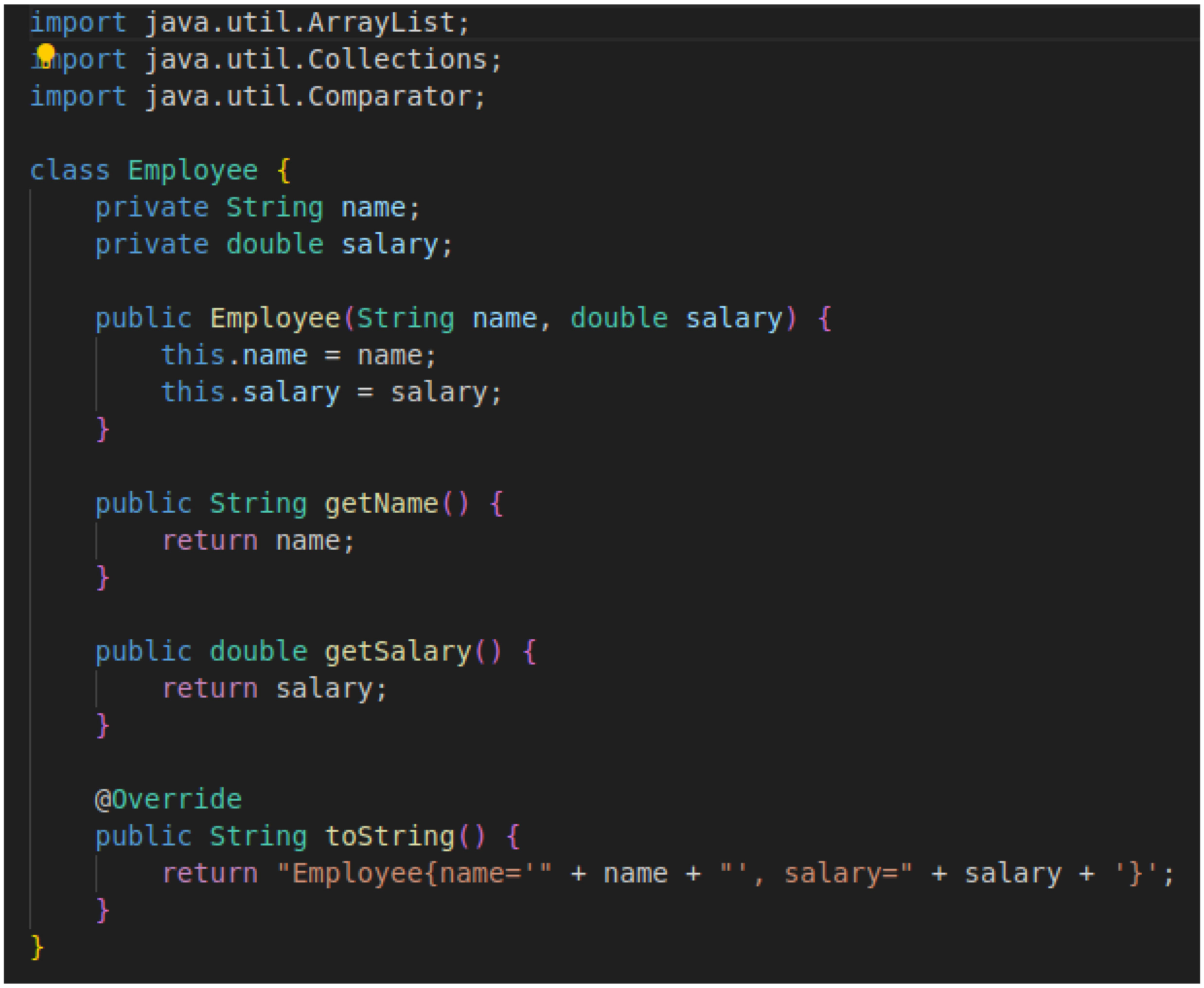
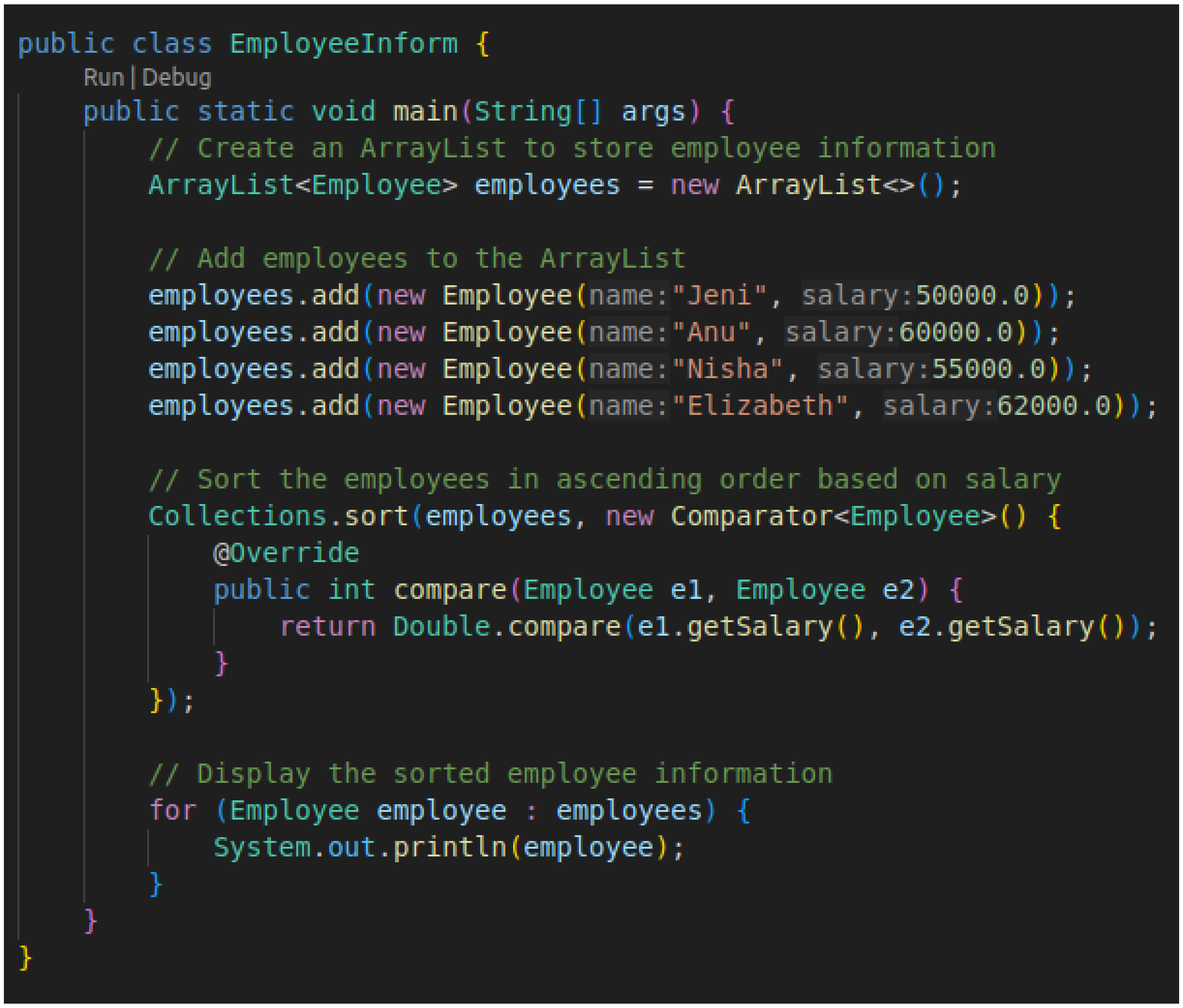
PG-DAC THIRUVANANTHAPURAM – SEPTEMBER 2023 BATCH CCPP SERIES

