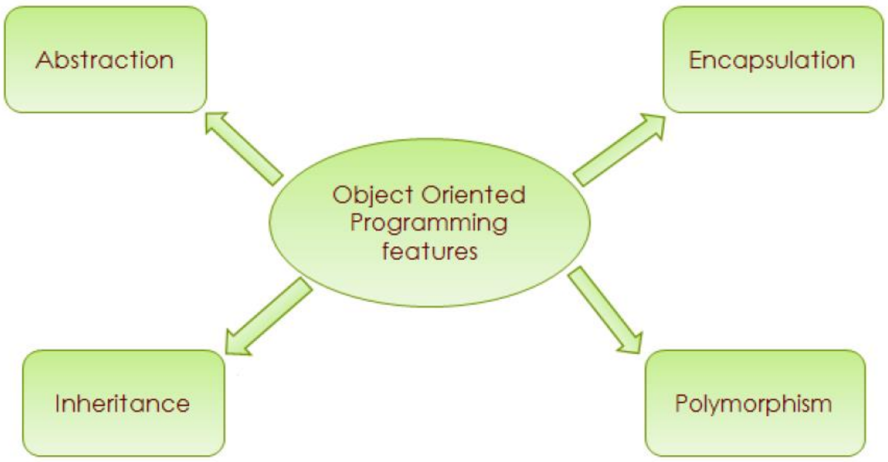
OOP FINAL SHEET-ANSWER

OOP BASIC & (OBJECT & CLASS)

Q-1: What are the Basic Feature Of OOP?

Answer:



Abstraction: Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user.

 Ex: A car is viewed as a car rather than its individual components.

Encapsulation: Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates.

 As in encapsulation, the data in a class is hidden from other classes, so

it is also known as data-hiding.

 Encapsulation can be achieved by declaring all the variables in the class

as private and writing public methods in the class to set and get the values

of variables.

Polymorphism: Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities.

Inheritance: Inheritance is an important pillar of OOP (Object Oriented Programming).It is the mechanism in java by which one class is allow to inherit the features (fields and methods) of another class.

Q-2: Mention the Advantage of OOP.

Answer:

- OOPs makes development and maintenance easier where as in Procedure oriented programming language it is not easy to manage if code grows as project size grows.
- OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
- OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.
- Provide facility of reusability

Q-3: Write Down General form of Class. Difference between class and object with example.

Answer:

```
class Classname {
        type instance-variable1;
        type instance-variable2;
        // ...
        type instance-variableN;
type methodname1(parameter-list) {
        // body of method
        }
type methodname2(parameter-list) {
        // body of method
        }
        // ...
type methodnameN(parameter-list) {
        // body of method
        }
    }
```

Difference between class and object:

No.	Object	Class
1)	Object is an **instance** of a class.	Class is a **blueprint or template** from which objects are created.
2)	Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a **group of similar objects**.
3)	Object is a **physical** entity.	Class is a **logical** entity.
4)	Object is created through **new keyword** mainly e.g. Student s1=new Student();	Class is declared using **class keyword** e.g. class Student{}
5)	Object is created **many times** as per requirement.	Class is declared **once**.
6)	Object **allocates memory when it is created**.	Class **doesn't allocated memory when it is created**.
7)	There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only **one way to define class** in java using class keyword.

Example for Class and Object:

```
class Student {
        int id;//data member (also instance variable)
        String name;//data member(also instance variable)
       }
       class StudentTester {
        public static void main(String args[]){
         Student s1=new Student(); //creating an object of Student
         System.out.println(s1.id);
         System.out.println(s1.name);
 }
}
```

The new keyword is used to allocate memory at runtime.

Q-4: Create a class name 'Student' with String variable 'name' and integer variable 'roll no'. Assign the value the roll number as '2' and that of name as 'John' by creating by object of the class student.

Answer:

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

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

    public static void main(String[] args) {
        Student student = new Student("John", 2);
    }
}
```

Q-5: Is it possible to have array of object? If possible, how? Explain with example. Or How to Create an object array of a class

Answer:

Yes, it is definitely possible to have an array of objects in Java. In fact, arrays of objects are commonly used in Java programs.

Here is an example of how you might create an array of objects in Java:

```
class MyClass
{
int num;
String str;
MyClass [] myArray = new MyClass [5];
myArray [0] = new MyClass ();
myArray[0].num = 10;
myArray[0].str = "hello";
myArray[1] = new MyClass();
myArray[1].num = 20;
myArray[1].str = "world";
```

In this example, we first define a class called MyClass, which has two instance variables: an int called "num" and a String called "str". We then create an array of MyClass objects called myArray with a length of 5.

To add objects to the array, we create new MyClass objects using the "new" keyword and assign them to elements of the array. We can then access the instance variables of each object using the dot notation.

In the example above, we create two MyClass objects and assign them to the first two elements of the array, and set their instance variables. You can continue adding objects to the array in the same way.

<u>To create an object array of a class in Java, you can follow these steps:</u>

I. **Define the class:** First, define the class for the objects you want to store in the array. For example, let's say we want to create an array of Person objects:

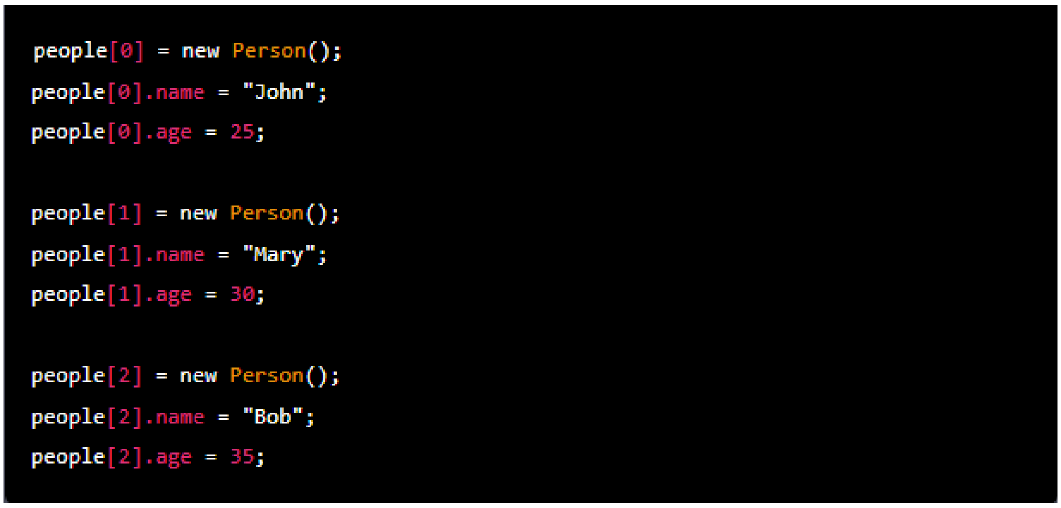
```java
public class Person {
    String name;
    int age;
}
```

II. **Declare the array:** Declare the object array variable with the desired size:

```java
Person[] people = new Person[3];
```

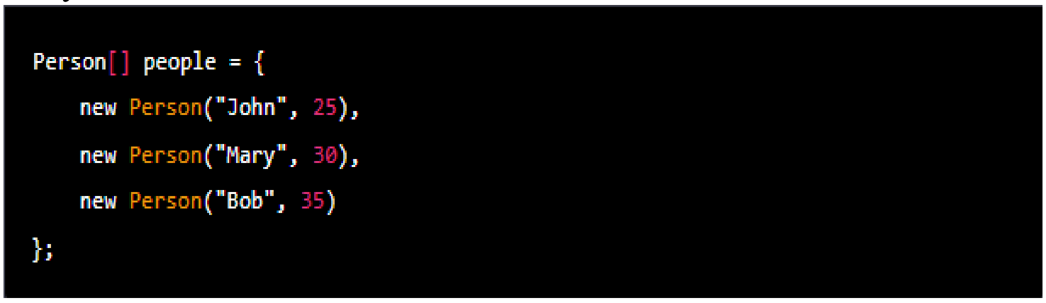
This creates an array of Person objects with a size of 3.

III. **Initialize the array elements:** Initialize each element of the array by creating a new Person object and assigning it to the array:



```
people[0] = new Person();
people[0].name = "John";
people[0].age = 25;

people[1] = new Person();
people[1].name = "Mary";
people[1].age = 30;

people[2] = new Person();
people[2].name = "Bob";
people[2].age = 35;
```

In this example, we create three Person objects and assign them to the elements of the array.
Note that you can also create and initialize the array in one step using an array initializer:



