {
        void setX(int value) {
            x = value; // Accessing the private member of the outer class
        }
    }
}
```

Advantages of inner classes:

- Better encapsulation, as the inner class can access private members of the outer class.
- Improved code organization, as the related classes are grouped together.
- Better readability and maintainability in some cases, as it reduces the scope of the inner class and avoids naming conflicts.
- Ability to implement more elegant designs using design patterns like the Iterator pattern.