Prepare the answers and submit it on or before 28/10/2023, 11.59 PM

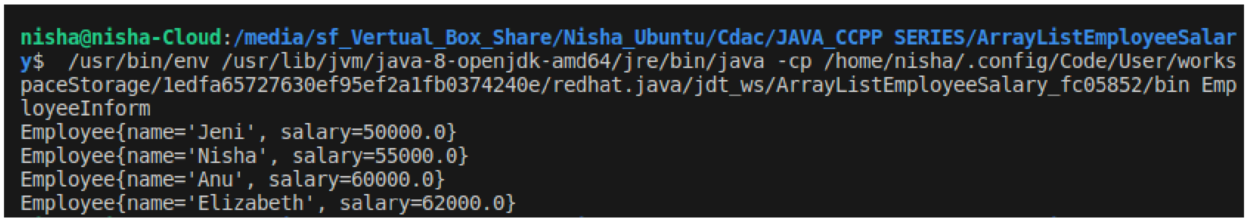
1. Use ArrayList collection class to Store the information (like name and salary)of employees . Display the employee information in an ascending order.



```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

class Employee {
    private String name;
    private double salary;

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{name='" + name + "', salary=" + salary + '}';
    }
}
```

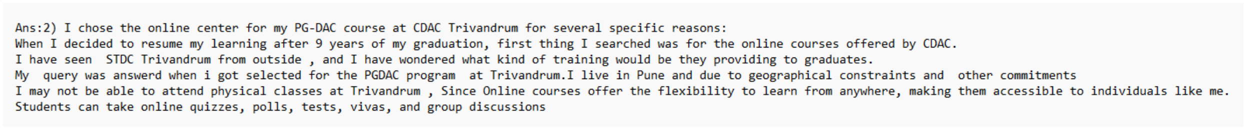


```java
public class EmployeeInform {
    public static void main(String[] args) {
        // Create an ArrayList to store employee information
        ArrayList<Employee> employees = new ArrayList<>();

        // Add employees to the ArrayList
        employees.add(new Employee(name:"Jeni", salary:50000.0));
        employees.add(new Employee(name:"Anu", salary:60000.0));
        employees.add(new Employee(name:"Nisha", salary:55000.0));
        employees.add(new Employee(name:"Elizabeth", salary:62000.0));

        // Sort the employees in ascending order based on salary
        Collections.sort(employees, new Comparator<Employee>() {
            @Override
            public int compare(Employee e1, Employee e2) {
                return Double.compare(e1.getSalary(), e2.getSalary());
            }
        });

        // Display the sorted employee information
        for (Employee employee : employees) {
            System.out.println(employee);
        }
    }
}
```



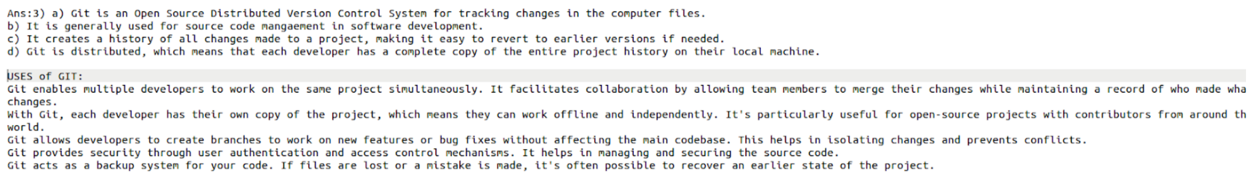
```
nisha@nisha-Cloud:/media/sf_Vertual_Box_Share/Nisha_Ubuntu/Cdac/JAVA_CCPP SERIES/ArrayListEmployeeSalar
y$  /usr/bin/env /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java -cp /home/nisha/.config/Code/User/works
paceStorage/1edfa65727630ef95ef2a1fb0374240e/redhat.java/jdt_ws/ArrayListEmployeeSalary_fc05852/bin Emp
loyeeInform
Employee{name='Jeni', salary=50000.0}
Employee{name='Nisha', salary=55000.0}
Employee{name='Anu', salary=60000.0}
Employee{name='Elizabeth', salary=62000.0}
```