```
Person[] people = {
    new Person("John", 25),
    new Person("Mary", 30),
    new Person("Bob", 35)
};
```

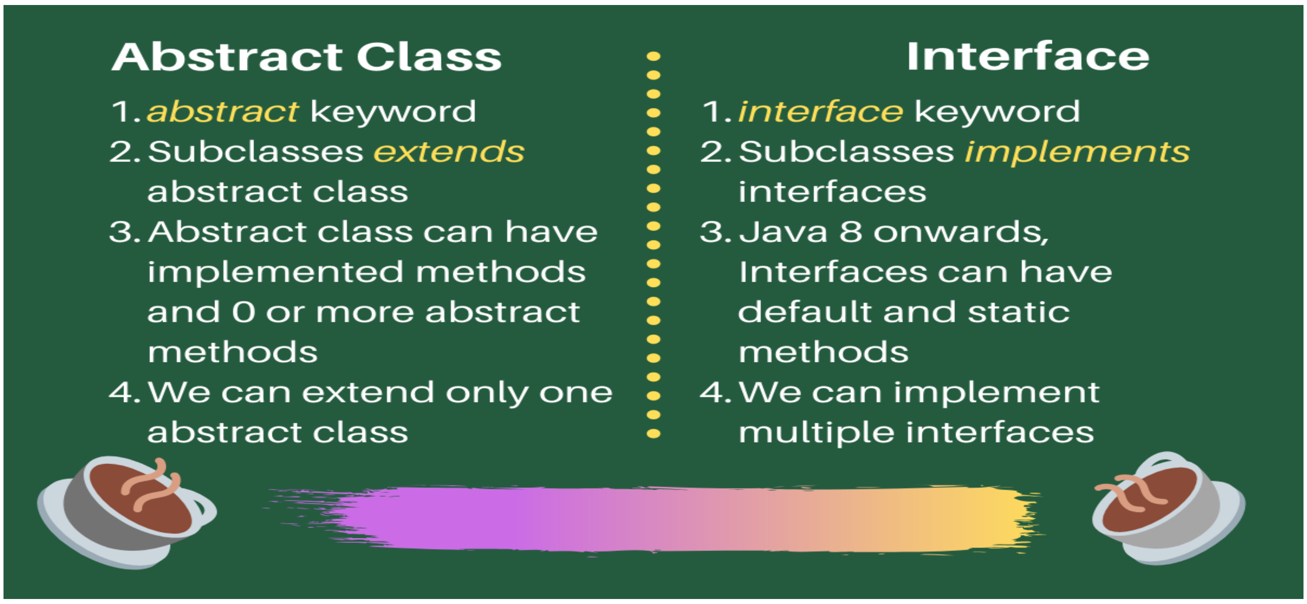
Q-6: Different way to create an object in java.

Answer:

- By new keyword
- By new Instance () method
- By clone () method
- Anonymous object
- multiple objects by one type only

Q-7: What are similarities and differences between java class and java interface?

Answer:



Abstract Class

1. *abstract* keyword
2. Subclasses *extends* abstract class
3. Abstract class can have implemented methods and 0 or more abstract methods
4. We can extend only one abstract class

Interface

1. *interface* keyword
2. Subclasses *implements* interfaces
3. Java 8 onwards, Interfaces can have default and static methods
4. We can implement multiple interfaces

Relationship between java class & java interface

Relationship:

On the other hand, a Java interface is a type that defines a set of methods (without implementations) that a class must implement if it wants to conform to that interface. An interface provides a way to specify a contract between multiple classes, defining a common set of methods that these classes agree to implement.

অন্যদিকে, একটি জাভা ইন্টারফেস হল এমন একটি ধরন যা পদ্ধতির একটি সেট (বাস্তবায়ন ছাড়াই) সংজ্ঞায়িত করে যেটি একটি ক্লাস যদি সেই ইন্টারফেসের সাথে সামঞ্জস্য করতে চায় তাহলে তাকে অবশ্যই প্রয়োগ করতে হবে। একটি ইন্টারফেস একাধিক শ্রেণীর মধ্যে একটি চুক্তি নির্দিষ্ট করার একটি উপায় প্রদান করে, একটি সাধারণ সেটকে সংজ্ঞায়িত করে যা এই ক্লাসগুলি বাস্তবায়ন করতে সম্মত হয়।

Both classes and interfaces can be used to define types in Java, but they serve different purposes. Whereas a class typically represents a concrete entity with a specific implementation, an interface is a contract that defines a set of methods that a class must implement. A class can implement multiple interfaces, but can extend only one class.

উভয় ক্লাস এবং ইন্টারফেস জাভাতে প্রকারগুলি সংজ্ঞায়িত করতে ব্যবহার করা যেতে পারে, তবে তারা বিভিন্ন উদ্দেশ্যে পরিবেশন করে। যেখানে একটি শ্রেণী সাধারণত একটি নির্দিষ্ট বাস্তবায়নের সাথে একটি কংক্রিট সত্তার প্রতিনিধিত্ব করে, একটি ইন্টারফেস হল একটি চুক্তি যা একটি শ্রেণীকে অবশ্যই প্রয়োগ করতে হবে এমন পদ্ধতির

একটি সেট সংজ্ঞায়িত করে। একটি ক্লাস একাধিক ইন্টারফেস বাস্তবায়ন করতে পারে, কিন্তু শুধুমাত্র একটি শ্রেণী প্রসারিত করতে পারে।

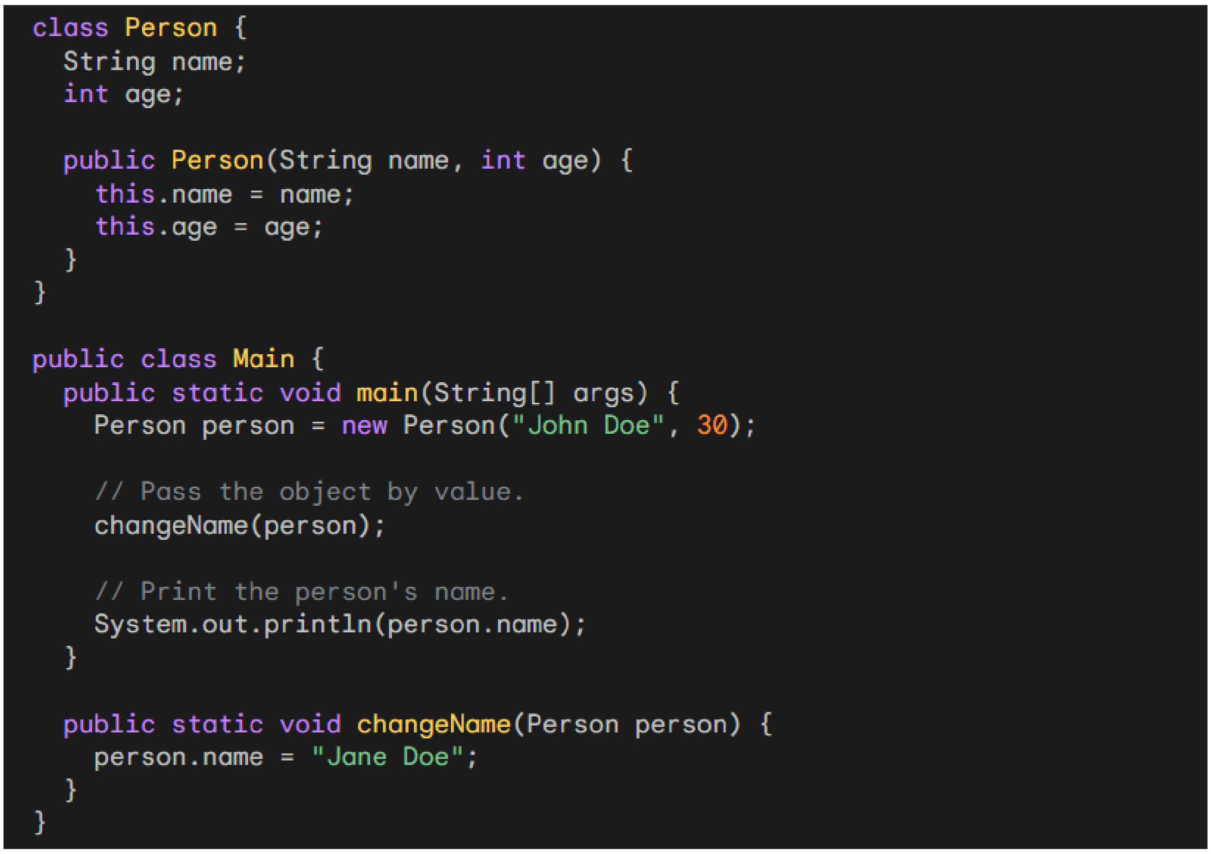
Q-8: How can object be passed to the function?

Answer:
In Java, objects can be passed to functions in two ways: by value and by reference.

Passing by value means that a copy of the object is passed to the function. Any changes made to the object inside the function will not be reflected in the original object.

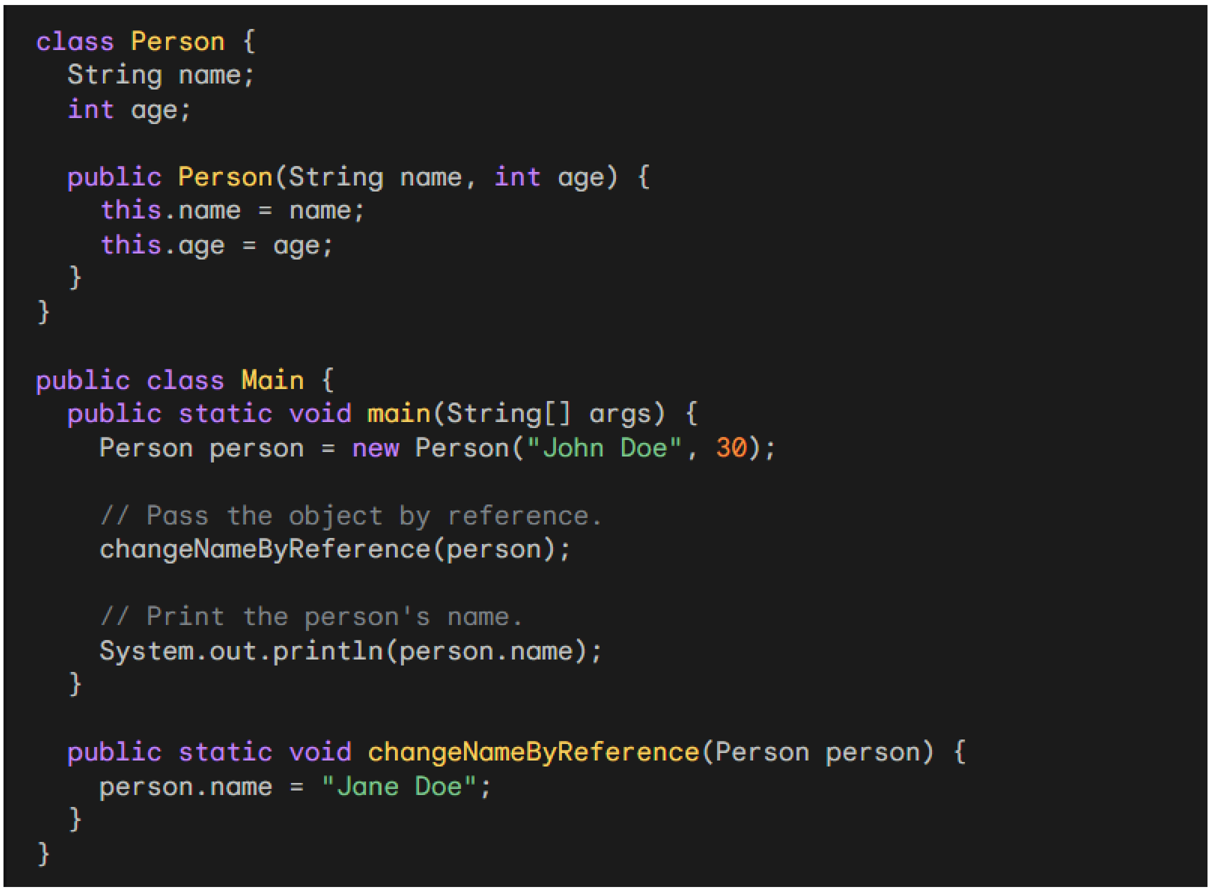
Passing by reference means that the function is passed a reference to the object. Any changes made to the object inside the function will be reflected in the original object.

The following code shows how to pass an object to a function by value in Java:



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

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}

public class Main {
  public static void main(String[] args) {
    Person person = new Person("John Doe", 30);

    // Pass the object by value.
    changeName(person);

    // Print the person's name.
    System.out.println(person.name);
  }

  public static void changeName(Person person) {
    person.name = "Jane Doe";
  }
}
```

The following code shows how to pass an object to a function by reference in Java:



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

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}

public class Main {
  public static void main(String[] args) {
    Person person = new Person("John Doe", 30);

    // Pass the object by reference.
    changeNameByReference(person);

    // Print the person's name.
    System.out.println(person.name);
  }

  public static void changeNameByReference(Person person) {
    person.name = "Jane Doe";
  }
}
```

Q-9:What are bounded types in java generics? provide an example of a generic class and demonstrate how it can be instantiated with different types in Java?

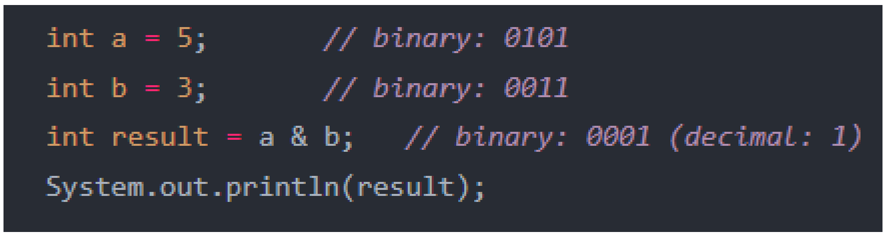
Answer:

Q-10:What do you know about JAVA Bitwise Operators? Give some appropriate example.

Answer:

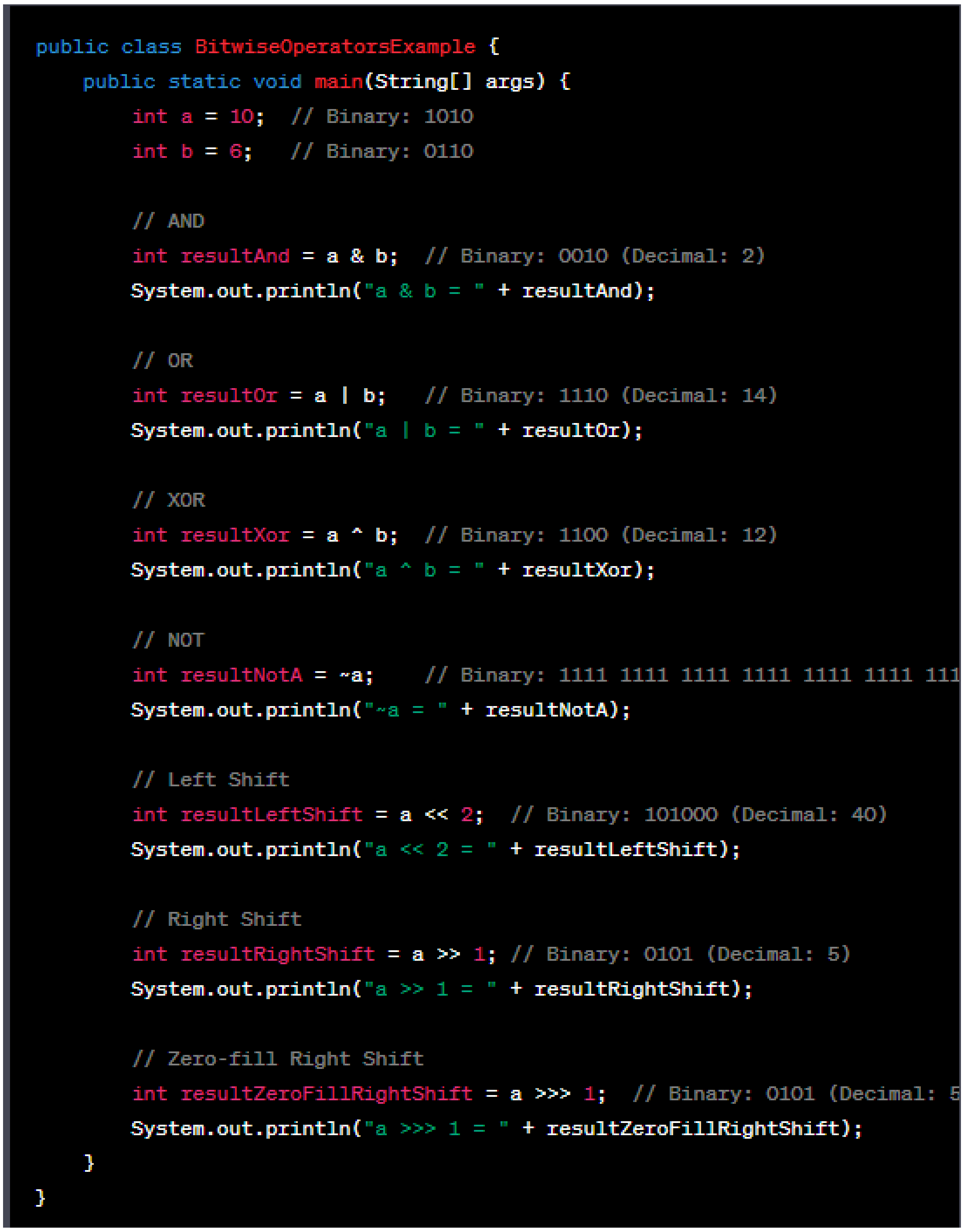
Java Bitwise Operators are used to perform operations on individual bits of integer values. These operators work at the bit level, manipulating the binary representations of numbers. Java provides several bitwise operators:

1. AND (&): Performs a bitwise AND operation between each corresponding bit of two integers. If both bits are 1, the resulting bit is 1; otherwise, it's 0.
2. OR (|): Performs a bitwise OR operation between each corresponding bit of two integers. If either of the bits is 1, the resulting bit is 1; otherwise, it's 0.
3. XOR (^): Performs a bitwise exclusive OR operation between each corresponding bit of two integers. If the bits are different (one is 1 and the other is 0), the resulting bit is 1; otherwise, it's 0.
4. NOT (~): Performs a bitwise NOT operation, which flips each bit of an integer. If the bit is 1, it becomes 0, and vice versa.
5. Left Shift (<<): Shifts the bits of the left-hand operand to the left by the number of positions specified by the right-hand operand. The leftmost bits are filled with zeros.
6. Right Shift (>>): Shifts the bits of the left-hand operand to the right by the number of positions specified by the right-hand operand. The rightmost bits are filled according to the sign bit (for signed integers).
7. Zero-fill Right Shift (>>>): Similar to the right shift operator (>>), but the leftmost bits are filled with zeros, even for signed integers.