2. Why do you choose online centre fo PG-DAC course?



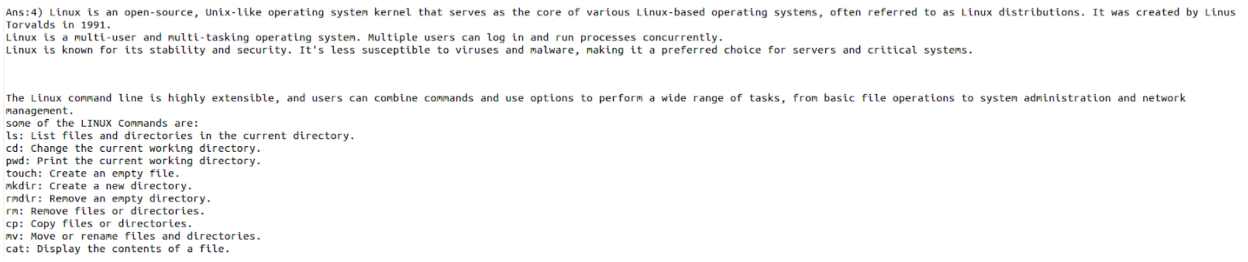
```
Ans:2) I chose the online center for my PG-DAC course at CDAC Trivandrum for several specific reasons:
When I decided to resume my learning after 9 years of my graduation, first thing I searched was for the online courses offered by CDAC.
I have seen  STDC Trivandrum from outside , and I have wondered what kind of training would be they providing to graduates.
My  query was answerd when i got selected for the PGDAC program  at Trivandrum.I live in Pune and due to geographical constraints and  other commitments
I may not be able to attend physical classes at Trivandrum , Since Online courses offer the flexibility to learn from anywhere, making them accessible to individuals like me.
Students can take online quizzes, polls, tests, vivas, and group discussions
```

3. Briefly explain what is git why we use git



```
Ans:3) a) Git is an Open Source Distributed Version Control System for tracking changes in the computer files.
b) It is generally used for source code mangaement in software development.
c) It creates a history of all changes made to a project, making it easy to revert to earlier versions if needed.
d) Git is distributed, which means that each developer has a complete copy of the entire project history on their local machine.

USES of GIT:
Git enables multiple developers to work on the same project simultaneously. It facilitates collaboration by allowing team members to merge their changes while maintaining a record of who made wha
changes.
With Git, each developer has their own copy of the project, which means they can work offline and independently. It's particularly useful for open-source projects with contributors from around th
world.
Git allows developers to create branches to work on new features or bug fixes without affecting the main codebase. This helps in isolating changes and prevents conflicts.
Git provides security through user authentication and access control mechanisms. It helps in managing and securing the source code.
Git acts as a backup system for your code. If files are lost or a mistake is made, it's often possible to recover an earlier state of the project.
```

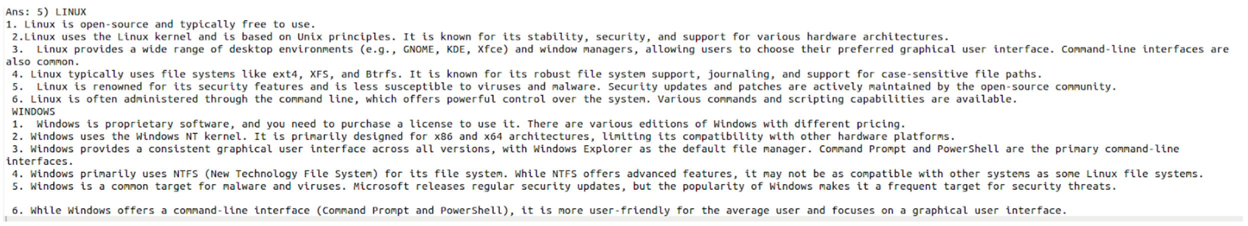
4. What do you know about Linux operating system and commands used in Linux



Ans:4) Linux is an open-source, Unix-like operating system kernel that serves as the core of various Linux-based operating systems, often referred to as Linux distributions. It was created by Linus Torvalds in 1991.
Linux is a multi-user and multi-tasking operating system. Multiple users can log in and run processes concurrently.
Linux is known for its stability and security. It's less susceptible to viruses and malware, making it a preferred choice for servers and critical systems.

The Linux command line is highly extensible, and users can combine commands and use options to perform a wide range of tasks, from basic file operations to system administration and network management.
some of the LINUX Commands are:
ls: List files and directories in the current directory.
cd: Change the current working directory.
pwd: Print the current working directory.
touch: Create an empty file.
mkdir: Create a new directory.
rmdir: Remove an empty directory.
rm: Remove files or directories.
cp: Copy files or directories.
mv: Move or rename files and directories.
cat: Display the contents of a file.

5. Briefly explain the Differences between Linux and windows operating system.

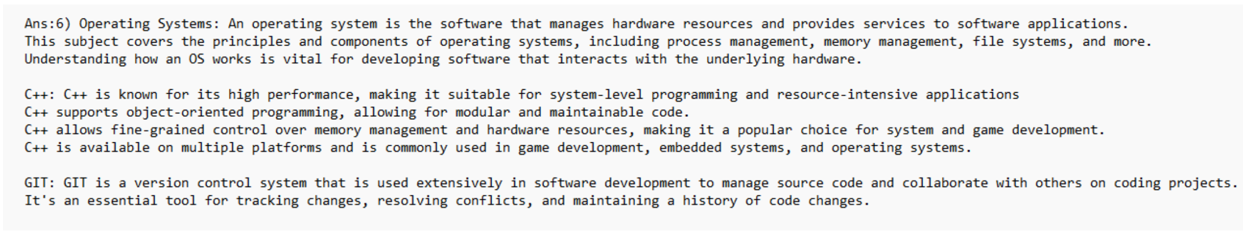


Ans: 5) LINUX
1. Linux is open-source and typically free to use.
2.Linux uses the Linux kernel and is based on Unix principles. It is known for its stability, security, and support for various hardware architectures.
3. Linux provides a wide range of desktop environments (e.g., GNOME, KDE, Xfce) and window managers, allowing users to choose their preferred graphical user interface. Command-line interfaces are also common.
4. Linux typically uses file systems like ext4, XFS, and Btrfs. It is known for its robust file system support, journaling, and support for case-sensitive file paths.
5. Linux is renowned for its security features and is less susceptible to viruses and malware. Security updates and patches are actively maintained by the open-source community.
6. Linux is often administered through the command line, which offers powerful control over the system. Various commands and scripting capabilities are available.