```java
int a = 5;        // binary: 0101
int b = 3;        // binary: 0011
int result = a & b;    // binary: 0001 (decimal: 1)
System.out.println(result);
```

Or,

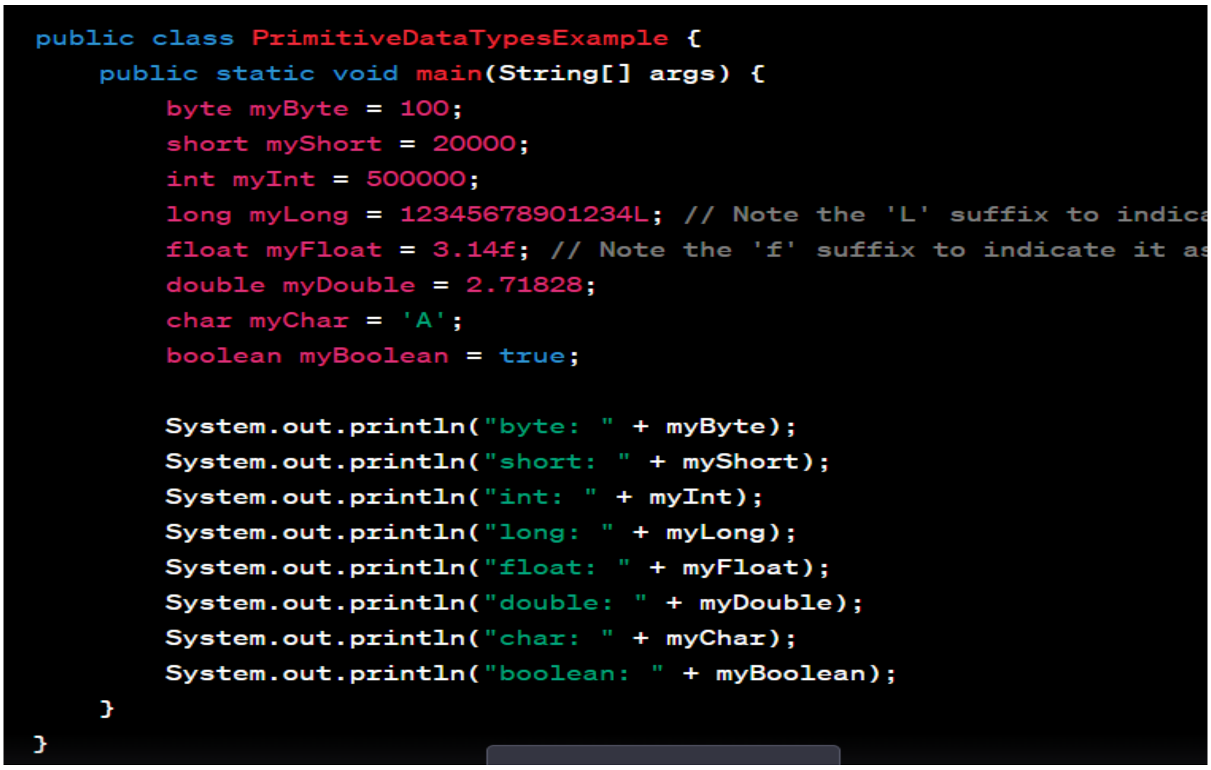


```java
public class BitwiseOperatorsExample {
    public static void main(String[] args) {
        int a = 10;  // Binary: 1010
        int b = 6;   // Binary: 0110

        // AND
        int resultAnd = a & b;  // Binary: 0010 (Decimal: 2)
        System.out.println("a & b = " + resultAnd);

        // OR
        int resultOr = a | b;   // Binary: 1110 (Decimal: 14)
        System.out.println("a | b = " + resultOr);

        // XOR
        int resultXor = a ^ b;  // Binary: 1100 (Decimal: 12)
        System.out.println("a ^ b = " + resultXor);

        // NOT
        int resultNotA = ~a;    // Binary: 1111 1111 1111 1111 1111 1111 111
        System.out.println("~a = " + resultNotA);

        // Left Shift
        int resultLeftShift = a << 2;  // Binary: 101000 (Decimal: 40)
        System.out.println("a << 2 = " + resultLeftShift);

        // Right Shift
        int resultRightShift = a >> 1; // Binary: 0101 (Decimal: 5)
        System.out.println("a >> 1 = " + resultRightShift);

        // Zero-fill Right Shift
        int resultZeroFillRightShift = a >>> 1;  // Binary: 0101 (Decimal: 5
        System.out.println("a >>> 1 = " + resultZeroFillRightShift);
    }
}
```

Q-11:In JAVA programming language how many of Primitive Data Types we have? Briefly Explain whatever you know about them.

Answer: there are 8 primitive data types in Java:

- byte - A byte is an 8-bit signed integer. It can store values from -128 to 127.
- short - A short is a 16-bit signed integer. It can store values from -32,768 to 32,767.
- int - An int is a 32-bit signed integer. It can store values from -2,147,483,648 to 2,147,483,647.
- long - A long is a 64-bit signed integer. It can store values from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
- float - A float is a 32-bit floating-point number. It can store values with a precision of up to 7 decimal places.
- double - A double is a 64-bit floating-point number. It can store values with a precision of up to 15 decimal places.
- boolean - A boolean is a 1-bit data type. It can store only two values: true or false.
- char - A char is a 16-bit unsigned integer. It can store a single character from the Unicode character set.



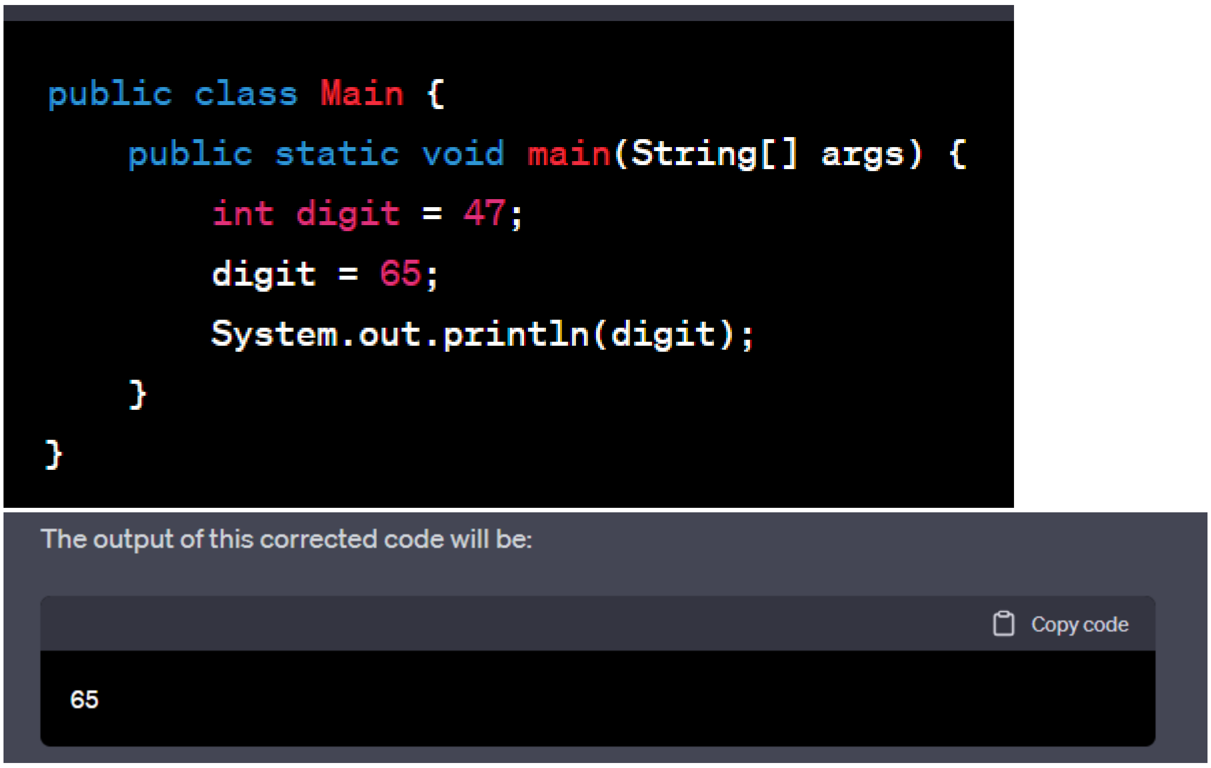
```
public class PrimitiveDataTypesExample {
    public static void main(String[] args) {
        byte myByte = 100;
        short myShort = 20000;
        int myInt = 500000;
        long myLong = 12345678901234L; // Note the 'L' suffix to indica
        float myFloat = 3.14f; // Note the 'f' suffix to indicate it as
        double myDouble = 2.71828;
        char myChar = 'A';
        boolean myBoolean = true;

        System.out.println("byte: " + myByte);
        System.out.println("short: " + myShort);
        System.out.println("int: " + myInt);
        System.out.println("long: " + myLong);
        System.out.println("float: " + myFloat);
        System.out.println("double: " + myDouble);
        System.out.println("char: " + myChar);
        System.out.println("boolean: " + myBoolean);
    }
}
```

Q-12:Write down the output of following code:

a) public class Main { public static void main(string args[]) int digit = 47{; digit = 65; System.out.println(digit); }	b) public class Main (public static void main (String args[]) { int Hello World= 1111; System.out.println("Hello World"); System.out.println("Welcome To MEC"); //System.out.println(Hello World); System.out.println("Hello World");

Answer:

a)

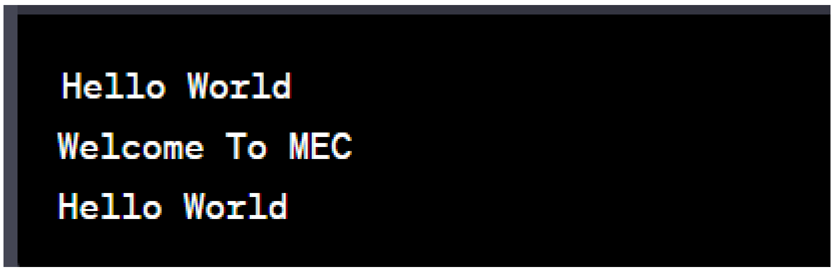


```
public class Main {
    public static void main(String[] args) {
        int digit = 47;
        digit = 65;
        System.out.println(digit);
    }
}
```

The output of this corrected code will be:

```
65
```

b)



```
Hello World
Welcome To MEC
Hello World
```

Q-11:How you can assign object a reference variable? Give proper explanation.

Answer:

In most programming languages, including Java, you can assign an object to a reference variable using the following steps:

1. Declare the reference variable: Start by declaring a variable of the appropriate type that will hold the reference to the object. For example, if you have a class called "Person", you can declare a reference variable of type "Person" like this: Person personObj;
2. Create an instance of the object: Use the new keyword followed by the constructor of the class to create a new instance of the object. For example, to create a new instance of the "Person" class, you can use: personObj = new Person();
3. Assign the reference: Finally, assign the reference of the newly created object to the reference variable using the assignment operator (=). This establishes a connection between the reference variable and the object in memory. For example: personObj = new Person();

<u>Here's a complete example:</u>

```
public class Person {
   private String name;

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

   public String getName() {
      return name;
   }
}

public class Main {
   public static void main(String[] args) {
      // Step 1: Declare the reference variable
      Person personObj;

      // Step 2: Create an instance of the object
      personObj = new Person("John");

      // Step 3: Assign the reference
      System.out.println(personObj.getName()); // Output: John
   }
}
```

Q-12:What is instance variable hiding? Write the proper method to solve this problem.

Answer:

Instance variable hiding is a phenomenon in Java where an instance variable in a subclass has the same name as an instance variable in its superclass. When this happens, the instance variable in the subclass hides the instance variable in the superclass. This means that if you access the instance variable using the subclass object, the instance variable in the subclass will be used, even if the superclass object has an instance variable with the same name.

There are two ways to solve the problem of instance variable hiding:

- Use the super keyword to access the instance variable in the superclass. For example, the following code shows how to use the super keyword to access the instance variable name in the superclass Person:

```
class Student extends Person {

    private String studentId;

    public Student(String name, int age, String studentId) {
        super(name, age);
        this.studentId = studentId;
    }

    public void sayHello() {
        System.out.println("Hello, my name is " + super.name + " and I am " + super.age + " years old. My student ID is
    }
}
```

- Rename the instance variable in the subclass. This is the simplest way to solve the problem of instance variable hiding, but it can make the code more difficult to read and understand.

Here is an example of how to rename the instance variable in the subclass:

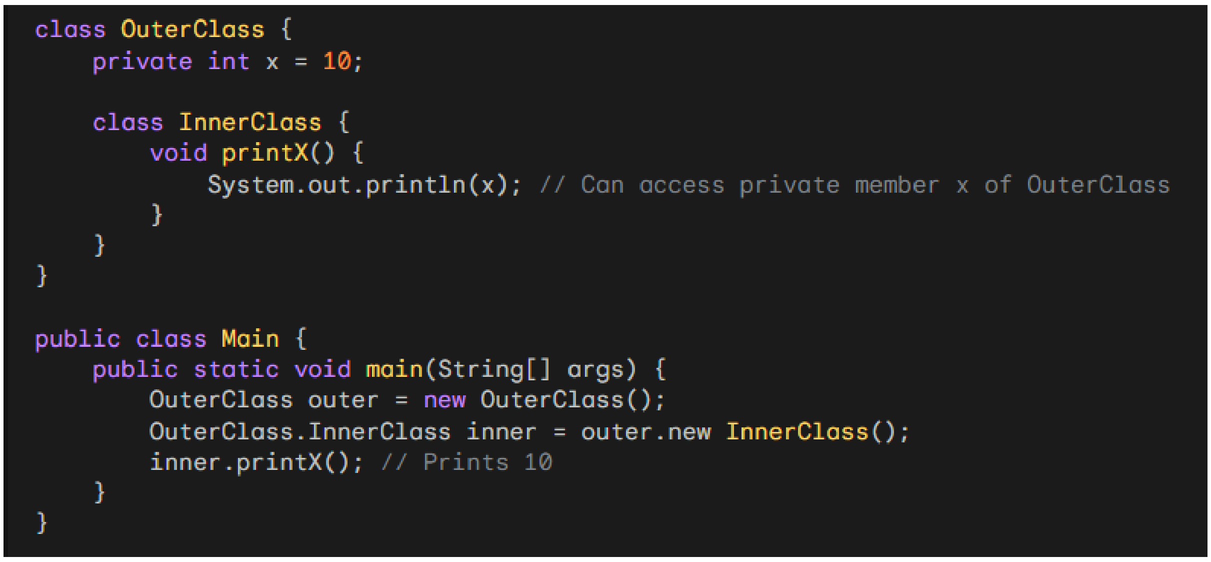
```
class Student extends Person {

    private String studentName;

    public Student(String name, int age, String studentId) {
        super(name, age);
        this.studentName = name;
    }

    public void sayHello() {
        System.out.println("Hello, my name is " + studentName + " and I am " + age + " years old. My student ID is " +
    }
}
```

Q-13:Discribe inner class and outer class with suitable code example.

Answer:

An inner class is a class that is declared within another class. The class that contains the inner class is called the outer class. Inner classes can access all the members of the outer class, including private members.

Here is an example of an inner class in Java:

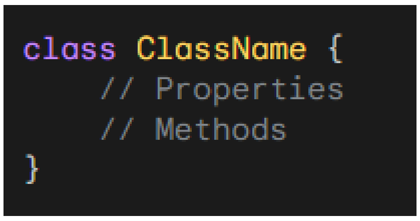


```
class OuterClass {
    private int x = 10;

    class InnerClass {
        void printX() {
            System.out.println(x); // Can access private member x of OuterClass
        }
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        OuterClass.InnerClass inner = outer.new InnerClass();
        inner.printX(); // Prints 10
    }
}
```

Q-14:Define class and object with there syntax.

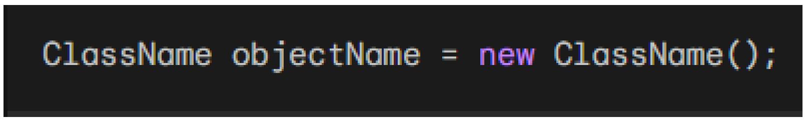
Answer:

In Java, a class is a blueprint for creating objects. It defines the properties and behaviors of an object. The syntax for declaring a class in Java is:



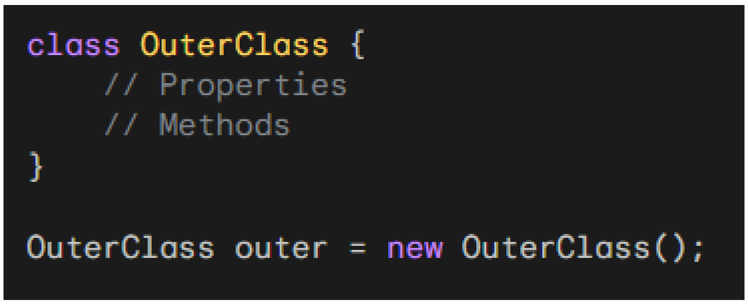
```
class ClassName {
    // Properties
    // Methods
}
```

An object is an instance of a class. It is a concrete representation of the class. The syntax for creating an object in Java is:



```
ClassName objectName = new ClassName();
```

For example, the following code declares a class called OuterClass and creates an object called outer:



```
class OuterClass {
    // Properties
    // Methods
}

OuterClass outer = new OuterClass();
```

Q-15:Discuss Dynamic Binding in java with code example.

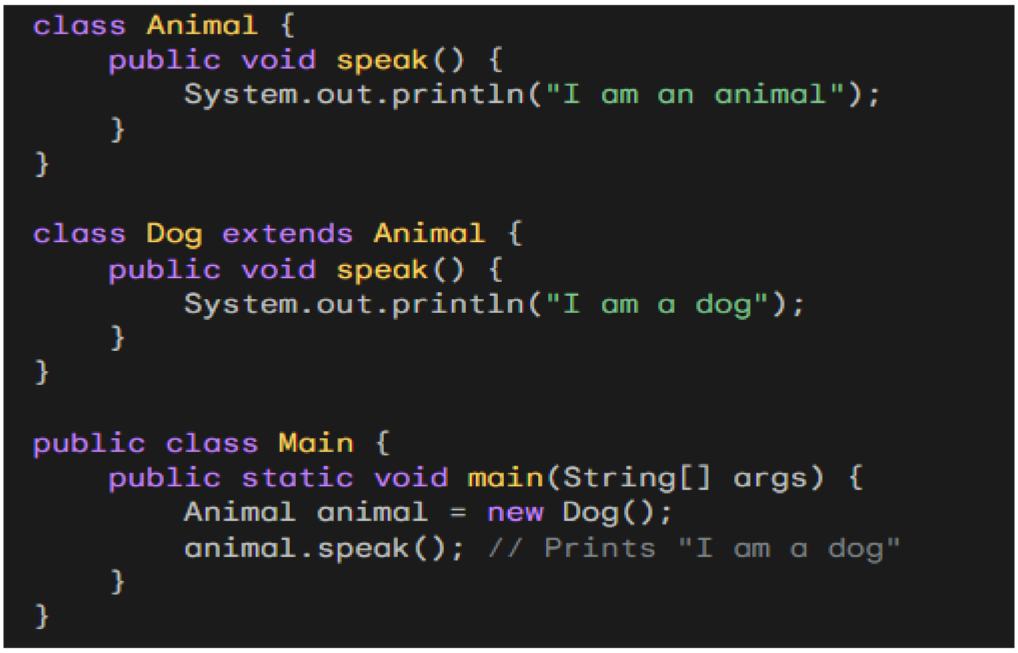
Answer:

Dynamic binding is a feature of Java that allows the type of an object to be determined at runtime. This is in contrast to static binding, where the type of an object is determined at compile time.

Dynamic binding is used in Java for a variety of purposes, including:

- Polymorphism: The ability of an object to behave differently depending on its type.
- Method overriding: The ability for a subclass to override a method from its superclass.
- Runtime type information (RTTI): The ability to get information about the type of an object at runtime.

Here is an example of dynamic binding in Java:



```
class Animal {
    public void speak() {
        System.out.println("I am an animal");
    }
}

class Dog extends Animal {
    public void speak() {
        System.out.println("I am a dog");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Dog();
        animal.speak(); // Prints "I am a dog"
    }
}
```

Constructor

Q-1: Define Constructor/Copy constructor and Destructor with example. How Constructor is different from a normal member function?

Answer:

Definition Constructor: A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. **Example:**

```
public class Main {
  int x;
  public Main () {
    x = 5; // Set the initial value for the class attribute x
  }
  public static void main (String [] args) {
    Main myObj = new Main (); // Create an object of class Main (This will call
the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}
// Outputs 5
```

Definition Copy Constructor: Java copy constructor is a particular type of constructor that we use to create a duplicate (exact copy) of the existing object of a class. Copy constructors create and return a new object using the existing object of a class.

Example:

```
public class Student {
        private int roll;
        private String name;
        // Creating a copy constructor by passing the parameter as the class
        public Student (Student student) {
        // copying each field of the existing object into the newly created object
        this. roll = student.Roll;
        this.name = student.name;
        }
}
```

Definition Destructor: A destructor in Java is a special method that gets called automatically as soon as the life cycle of an object is finished. A destructor is called to de-allocate and free memory. The following tasks get executed when a destructor is called. Destructors in Java, also known as finalizers are non-deterministic.

Example:

```
public class DestructorExample
{
public static void main (String [] args)
{
DestructorExample de = new DestructorExample ();
de.finalize();
de = null;
System.gc();
System.out.println("Inside the main () method");
}
protected void finalize ()
{
System.out.println("Object is destroyed by the Garbage Collector");
}
}
```

Constructor is different from a normal member function

In Java, a constructor is a special type of method that is used to initialize objects of a class. It is different from a normal member function in the following ways:

- **Name**: The name of the constructor is the same as the name of the class, while the name of a normal member function can be anything.
- **Return Type**: Constructors do not have a return type, not even void. Normal member functions, on the other hand, must have a return type, which can be any valid data type or void.
- **Invocation:** Constructors are invoked automatically when an object is created, while normal member functions are invoked explicitly by calling them using the object reference.
- **Accessibility:** Constructors can have any access level, while normal member functions can have public, private, protected, or package-private access.
- **Usage:** Constructors are primarily used to initialize the state of an object, while normal member functions are used to perform various operations on objects.
- **Overloading:** Constructors can be overloaded in a class, meaning that multiple constructors can exist in the same class with different parameters. Normal member functions can also be overloaded in a class.

Overall, constructors play a unique role in initializing objects of a class and are essential to the creation of new objects in Java.

Q-2: Can we have both default constructor and parameterterized constructor in the same class?

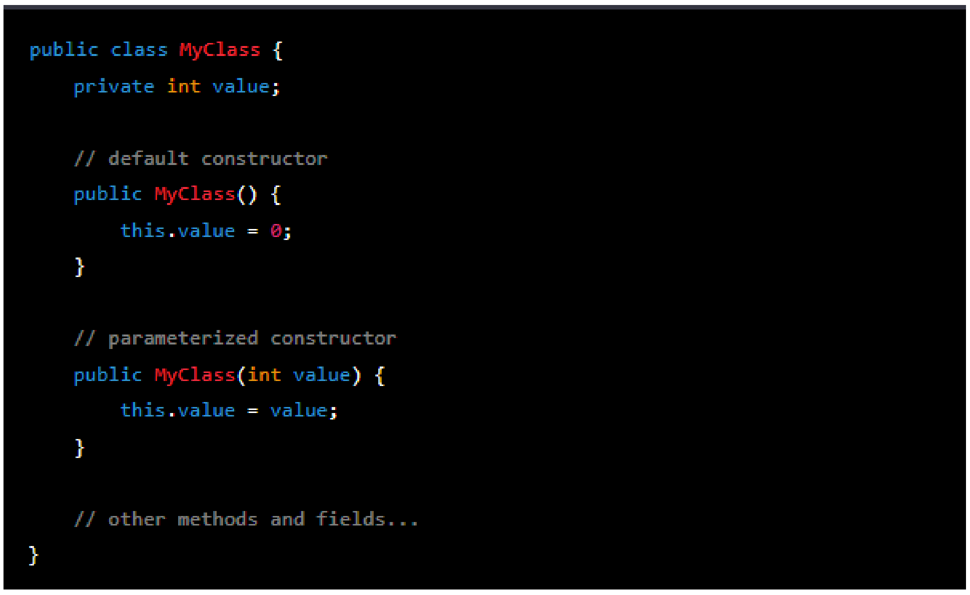
Answer:

Yes, it is possible to have both a default constructor and a parameterized constructor in the same class in Java.

A default constructor is a constructor that takes no arguments and is provided by the Java compiler if no other constructor is defined in the class. A parameterized constructor, on the other hand, is a constructor that takes one or more arguments.

If we define a parameterized constructor in a class, the Java compiler will not provide a default constructor automatically. However, we can still define a default constructor explicitly in the same class if you need to.

Here is an example of a Java class with both a default constructor and a parameterized constructor:

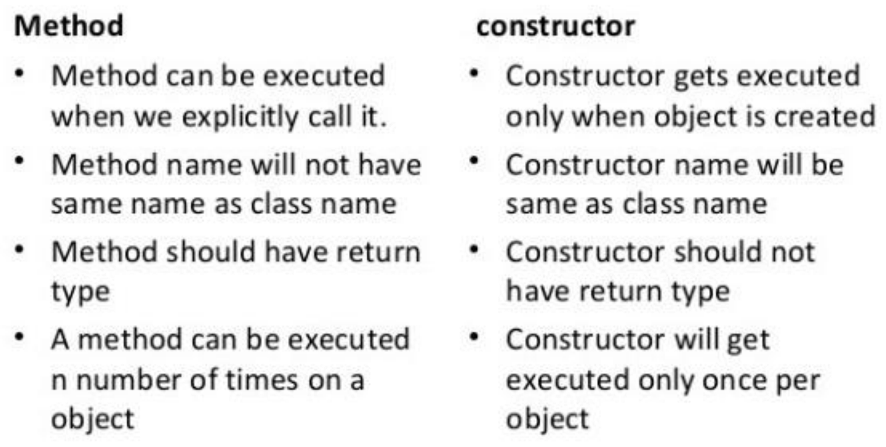


```java
public class MyClass {
    private int value;

    // default constructor
    public MyClass() {
        this.value = 0;
    }

    // parameterized constructor
    public MyClass(int value) {
        this.value = value;
    }

    // other methods and fields...
}
```

In this example, the default constructor sets the value field to 0, while the parameterized constructor sets it to the value passed as an argument. You can use either constructor to create objects of the MyClass class, depending on your needs.

Q-3: Difference between Method & Constructor.

Answer:



Method	constructor
• Method can be executed when we explicitly call it.	• Constructor gets executed only when object is created
• Method name will not have same name as class name	• Constructor name will be same as class name
• Method should have return type	• Constructor should not have return type
• A method can be executed n number of times on a object	• Constructor will get executed only once per object

Q-4: Explain the concept of operator overloading. What are the ways to overload an operator in java.

Answer:

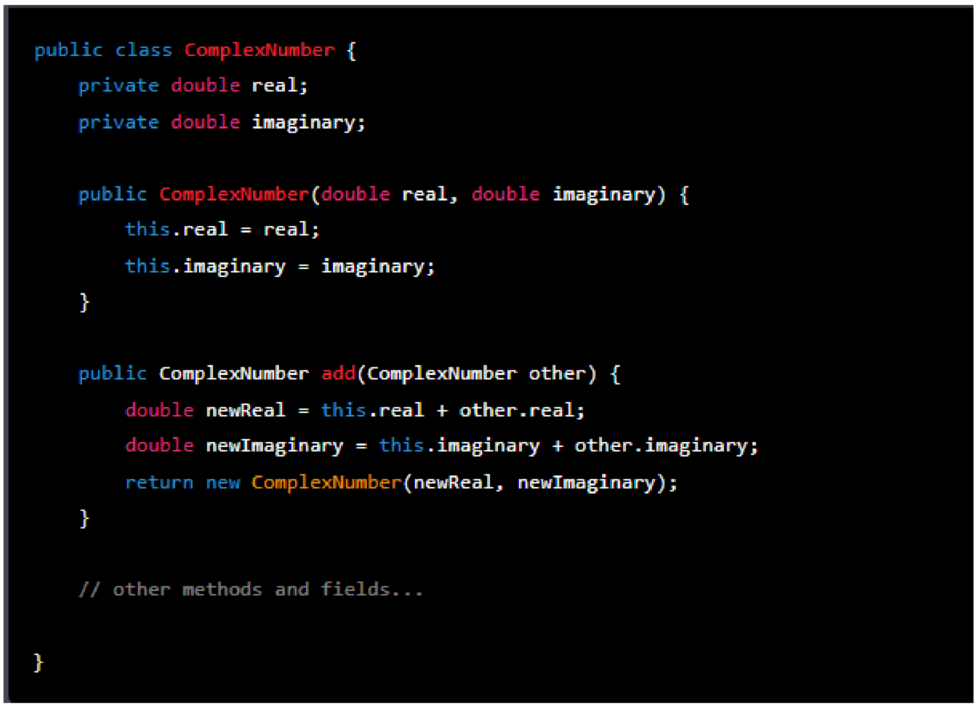
Operator overloading is a feature in object-oriented programming that allows operators, such as +, -, *, /, %, ==,!=, <, >, <=, >=, etc., to be given extended meanings when applied to objects of a certain class. In other words, it allows a class to define how operators should behave when used with objects of that class.

In Java, operator overloading is not supported for most operators, except for a few operators such as + and - for string concatenation and subtraction of numbers, respectively. However, Java allows you to create your own classes and define how operators should behave when used with objects of those classes.

<u>There are two ways to overload operators in Java:</u>

Method Overloading: Java allows you to overload methods, including special methods such as the "equals" method, which is used for object comparison, and the "compare To" method, which is used for object sorting. By overloading these methods, you can define how operators such as ==, !=, <, >, <=, >=, etc., should behave when used with objects of your class.

For example, here is how you can overload the "+" operator for a custom class named "ComplexNumber" that represents a complex number:



```java
public class ComplexNumber {
    private double real;
    private double imaginary;

    public ComplexNumber(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    public ComplexNumber add(ComplexNumber other) {
        double newReal = this.real + other.real;
        double newImaginary = this.imaginary + other.imaginary;
        return new ComplexNumber(newReal, newImaginary);
    }

    // other methods and fields...