WINDOWS
1. Windows is proprietary software, and you need to purchase a license to use it. There are various editions of Windows with different pricing.
2. Windows uses the Windows NT kernel. It is primarily designed for x86 and x64 architectures, limiting its compatibility with other hardware platforms.
3. Windows provides a consistent graphical user interface across all versions, with Windows Explorer as the default file manager. Command Prompt and PowerShell are the primary command-line interfaces.
4. Windows primarily uses NTFS (New Technology File System) for its file system. While NTFS offers advanced features, it may not be as compatible with other systems as some Linux file systems.
5. Windows is a common target for malware and viruses. Microsoft releases regular security updates, but the popularity of Windows makes it a frequent target for security threats.

6. While Windows offers a command-line interface (Command Prompt and PowerShell), it is more user-friendly for the average user and focuses on a graphical user interface.

6. Which are your top 3 favorite subjects in DAC and why?(May mention any 3 topic you studied for the time being)

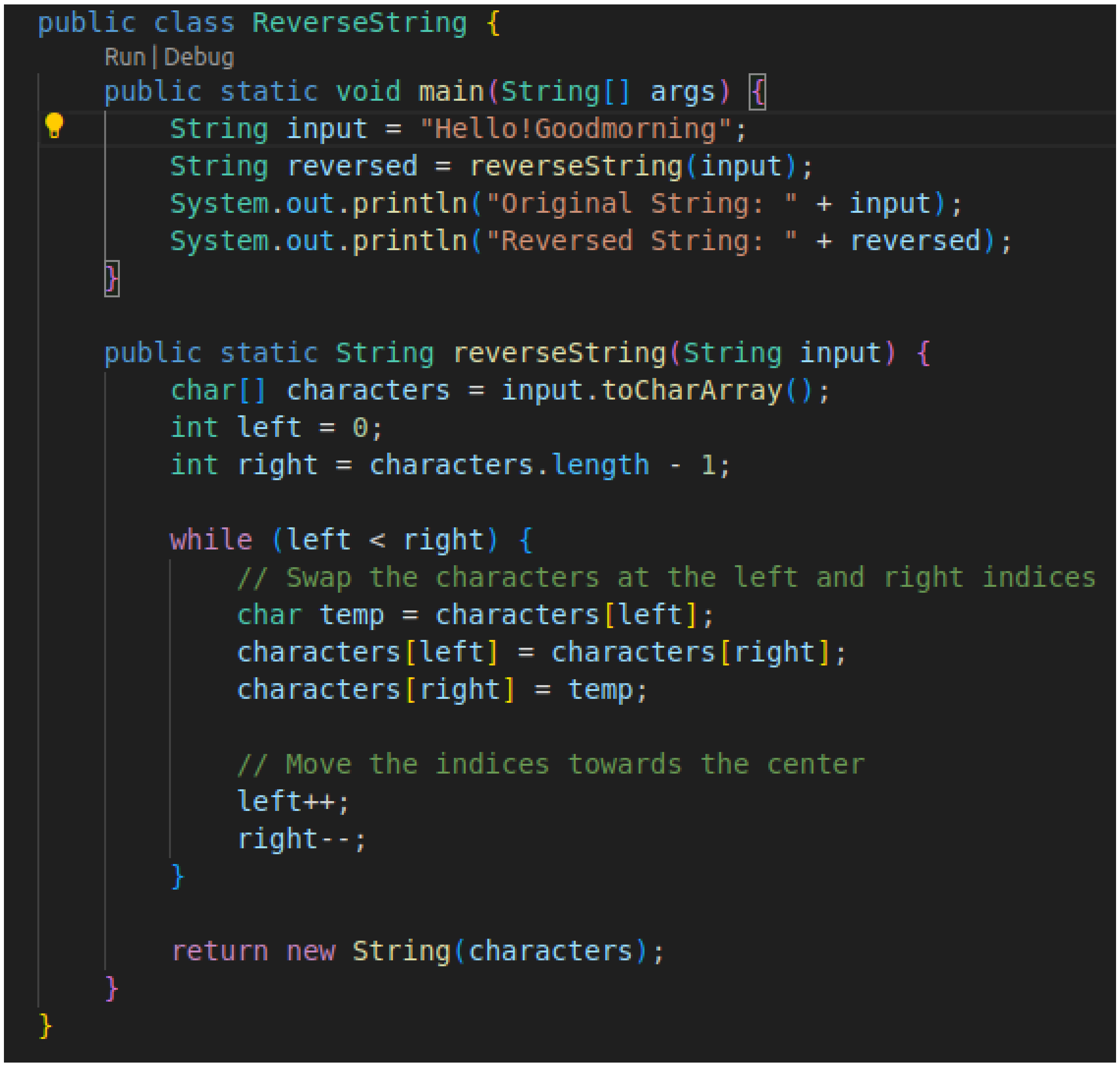


Ans:6) Operating Systems: An operating system is the software that manages hardware resources and provides services to software applications.
This subject covers the principles and components of operating systems, including process management, memory management, file systems, and more.
Understanding how an OS works is vital for developing software that interacts with the underlying hardware.

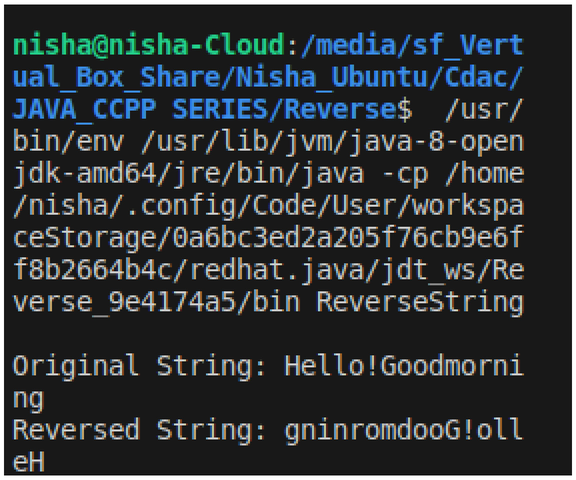
C++: C++ is known for its high performance, making it suitable for system-level programming and resource-intensive applications
C++ supports object-oriented programming, allowing for modular and maintainable code.
C++ allows fine-grained control over memory management and hardware resources, making it a popular choice for system and game development.
C++ is available on multiple platforms and is commonly used in game development, embedded systems, and operating systems.

GIT: GIT is a version control system that is used extensively in software development to manage source code and collaborate with others on coding projects.
It's an essential tool for tracking changes, resolving conflicts, and maintaining a history of code changes.

7. Write a code in Java to reverse a string (do not use string methods)

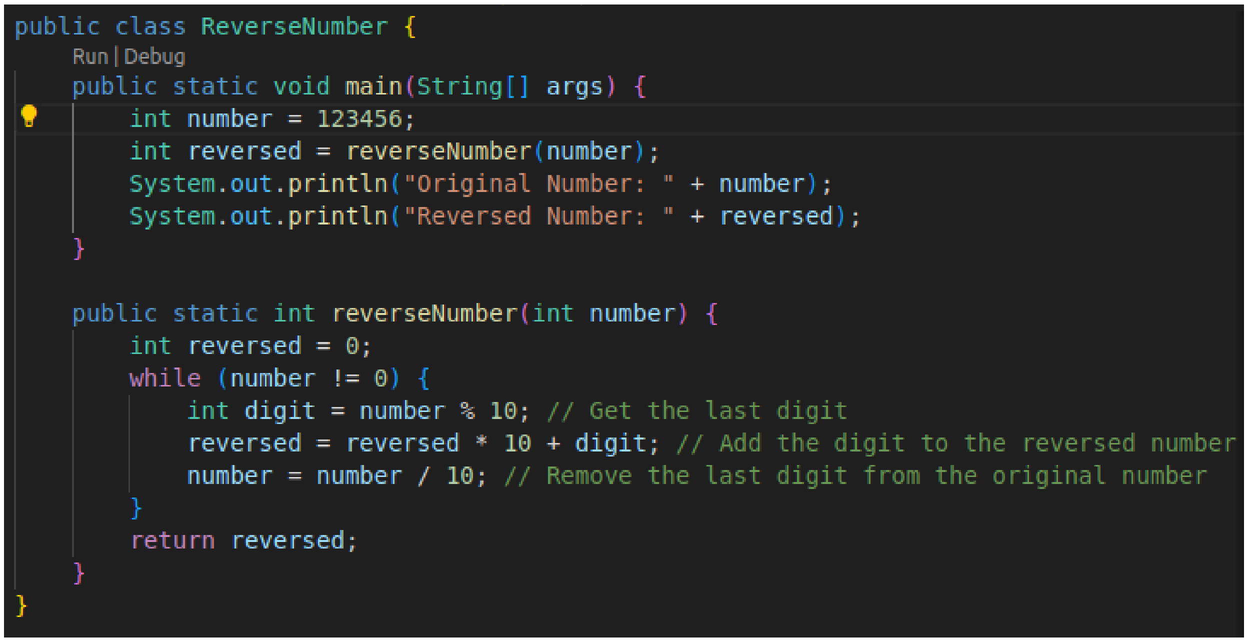


```java
public class ReverseString {
    public static void main(String[] args) {
        String input = "Hello!Goodmorning";
        String reversed = reverseString(input);
        System.out.println("Original String: " + input);
        System.out.println("Reversed String: " + reversed);
    }

    public static String reverseString(String input) {
        char[] characters = input.toCharArray();
        int left = 0;
        int right = characters.length - 1;

        while (left < right) {
            // Swap the characters at the left and right indices
            char temp = characters[left];
            characters[left] = characters[right];
            characters[right] = temp;

            // Move the indices towards the center
            left++;
            right--;
        }

        return new String(characters);
    }
}
```

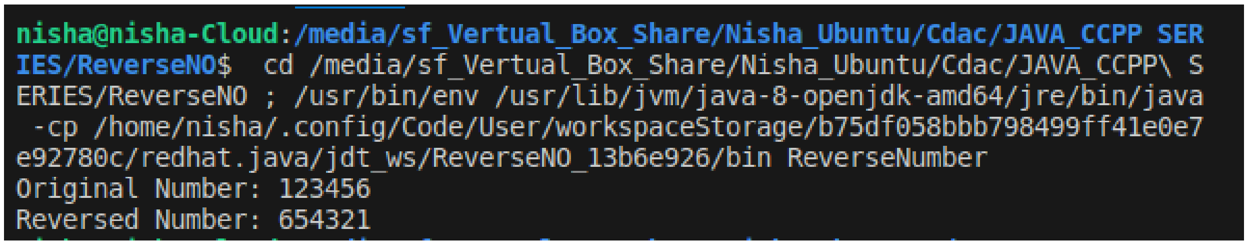


```
nisha@nisha-Cloud:/media/sf_Vert
ual_Box_Share/Nisha_Ubuntu/Cdac/
JAVA_CCPP SERIES/Reverse$  /usr/
bin/env /usr/lib/jvm/java-8-open
jdk-amd64/jre/bin/java -cp /home
/nisha/.config/Code/User/workspa
ceStorage/0a6bc3ed2a205f76cb9e6f
f8b2664b4c/redhat.java/jdt_ws/Re
verse_9e4174a5/bin ReverseString

Original String: Hello!Goodmorni
ng
Reversed String: gninromdooG!oll
eH
```