}
```

In this example, the add() method is used to overload the "+" operator. When called, it returns a new Complex Number object that represents the sum of two Complex Number objects.

Operator Overloading Using Third-Party Libraries: Java also allows you to overload operators using third-party libraries such as Javolution, JScience, and Apache Commons Math. These libraries provide classes and methods that allow you to perform mathematical operations on objects of your class.

For example, the Javolution library provides a class named "FastMath" that contains methods for performing mathematical operations such as addition, subtraction, multiplication, and division on objects of various classes, including custom classes.

```java
import javolution.lang.*;

public class ComplexNumber extends Struct {
    public double real;
    public double imaginary;

    public ComplexNumber(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    // other methods and fields...

    public ComplexNumber plus(ComplexNumber other) {
        ComplexNumber result = new ComplexNumber(0, 0);
        FastMath.plus(this, other, result);
        return result;
    }
}
```

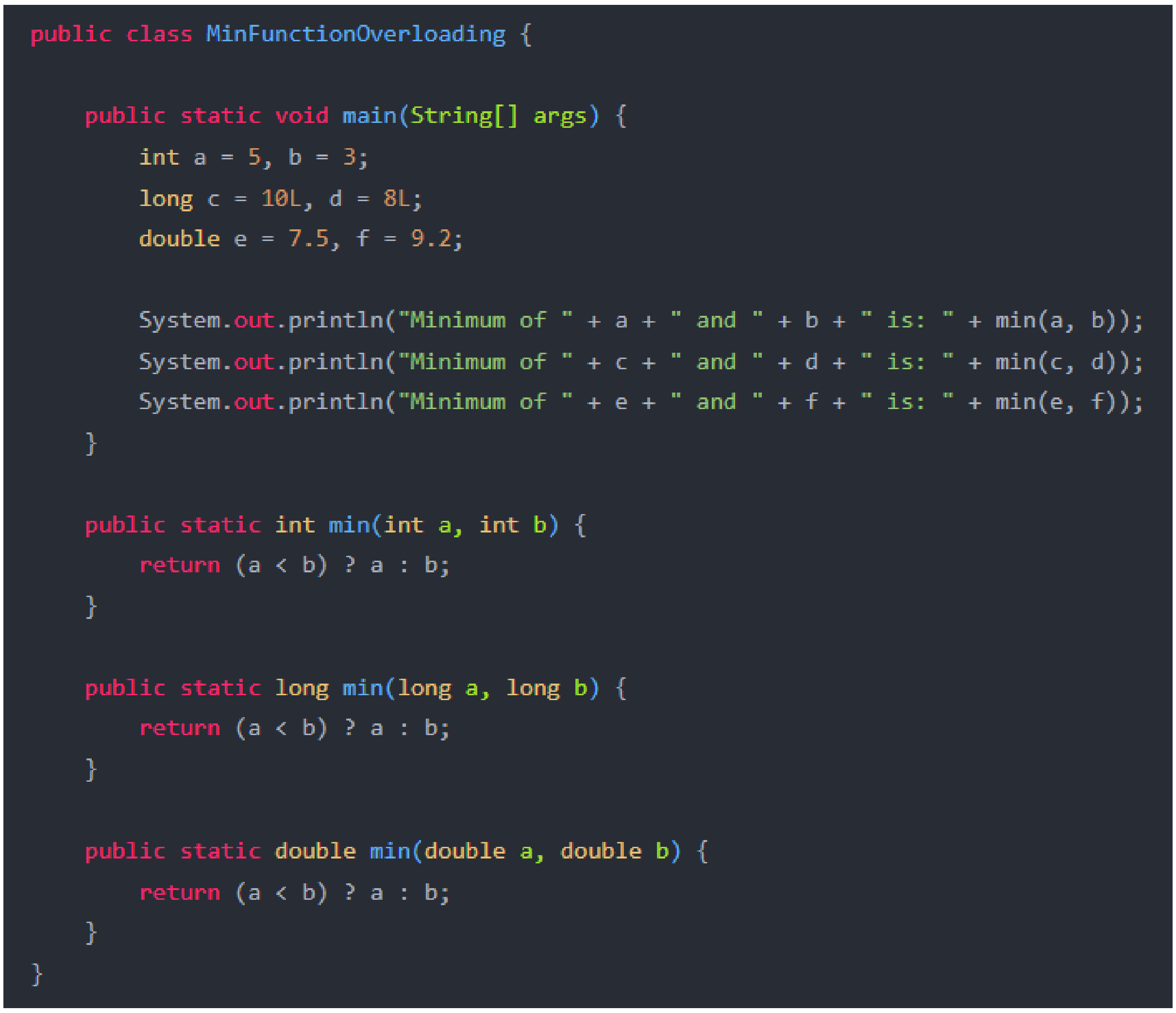
In this example, the plus () method uses the "FastMath. Plus ()" method to perform addition on two ComplexNumber objects. The "Struct" class is extended to make ComplexNumber objects mutable, which is required by the Javolution library.

Note that operator overloading can make your code more concise and readable, but it can also make your code more complex and harder to understand if not used judiciously. Therefore, it is important to use operator overloading only when it makes sense and to follow good coding practices when doing so.

Q-5: Write a program to find out the minimum of two given numbers. The numbers can be integer, long and double. Overload a min () function to handle those three types of data.

Answer:

Sure, here's a Java program that overloads the `min()` function to find the minimum of two given numbers of type `int`, `long` or `double`:



```java
public class MinFunctionOverloading {

    public static void main(String[] args) {
        int a = 5, b = 3;
        long c = 10L, d = 8L;
        double e = 7.5, f = 9.2;

        System.out.println("Minimum of " + a + " and " + b + " is: " + min(a, b));
        System.out.println("Minimum of " + c + " and " + d + " is: " + min(c, d));
        System.out.println("Minimum of " + e + " and " + f + " is: " + min(e, f));
    }

    public static int min(int a, int b) {
        return (a < b) ? a : b;
    }

    public static long min(long a, long b) {
        return (a < b) ? a : b;
    }

    public static double min(double a, double b) {
        return (a < b) ? a : b;
    }
}
```

Q-6: A constructure is a special function in a class. Explain why a constructor is different than other member function. give example.

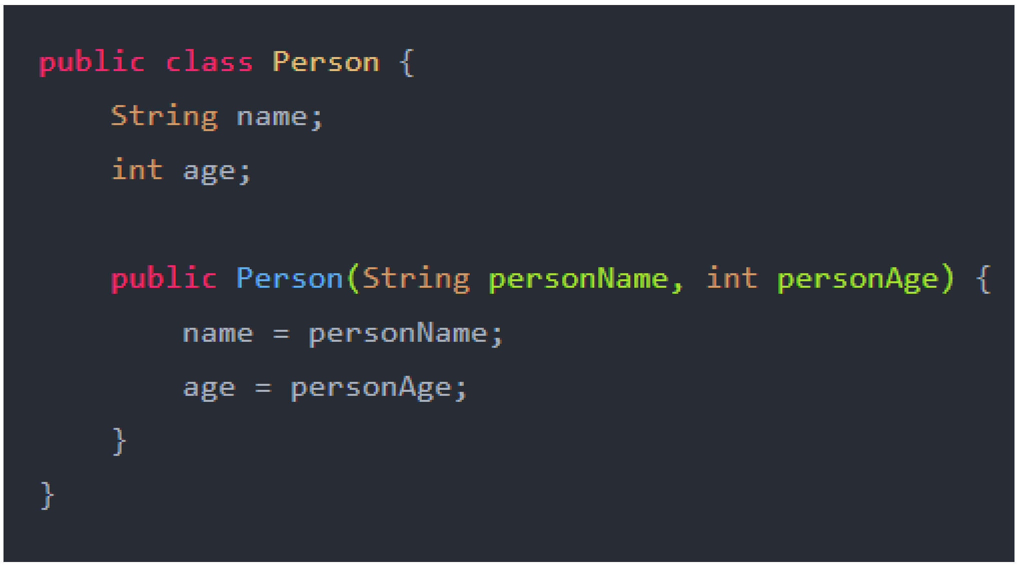
Answer:

A constructor is a special method in a Java class that is responsible for initializing the state of an object. Unlike other member functions, which are invoked explicitly by the programmer, constructors are automatically called when an object is created using the new keyword.

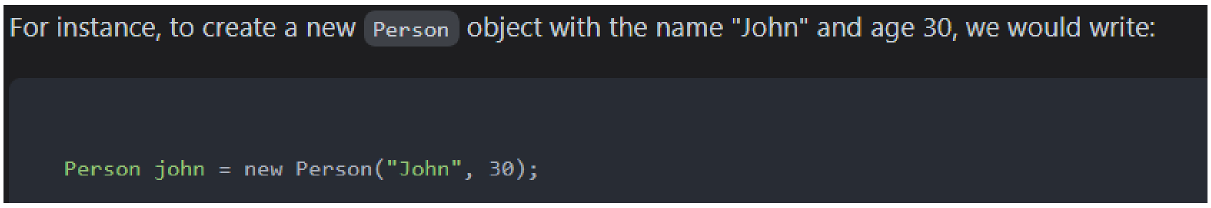
Here are some key differences between constructors and other member functions:

- ➢ Name: Constructors always have the same name as the class they belong to, whereas other member functions can have any valid identifier as their name.
- ➢ Return type: Constructors do not have a return type, not even void, whereas other member functions must have a return type or void.
- ➢ Parameters: Constructors can take parameters, but their types and order must match those declared in the constructor definition. Other member functions can also take parameters, but their names, types, and order can differ from the function definition.

Here's an example of a constructor in a Java class:



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

    public Person(String personName, int personAge) {
        name = personName;
        age = personAge;
    }
}
```



For instance, to create a new `Person` object with the name "John" and age 30, we would write:

```java
Person john = new Person("John", 30);
```

Q-7: Define constructor overloading? What is the reason of overload constructor.

Answer:

In Java, constructor overloading is the concept of having more than one constructor with different parameter lists in the same class.

Reasons for constructor overloading:

Initialization flexibility: Constructor overloading provides flexibility in initializing objects by accepting different sets of parameters. It allows objects to be created with different initial states based on the provided arguments.

Convenience: Overloaded constructors provide convenience to the users of a class by offering different ways to create objects. Users can choose the constructor that suits their needs or provides the most convenient way to initialize the object.

Encapsulation and abstraction: Overloaded constructors can be used to encapsulate and abstract the complexity of object creation. They can hide the details of initialization and provide a simplified interface for creating objects.

Code reusability: By overloading constructors, common initialization code can be shared among different constructors, reducing code duplication. This promotes code reusability and maintenance.

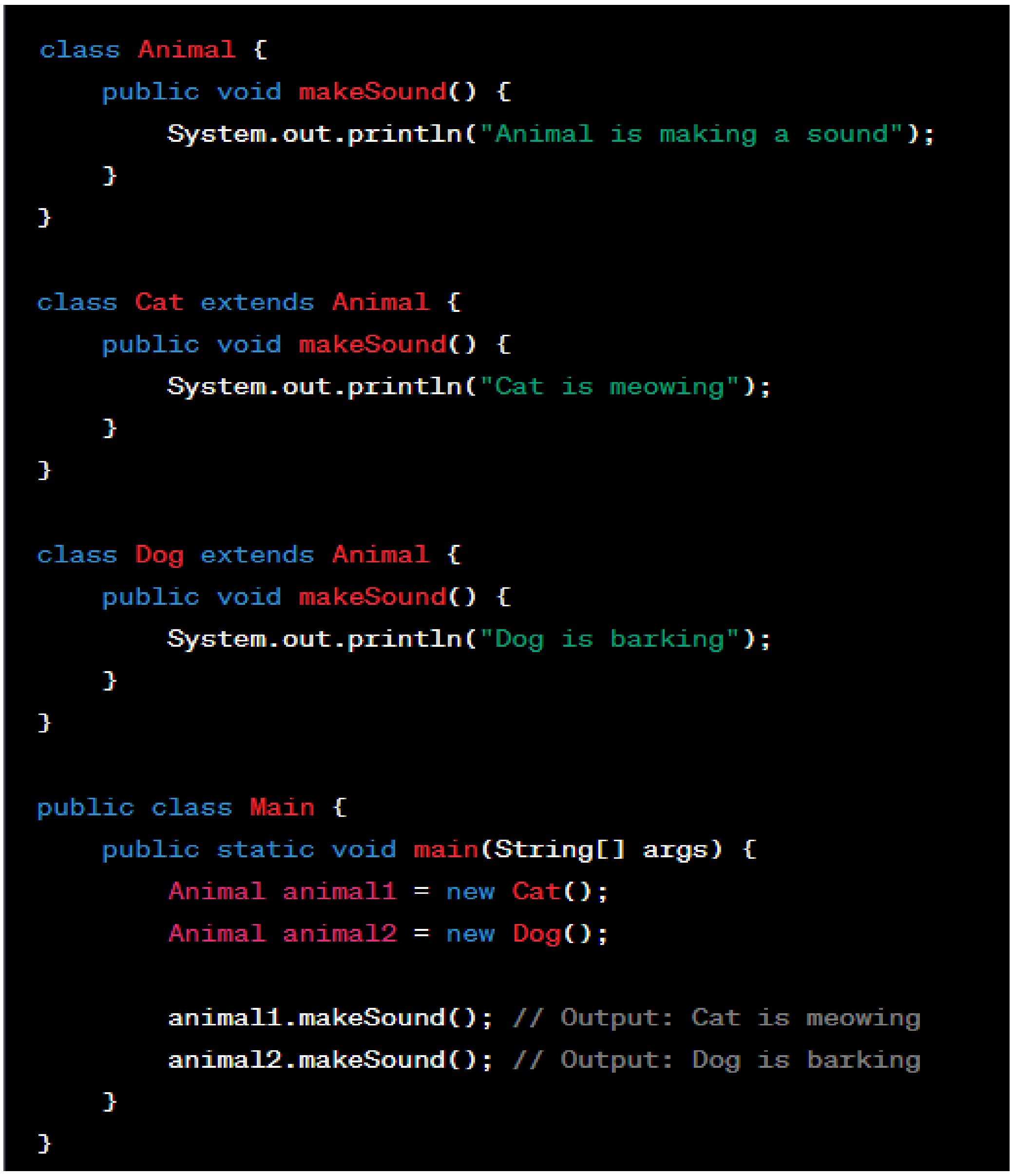
Polymorphism: Constructor overloading contributes to polymorphism, allowing objects to be created using different constructor signatures. This supports the principle of "one interface, multiple implementations" in object-oriented programming.