8. Write a code in Java to reverse a number without using the 3rd variable

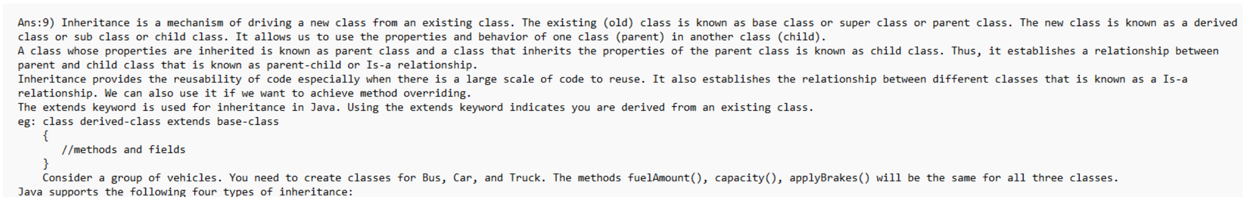


```java
public class ReverseNumber {
    Run | Debug
    public static void main(String[] args) {
        int number = 123456;
        int reversed = reverseNumber(number);
        System.out.println("Original Number: " + number);
        System.out.println("Reversed Number: " + reversed);
    }

    public static int reverseNumber(int number) {
        int reversed = 0;
        while (number != 0) {
            int digit = number % 10; // Get the last digit
            reversed = reversed * 10 + digit; // Add the digit to the reversed number
            number = number / 10; // Remove the last digit from the original number
        }
        return reversed;
    }
}
```



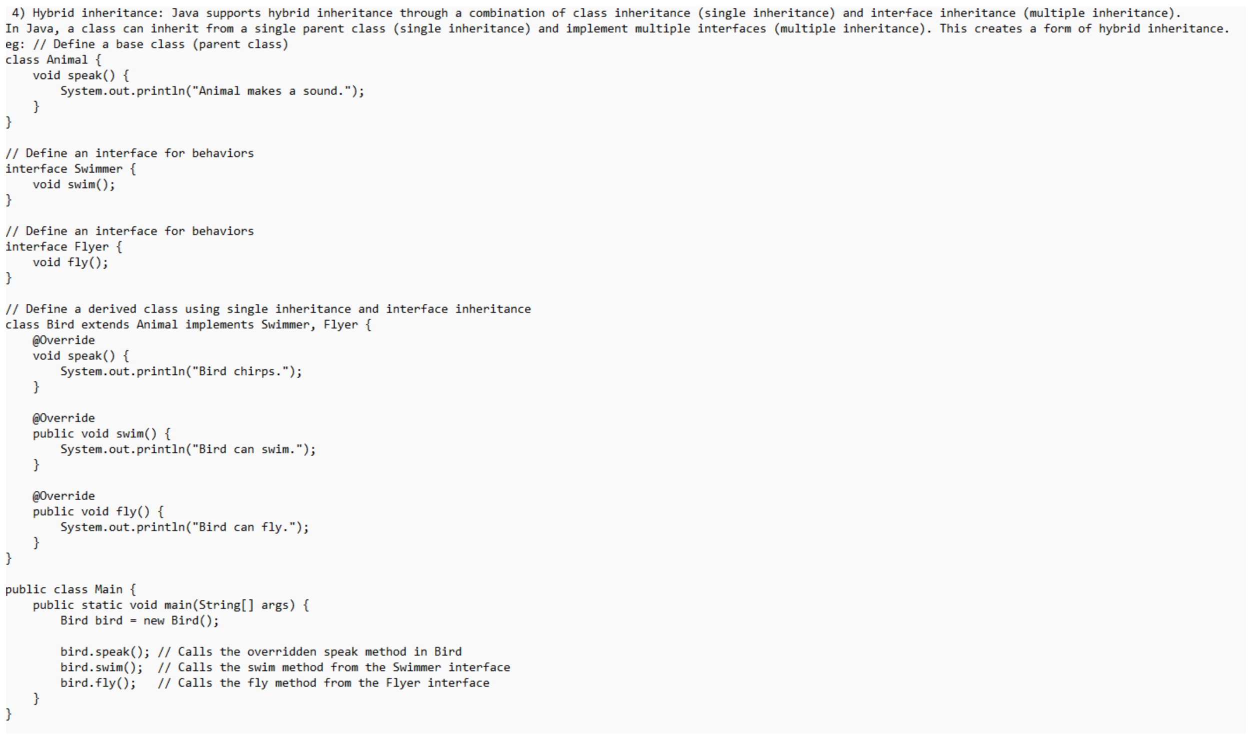
```
nisha@nisha-Cloud:/media/sf_Vertual_Box_Share/Nisha_Ubuntu/Cdac/JAVA_CCPP SER
IES/ReverseNO$  cd /media/sf_Vertual_Box_Share/Nisha_Ubuntu/Cdac/JAVA_CCPP\ S
ERIES/ReverseNO ; /usr/bin/env /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
 -cp /home/nisha/.config/Code/User/workspaceStorage/b75df058bbb798499ff41e0e7
e92780c/redhat.java/jdt_ws/ReverseNO_13b6e926/bin ReverseNumber
Original Number: 123456
Reversed Number: 654321
```