Q-8: What is dynamic method dispatch? illustrated with example. (Uses)

Answer:

Dynamic method dispatch in Java refers to the mechanism where the selection of the method to be executed at runtime is based on the actual object being referred to, rather than the type of the reference variable. It allows for method overriding to be resolved dynamically during program execution.

Here's an example to illustrate dynamic method dispatch:



```java
class Animal {
    public void makeSound() {
        System.out.println("Animal is making a sound");
    }
}

class Cat extends Animal {
    public void makeSound() {
        System.out.println("Cat is meowing");
    }
}

class Dog extends Animal {
    public void makeSound() {
        System.out.println("Dog is barking");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal1 = new Cat();
        Animal animal2 = new Dog();

        animal1.makeSound(); // Output: Cat is meowing
        animal2.makeSound(); // Output: Dog is barking
    }
}
```

Uses:

Dynamic method dispatch is a powerful feature of Java that allows for greater flexibility and code reuse. It is one of the key features that makes Java an object-oriented programming language.

Here are some of the benefits of dynamic method dispatch:

- ➢ It allows for greater flexibility and code reuse.
- ➢ It can improve performance by allowing the JVM to optimize the code at runtime.
- ➢ It can make the code more readable and maintainable.

PACKAGE XX Constructor

Q-1: Why should you define your classes within your own package? How can you define a package?

here are some reasons why you should define your classes within your own package in Java:

- Package namespacing: Packages provide a way to namespace your classes, which helps to avoid name collisions. For example, if you have two different classes with the same name, but they are in different packages, then they will not conflict with each other.
- Package visibility control: Packages can be used to control the visibility of your classes. For example, you can mark a class as public, which means that it can be accessed from any other package. Or, you can mark a class as private, which means that it can only be accessed from within the same package.
- Package organization: Packages can be used to organize your classes into logical groups. This can make your code easier to understand and maintain.

To define a package in Java, you use the package keyword. For example, the following code defines a package named com.example:

package com.example;

Once you have defined a package, you can then use the import keyword to import classes from that package. For example, the following code imports the Person class from the com.example package:

import com.example.Person;

Once you have imported a class, you can then use it in your code. For example, the following code creates a new Person object:

Person person = new Person();

Q-2: Write the definition of package with example. What are the necessities of package? What is the procedure of creating a package? Illustrate with pseudo code.

Answer:

Definition of a package

A package is a collection of related files that are bundled together to be installed and used as a single unit. Packages are often used to distribute software, but they can also be used to distribute other types of files, such as data files, configuration files, and documentation.

Example of a package

A common example of a package is a Java JAR file. A JAR file is a type of archive file that can contain Java class files, resources, and other files. JAR files are often used to distribute Java libraries and applications.

Necessities of a package

A package typically contains the following elements:

- ➢ A manifest file: The manifest file is a text file that contains information about the package, such as the package name, version, and dependencies.
- ➢ Class files: Class files are the compiled Java source code for the classes in the package.
- ➢ Resources: Resources are files that are not Java class files, such as images, audio files, and text files.

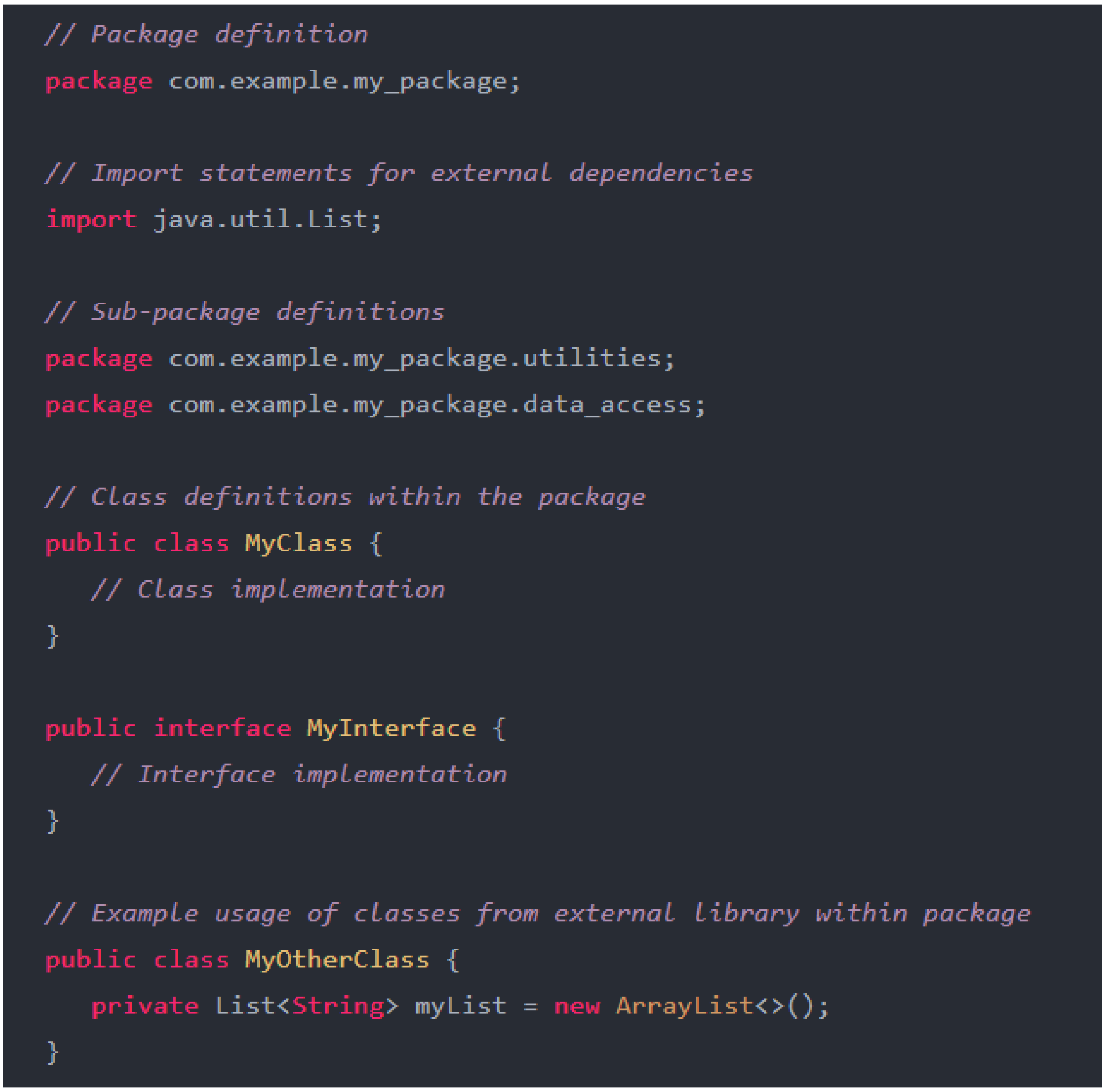
Procedure of creating a package

The procedure for creating a package varies depending on the type of package being created. However, the general steps involved in creating a package are as follows:

- ✓ Create the manifest file.
- ✓ Compile the Java source code.
- ✓ Package the files into a single archive file.
- ✓ Sign the package (optional).
- ✓ Deploy the package.

Pseudocode for creating a package

The following pseudocode shows the steps involved in creating a package:



```java
// Package definition
package com.example.my_package;

// Import statements for external dependencies
import java.util.List;

// Sub-package definitions
package com.example.my_package.utilities;
package com.example.my_package.data_access;

// Class definitions within the package
public class MyClass {
   // Class implementation
}

public interface MyInterface {
   // Interface implementation
}

// Example usage of classes from external library within package
public class MyOtherClass {
   private List<String> myList = new ArrayList<>();
}
```

Q-3: What is mean by data hiding? What is a member access modifier? Discuss different types of access modifiers used in java.

Or

What are the differences between public, protected & privet access specifier in a class.

Or

What are different access modifiers in Java?

Answer:

In object-oriented programming, a member access modifier is a keyword used to define the accessibility or visibility of class members such as variables, methods, and properties.

There are typically three levels of access modifiers:

Public: Members declared as public can be accessed from anywhere in the program.

Private: Members declared as private can only be accessed within the same class.

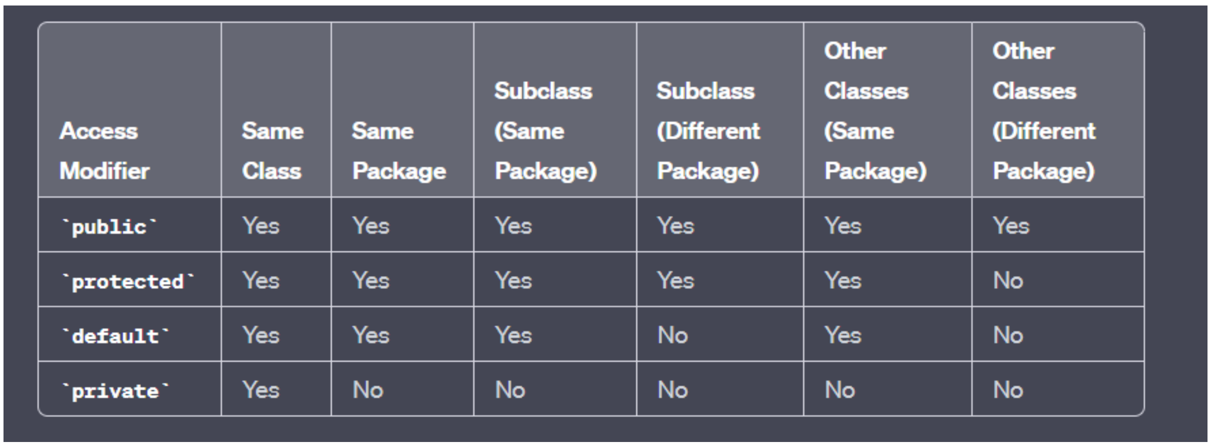
Protected: Members declared as protected can be accessed within the same class and its derived classes.

সর্বজনীন: সর্বজনীন হিসাবে ঘোষিত সদস্যদের প্রোগ্রামের যে কোনও জায়গা থেকে অ্যাক্সেস করা যেতে পারে।

ব্যক্তিগত: ব্যক্তিগত হিসাবে ঘোষিত সদস্যদের শুধুমাত্র একই শ্রেণীর মধ্যে অ্যাক্সেস করা যেতে পারে।

সুরক্ষিত: সুরক্ষিত হিসাবে ঘোষিত সদস্যদের একই শ্রেণী এবং এর থেকে প্রাপ্ত ক্লাসের মধ্যে অ্যাক্সেস করা যেতে পারে।

I. Public: A public member can be accessed from anywhere in the program. This means that any class, regardless of whether it's in the same package or a different package, can access the public member.
II. Private: A private member can only be accessed within the same class. This means that the member is not visible to even subclasses of the class or to other classes in the same package.
III. Protected: A protected member can be accessed within the same class, by subclasses in the same package or in a different package. This means that the member is not visible to classes that are not in the same package or that do not inherit from the class.
IV. Default (also known as package-private): A member with no access modifier specified (i.e., not public, private, or protected) can only be accessed within the same package. This means that classes outside of the package cannot access a default field or method.



Access Modifier	Same Class	Same Package	Subclass (Same Package)	Subclass (Different Package)	Other Classes (Same Package)	Other Classes (Different Package)
`public`	Yes	Yes	Yes	Yes	Yes	Yes
`protected`	Yes	Yes	Yes	Yes	Yes	No
`default`	Yes	Yes	Yes	No	Yes	No
`private`	Yes	No	No	No	No	No

Q-4: What is Garbage Collection in JAVA? How does Garbage Collection works in JAVA?

Answer:

Garbage collection (GC) is a automatic memory management feature in Java. The garbage collector automatically deallocates memory that is no longer being used by an application. This frees the application up to use that memory for other purposes.

In Java, objects are allocated on the heap. The heap is a region of memory that is dynamically allocated by the JVM. When an object is no longer being used, it is eligible for garbage collection. The garbage collector determines which objects are eligible for garbage collection by tracing the references to those objects. A reference is a way to access an object. There are two types of references: strong references and weak references.

A strong reference is a reference that prevents an object from being garbage collected. As long as there is at least one strong reference to an object, the object will not be garbage collected. A weak reference is a reference that does not prevent an object from being garbage collected. If an object has only weak references, it will be garbage collected as soon as there are no more strong references to it.

The garbage collector is a complex piece of software. There are many different algorithms that can be used to implement garbage collection. The most common algorithm is called the mark-and-sweep algorithm. The mark-and-sweep algorithm works by first marking all of the objects that are reachable from strong references. Once all of the reachable objects have been marked, the garbage collector sweeps through the heap and deallocates all of the unmarked objects.

Garbage collection is a powerful feature that can help to improve the performance and reliability of Java applications. However, it is important to understand how garbage collection works so that you can avoid common problems such as memory leaks and performance bottlenecks.

Here are some of the benefits of garbage collection:

- Automatic memory management: Garbage collection frees the programmer from having to manually manage memory. This can help to reduce errors and improve the performance of applications.
- Memory safety: Garbage collection helps to ensure that memory is used safely. This is because the garbage collector will automatically deallocate memory that is no longer being used. This can help to prevent memory leaks and other memory-related errors.
- Reliability: Garbage collection can help to improve the reliability of applications. This is because it can help to prevent memory leaks and other memory-related errors.

Here are some of the drawbacks of garbage collection:

- Performance: Garbage collection can have a negative impact on the performance of applications. This is because the garbage collector can cause pauses in the execution of the application.
- Non-deterministic behavior: Garbage collection can have non-deterministic behavior. This is because the garbage collector can run at any time during the execution of the application. This can make it difficult to debug applications and to predict the behavior of applications.
- Memory fragmentation: Garbage collection can cause memory fragmentation. This is because the garbage collector may deallocate memory that is no longer being used. This can lead to situations where there are small pockets of free memory that are not large enough to be used by objects.

However, it is important to understand the benefits and drawbacks of garbage collection so that you can use it effectively.

Inheritance:

Q-1: What is inheritance (উত্তরাধিকার)? What is the ambiguity that arises in multiple inheritance? How it can be overcome? explain with example.