9. What is inheritance & types of inheritance with examples



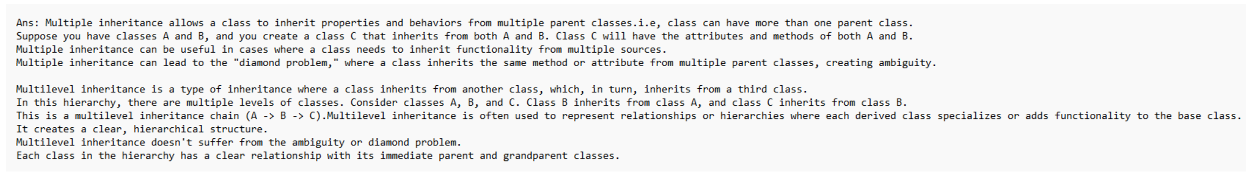
```
Ans:9) Inheritance is a mechanism of driving a new class from an existing class. The existing (old) class is known as base class or super class or parent class. The new class is known as a derived
class or sub class or child class. It allows us to use the properties and behavior of one class (parent) in another class (child).
A class whose properties are inherited is known as parent class and a class that inherits the properties of the parent class is known as child class. Thus, it establishes a relationship between
parent and child class that is known as parent-child or Is-a relationship.
Inheritance provides the reusability of code especially when there is a large scale of code to reuse. It also establishes the relationship between different classes that is known as a Is-a
relationship. We can also use it if we want to achieve method overriding.
The extends keyword is used for inheritance in Java. Using the extends keyword indicates you are derived from an existing class.
eg: class derived-class extends base-class
    {
       //methods and fields
    }
    Consider a group of vehicles. You need to create classes for Bus, Car, and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be the same for all three classes.
Java supports the following four types of inheritance:
 1) Single Inheritance
 In single inheritance, a sub-class is derived from only one super class. It inherits the properties and behavior of a single-parent class.
 It is also known as simple inheritance.
eg: // Define a superclass (parent class)
class Animal {
    String name;

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

    public void speak() {
        System.out.println(name + " makes a sound.");
    }
}

// Define a subclass (child class) using single inheritance
class Dog extends Animal {
    public Dog(String name) {
        super(name);
    }

    @Override
    public void speak() {
        System.out.println(name + " says Woof!");
    }
}

public class Main {
    public static void main(String[] args) {
        // Create an instance of the Dog class
        Dog dog = new Dog("Buddy");

        // Call the speak method on the Dog instance
        dog.speak();
    }
}
```

```
 2) Multi-level Inheritance
  In multi-level inheritance, a class is derived from a class which is also derived from another class is called multi-level inheritance.
 A class that has more than one parent class is called multi
  level inheritance.There exists a single base class and single derived class but multiple intermediate base classes.
 eg:
 // Define the base class (parent class)
 class Vehicle {
     void start() {
         System.out.println("Vehicle started.");
     }
 }

 // Define an intermediate class derived from Vehicle
 class Car extends Vehicle {
     void drive() {
         System.out.println("Car is driving.");
     }
 }

 // Define the final derived class using multi-level inheritance
 class SportsCar extends Car {
     void accelerate() {
         System.out.println("Sports car is accelerating.");
     }
 }

 public class Main {
     public static void main(String[] args) {
         SportsCar sportsCar = new SportsCar();

         sportsCar.start(); // Inherited from Vehicle
         sportsCar.drive(); // Inherited from Car
         sportsCar.accelerate(); // Defined in SportsCar
     }
 }
 3) Hierarchical Inheritance: If a number of classes are derived from a single base class,
it is called hierarchical inheritance.
eg:
 // Define the base class (parent class)
class Shape {
    void draw() {
        System.out.println("Drawing a shape.");
    }
}

// Define two child classes derived from Shape
class Circle extends Shape {
    void draw() {
        System.out.println("Drawing a circle.");
    }
}

class Rectangle extends Shape {
    void draw() {
        System.out.println("Drawing a rectangle.");
    }
}

public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle();
        Rectangle rectangle = new Rectangle();

        circle.draw(); // Calls the draw method in Circle
        rectangle.draw(); // Calls the draw method in Rectangle
    }
}
```



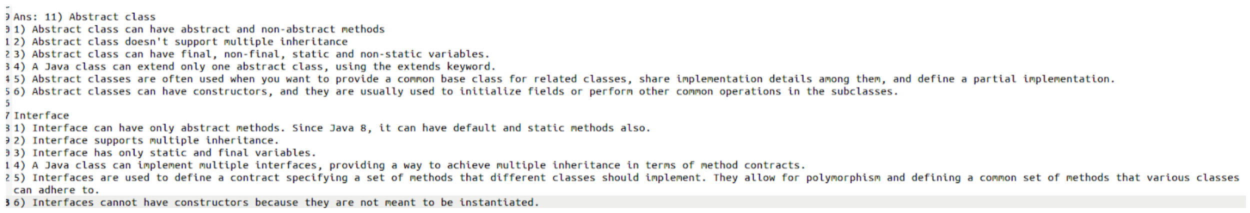
```
 4) Hybrid inheritance: Java supports hybrid inheritance through a combination of class inheritance (single inheritance) and interface inheritance (multiple inheritance).
In Java, a class can inherit from a single parent class (single inheritance) and implement multiple interfaces (multiple inheritance). This creates a form of hybrid inheritance.
eg: // Define a base class (parent class)
class Animal {
    void speak() {
        System.out.println("Animal makes a sound.");
    }
}

// Define an interface for behaviors
interface Swimmer {
    void swim();
}

// Define an interface for behaviors
interface Flyer {
    void fly();
}

// Define a derived class using single inheritance and interface inheritance
class Bird extends Animal implements Swimmer, Flyer {
    @Override
    void speak() {
        System.out.println("Bird chirps.");
    }

    @Override
    public void swim() {
        System.out.println("Bird can swim.");
    }

    @Override
    public void fly() {
        System.out.println("Bird can fly.");
    }
}

public class Main {
    public static void main(String[] args) {
        Bird bird = new Bird();

        bird.speak(); // Calls the overridden speak method in Bird
        bird.swim();  // Calls the swim method from the Swimmer interface
        bird.fly();   // Calls the fly method from the Flyer interface
    }
}
```

10. Difference between multiple & multilevel inheritance



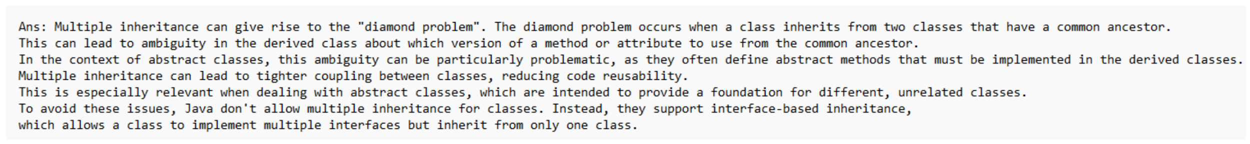
```
Ans: Multiple inheritance allows a class to inherit properties and behaviors from multiple parent classes.i.e, class can have more than one parent class.
Suppose you have classes A and B, and you create a class C that inherits from both A and B. Class C will have the attributes and methods of both A and B.
Multiple inheritance can be useful in cases where a class needs to inherit functionality from multiple sources.
Multiple inheritance can lead to the "diamond problem," where a class inherits the same method or attribute from multiple parent classes, creating ambiguity.

Multilevel inheritance is a type of inheritance where a class inherits from another class, which, in turn, inherits from a third class.
In this hierarchy, there are multiple levels of classes. Consider classes A, B, and C. Class B inherits from class A, and class C inherits from class B.
This is a multilevel inheritance chain (A -> B -> C).Multilevel inheritance is often used to represent relationships or hierarchies where each derived class specializes or adds functionality to the base class.
It creates a clear, hierarchical structure.
Multilevel inheritance doesn't suffer from the ambiguity or diamond problem.
Each class in the hierarchy has a clear relationship with its immediate parent and grandparent classes.
```

11. Difference between abstract and interface



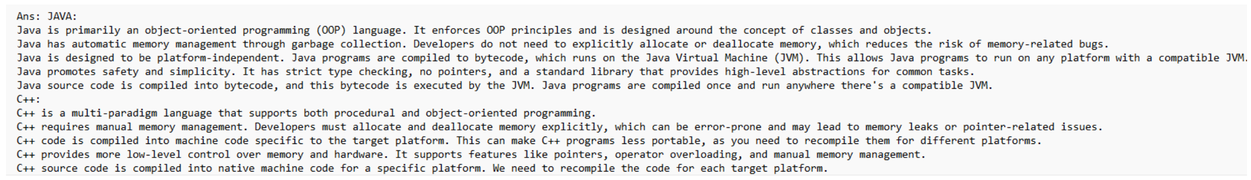
```
Ans: 11) Abstract class
1) Abstract class can have abstract and non-abstract methods
2) Abstract class doesn't support multiple inheritance
3) Abstract class can have final, non-final, static and non-static variables.
4) A Java class can extend only one abstract class, using the extends keyword.
5) Abstract classes are often used when you want to provide a common base class for related classes, share implementation details among them, and define a partial implementation.
6) Abstract classes can have constructors, and they are usually used to initialize fields or perform other common operations in the subclasses.

Interface
1) Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Interface supports multiple inheritance.
3) Interface has only static and final variables.
4) A Java class can implement multiple interfaces, providing a way to achieve multiple inheritance in terms of method contracts.
5) Interfaces are used to define a contract specifying a set of methods that different classes should implement. They allow for polymorphism and defining a common set of methods that various classes
can adhere to.
6) Interfaces cannot have constructors because they are not meant to be instantiated.
```

12. Why we cannot use multiple inheritance in abstract class



```
Ans: Multiple inheritance can give rise to the "diamond problem". The diamond problem occurs when a class inherits from two classes that have a common ancestor.
This can lead to ambiguity in the derived class about which version of a method or attribute to use from the common ancestor.
In the context of abstract classes, this ambiguity can be particularly problematic, as they often define abstract methods that must be implemented in the derived classes.
Multiple inheritance can lead to tighter coupling between classes, reducing code reusability.
This is especially relevant when dealing with abstract classes, which are intended to provide a foundation for different, unrelated classes.
To avoid these issues, Java don't allow multiple inheritance for classes. Instead, they support interface-based inheritance,
which allows a class to implement multiple interfaces but inherit from only one class.
```

13. What is the difference between Java and CPP



```
Ans: JAVA:
Java is primarily an object-oriented programming (OOP) language. It enforces OOP principles and is designed around the concept of classes and objects.
Java has automatic memory management through garbage collection. Developers do not need to explicitly allocate or deallocate memory, which reduces the risk of memory-related bugs.
Java is designed to be platform-independent. Java programs are compiled to bytecode, which runs on the Java Virtual Machine (JVM). This allows Java programs to run on any platform with a compatible JVM.
Java promotes safety and simplicity. It has strict type checking, no pointers, and a standard library that provides high-level abstractions for common tasks.
Java source code is compiled into bytecode, and this bytecode is executed by the JVM. Java programs are compiled once and run anywhere there's a compatible JVM.
C++:
C++ is a multi-paradigm language that supports both procedural and object-oriented programming.
C++ requires manual memory management. Developers must allocate and deallocate memory explicitly, which can be error-prone and may lead to memory leaks or pointer-related issues.
C++ code is compiled into machine code specific to the target platform. This can make C++ programs less portable, as you need to recompile them for different platforms.
C++ provides more low-level control over memory and hardware. It supports features like pointers, operator overloading, and manual memory management.
C++ source code is compiled into native machine code for a specific platform. We need to recompile the code for each target platform.
```

14. Difference between pointers & references

```
Ans:Pointers:
Pointers are variables that store memory addresses. They are declared using an asterisk (*) before the variable name (e.g., int* ptr;)
Pointers need to be explicitly assigned a memory address using the address-of operator (&) or by setting them equal to another pointer (e.g., int* ptr = &someVar;)
Pointers can be set to nullptr (or NULL in C or C++) to represent a "null" or "invalid" reference, meaning they don't point to any valid memory location.
Pointers require explicit dereferencing using the asterisk (*) to access the value they point to (e.g., int value = *ptr;).
Pointers are often used when you need to work with dynamic memory allocation (e.g., with new and delete in C++) or
when dealing with data structures that require low-level memory manipulation.
References:
References are aliases to other variables. They are declared using an ampersand (&) after the variable type (e.g., int& ref = someVar;).
References must be initialized at the point of declaration, and once they are initialized, they cannot be changed to refer to a different variable.
References always refer to a valid object; there is no concept of a null reference. This makes references safer in some situations because you don't need to check for nullness.
References do not need explicit dereferencing; you can use them directly as if they were the actual object (e.g., int value = ref;).
References are commonly used for passing arguments to functions, to create more readable and efficient code.
They are also used for creating aliases to existing variables without the need for pointer-like operations.
```