Answer:

Inheritance is a concept in object-oriented programming where a class can inherit properties and behaviours from another class. The class that inherits is called the child or derived class, and the class it inherits from is called the parent or base class.

উত্তরাধিকার হল অবজেক্ট-ওরিয়েন্টেড প্রোগ্রামিংয়ের একটি ধারণা যেখানে একটি শ্রেণী অন্য শ্রেণীর থেকে বৈশিষ্ট্য এবং আচরণের উত্তরাধিকারী হতে পারে। যে শ্রেণীটি উত্তরাধিকারসূত্রে প্রাপ্ত হয় তাকে শিশু বা উদ্ভূত শ্রেণী বলা হয় এবং যে শ্রেণীটি উত্তরাধিকার সূত্রে প্রাপ্ত হয় তাকে অভিভাবক বা ভিত্তি শ্রেণী বলা হয়।

In multiple inheritance, a class can inherit from multiple parent classes at once. However, this can lead to ambiguity when two or more parent classes have methods or attributes with the same name. In such cases, it may be unclear which method or attribute the child class should inherit.

একাধিক উত্তরাধিকারে, একটি শ্রেণী একাধিক অভিভাবক শ্রেণীর থেকে একবারে উত্তরাধিকারী হতে পারে। যাইহোক, এটি অস্পষ্টতার দিকে নিয়ে যেতে পারে যখন দুই বা ততোধিক অভিভাবক শ্রেণীর একই নামের পদ্ধতি বা বৈশিষ্ট্য থাকে। এই ধরনের ক্ষেত্রে, এটি অস্পষ্ট হতে পারে কোন পদ্ধতি বা বৈশিষ্ট্য শিশু শ্রেণীর উত্তরাধিকারী হওয়া উচিত।

To overcome this ambiguity, some programming languages use method resolution order (MRO) algorithms to determine the order in which methods and attributes should be searched for.

এই অস্পষ্টতা কাটিয়ে ওঠার জন্য, কিছু প্রোগ্রামিং ল্যাঙ্গুয়েজ মেথড রেজোলিউশন অর্ডার (MRO) অ্যালগরিদম ব্যবহার করে কোন ক্রমে পদ্ধতি এবং গুণাবলী অনুসন্ধান করা উচিত তা নির্ধারণ করতে।

Here's an example to illustrate the ambiguity and how it can be resolved:

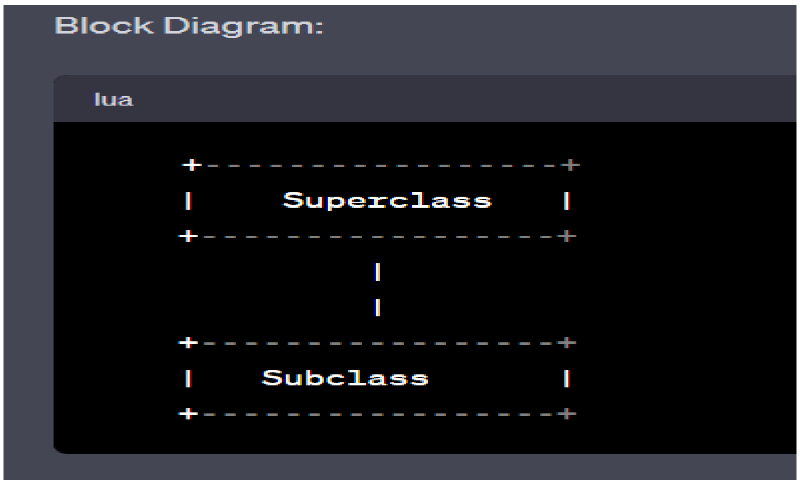
```
class Animal:
    def eat(self):
        print("Animal eats")

class Dog(Animal):
    def eat(self):
        print("Dog eats")

class Cat(Animal):
    def eat(self):
        print("Cat eats")

class DogCat(Dog, Cat):
    pass

d = DogCat()
d.eat()
```

In this example, we have a parent class `Animal` with a method `eat`, and two child classes `Dog` and `Cat`, each of which overrides the `eat` method. We then create a child class `DogCat` that inherits from both `Dog` and `Cat`.

Q-2: Discuss different type of inheritance with block diagram and example.

Answer;

<u>**Single Inheritance:**</u> Single inheritance is when a class inherits from only one superclass. The subclass acquires the properties and behavior of its superclass.

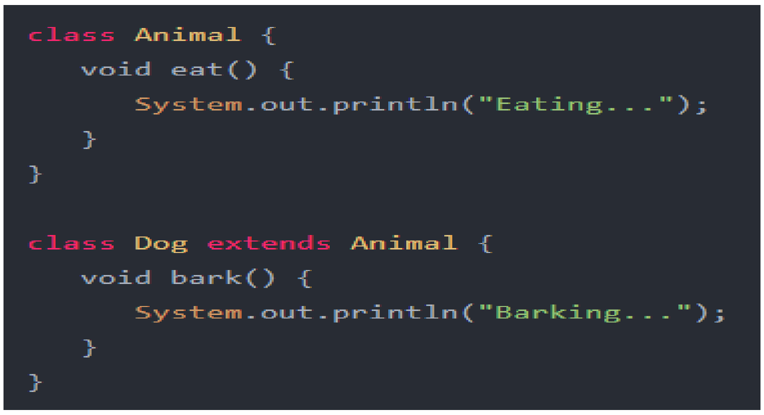


Block Diagram:

lua

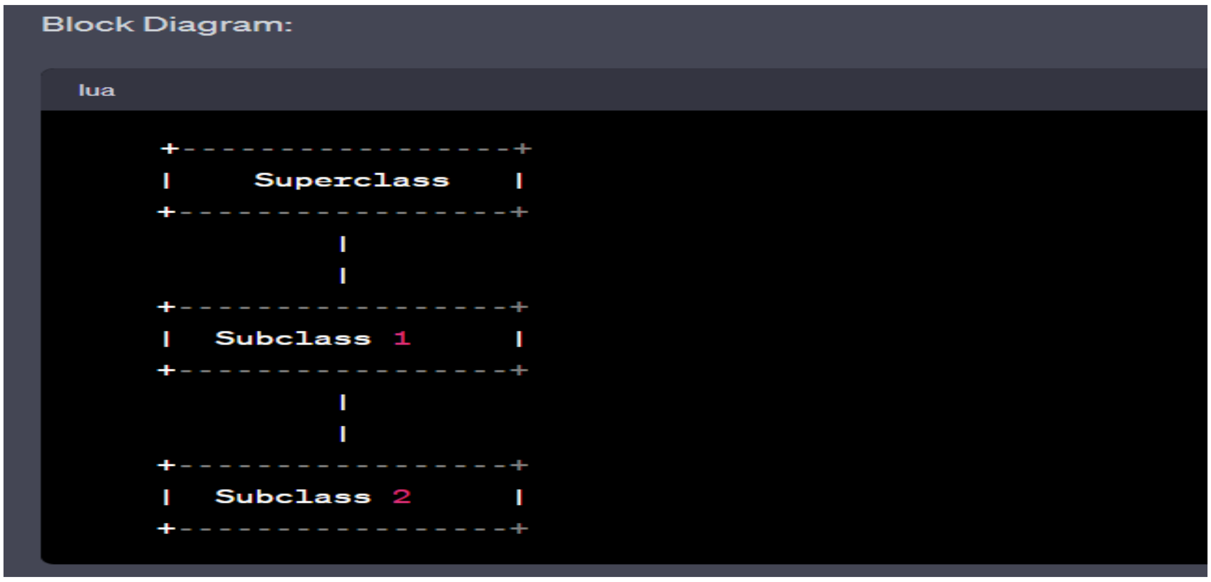
```
+------------------+
|    Superclass    |
+------------------+
         |
         |
+------------------+
|   Subclass       |
+------------------+
```

Example:



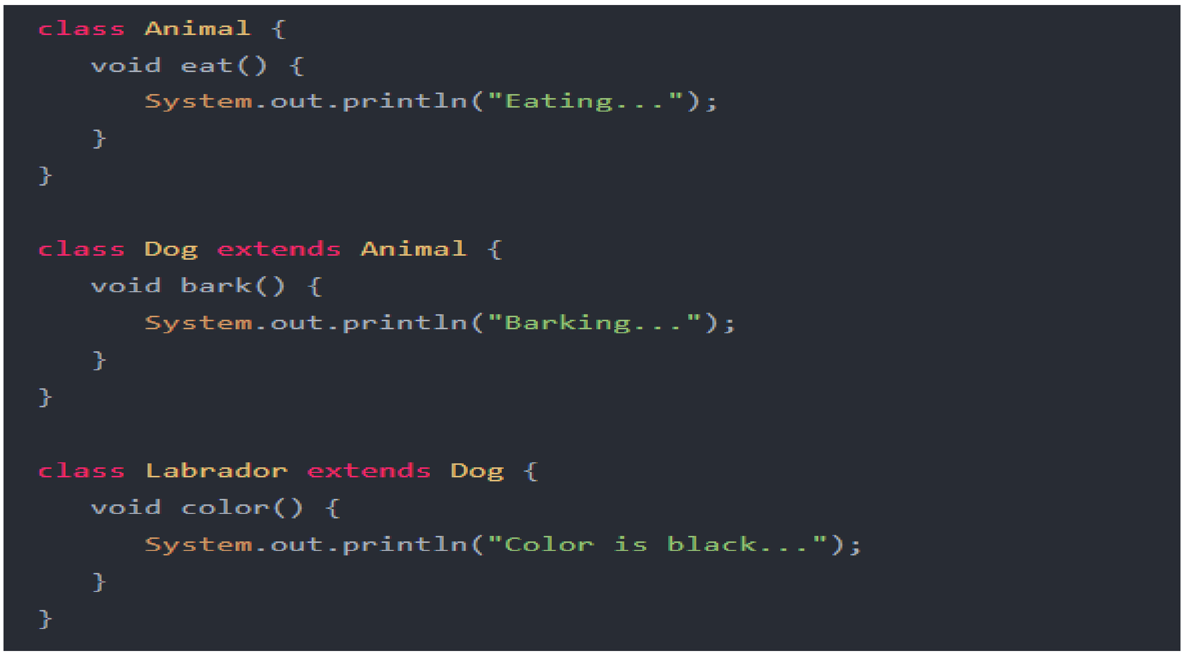
```
class Animal {
    void eat() {
        System.out.println("Eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking...");
    }
}
```

Multilevel Inheritance: Multilevel inheritance is when a class inherits from a superclass which in turn inherits from another superclass.

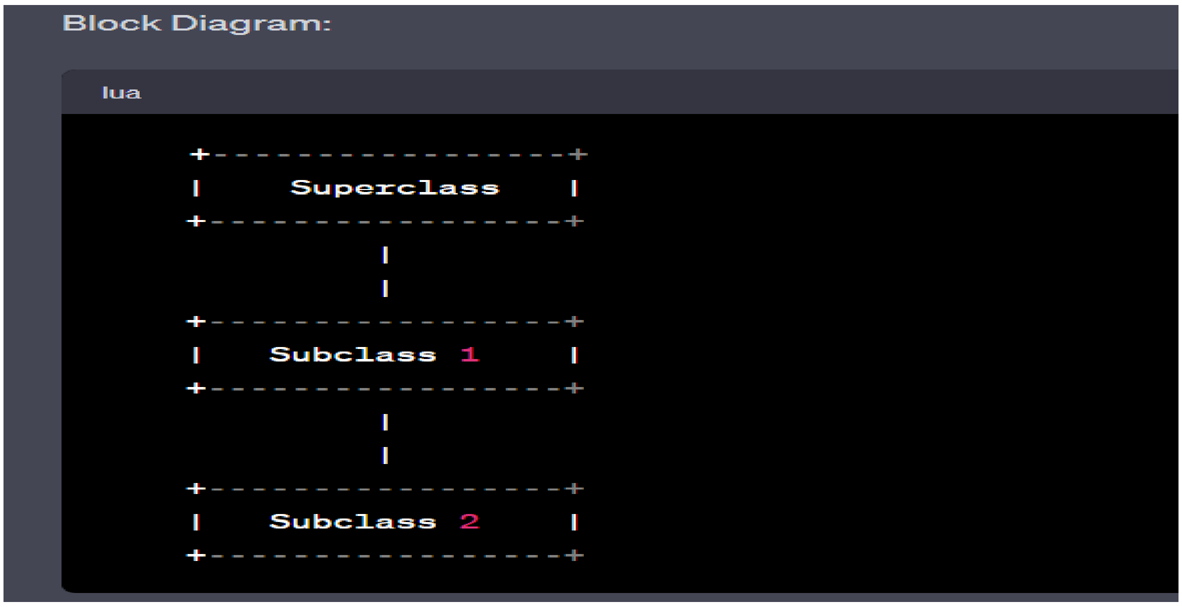

Block Diagram:

lua
```
+-----------------+
|   Superclass    |
+-----------------+
         |
         |
+-----------------+
| Subclass 1      |
+-----------------+
         |
         |
+-----------------+
| Subclass 2      |
+-----------------+
```

Here's an example of multilevel inheritance:


```
class Animal {
    void eat() {
        System.out.println("Eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking...");
    }
}

class Labrador extends Dog {
    void color() {
        System.out.println("Color is black...");
    }
}
```

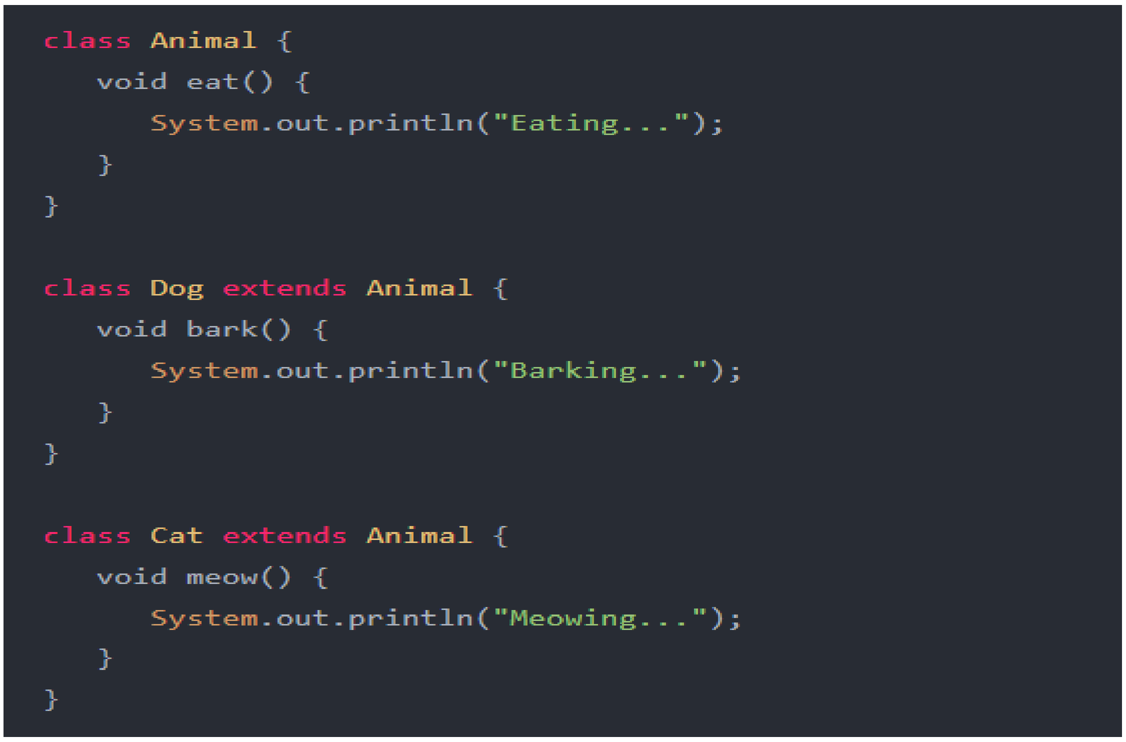
Hierarchical Inheritance: Hierarchical inheritance is when a single parent class is inherited by multiple child classes.


Block Diagram:

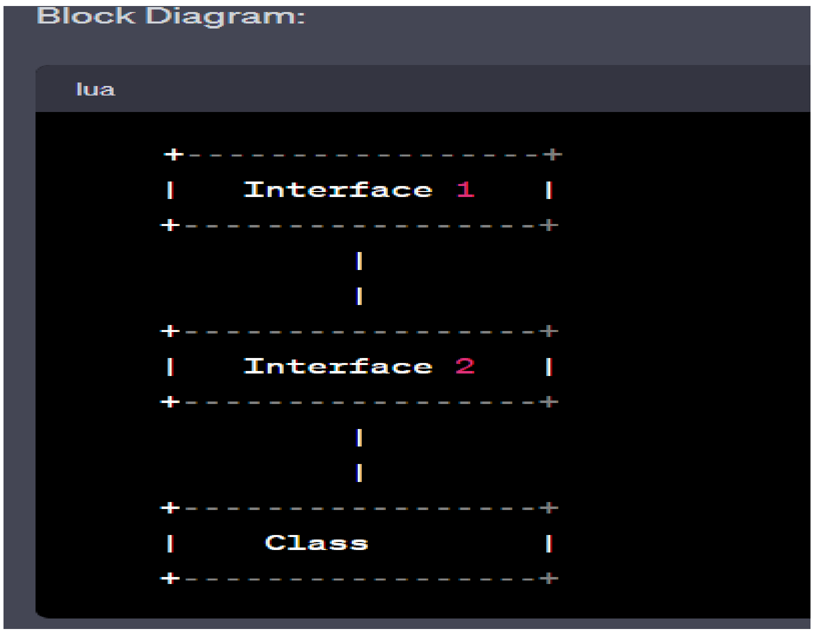
lua

```
+-----------------+
|   Superclass    |
+-----------------+
         |
         |
+-----------------+
|   Subclass 1    |
+-----------------+
         |
         |
+-----------------+
|   Subclass 2    |
+-----------------+
```

Here's an example of hierarchical inheritance:


```
class Animal {
    void eat() {
        System.out.println("Eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking...");
    }
}

class Cat extends Animal {
    void meow() {
        System.out.println("Meowing...");
    }
}
```

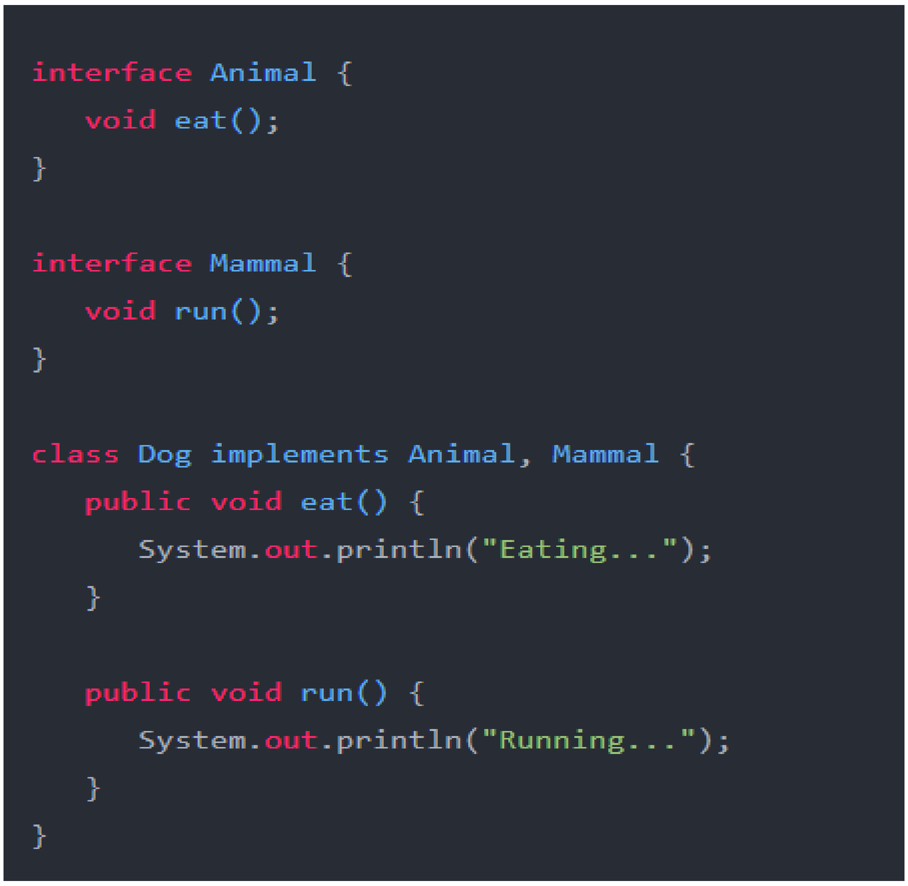
Multiple Inheritance (using Interfaces): Multiple inheritance is not directly supported in Java, but it can be achieved through the use of interfaces. An interface defines a set of methods that a class must implement. A class can implement multiple interfaces, thereby achieving multiple inheritance.


Block Diagram:

lua

```
+------------------+
|   Interface 1    |
+------------------+
         |
         |
+------------------+
|   Interface 2    |
+------------------+
         |
         |
+------------------+
|     Class        |
+------------------+
```

Here's an example of multiple inheritance using interfaces:



```
interface Animal {
    void eat();
}

interface Mammal {
    void run();
}

class Dog implements Animal, Mammal {
    public void eat() {
        System.out.println("Eating...");
    }

    public void run() {
        System.out.println("Running...");
    }
}
```

Q-3: Define abstract class? What are the condition for a class to become abstract? Explain with example.

Answer:

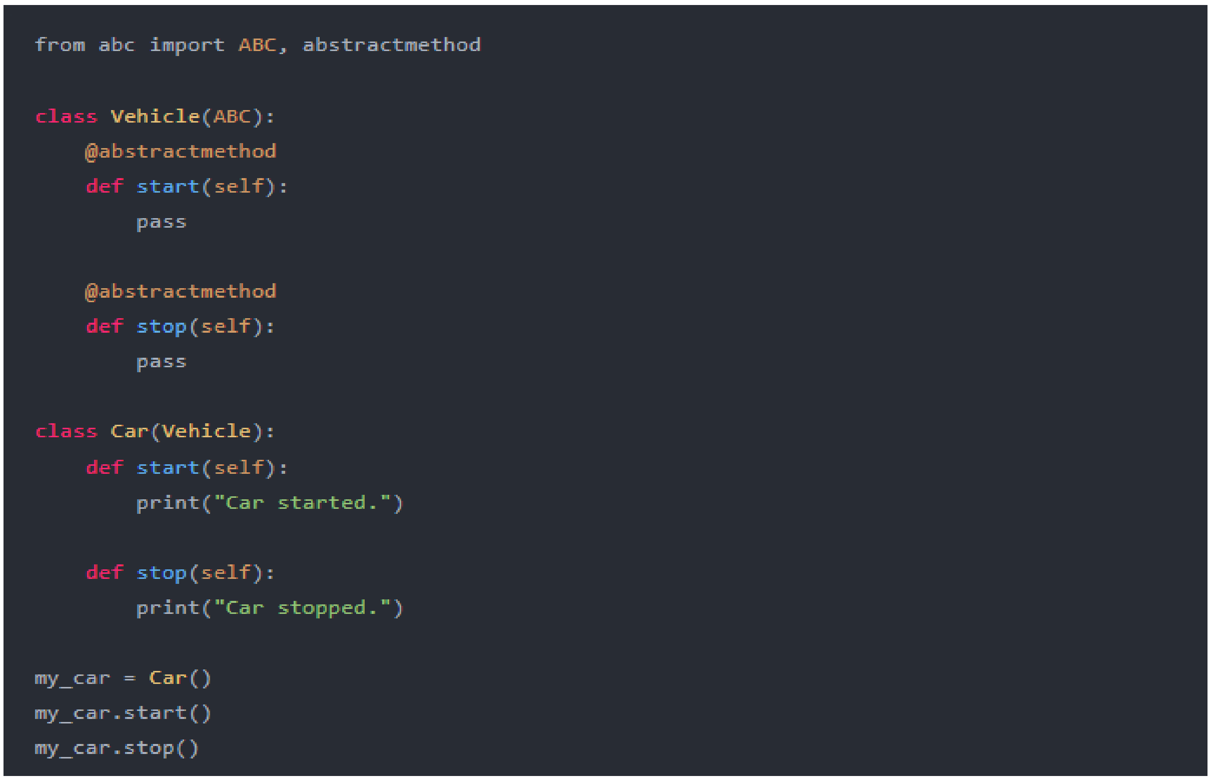
An abstract class is a class that cannot be instantiated and is meant to serve as a blueprint for other classes. It provides a common interface and defines certain methods or properties that derived classes must implement.

একটি বিমূর্ত শ্রেণী হল এমন একটি শ্রেণী যা তাত্‌ক্ষণিকভাবে করা যায় না এবং এটি অন্যান্য শ্রেণীর জন্য একটি নীলনকশা হিসাবে পরিবেশন করার জন্য বোঝানো হয়। এটি একটি সাধারণ ইন্টারফেস প্রদান করে এবং নির্দিষ্ট পদ্ধতি বা বৈশিষ্ট্যগুলিকে সংজ্ঞায়িত করে যা প্রাপ্ত ক্লাসগুলিকে অবশ্যই প্রয়োগ করতে হবে।

In order for a class to become abstract, it must meet the following conditions:

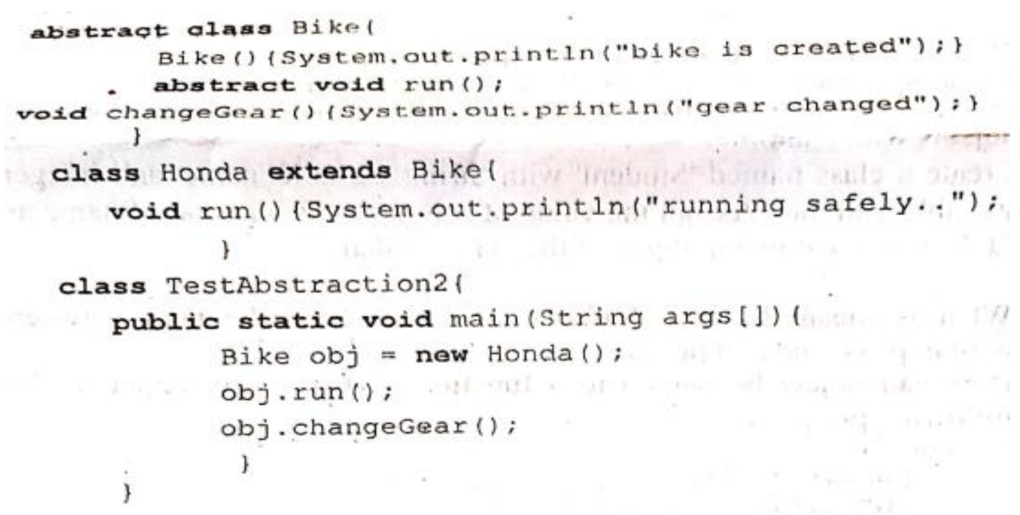
- The class must be declared with the abstract keyword.
- The class may or may not have abstract methods.
- If a class has at least one abstract method, the class itself must be declared as abstract.

An abstract method is a method that is declared in the abstract class but does not provide an implementation. It only contains the method signature, including the method name, parameters, and return type. The implementation of the abstract method is left to the derived classes.



```python
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass

    @abstractmethod
    def stop(self):
        pass

class Car(Vehicle):
    def start(self):
        print("Car started.")

    def stop(self):
        print("Car stopped.")

my_car = Car()
my_car.start()
my_car.stop()
```

Q-4: What will be the output of the following program?



```
abstract class Bike{
      Bike(){System.out.println("bike is created");}
    . abstract void run();
void changeGear(){System.out.println("gear changed");}
      }
  class Honda extends Bike{
     void run(){System.out.println("running safely..");
             }
  class TestAbstraction2{
     public static void main(String args[]){
           Bike obj = new Honda();
           obj.run();
           obj.changeGear();
            }
     }
```

Q-7: What is interface? Why use java interface? Why multiple is supported in interface while it is not supported in case of java?

Answer:

An interface in Java is a collection of abstract methods and constants, which can be implemented by a class. It defines a contract between the implementing class and the code that uses it.

জাভাতে একটি ইন্টারফেস হল বিমূর্ত পদ্ধতি এবং ধ্রুবকগুলির একটি সংগ্রহ, যা একটি শ্রেণী দ্বারা প্রয়োগ করা যেতে পারে। এটি বাস্তবায়নকারী শ্রেণী এবং এটি ব্যবহার করে এমন কোডের মধ্যে একটি চুক্তি সংজ্ঞায়িত করে।

One of the main reasons to use interfaces in Java is to allow for polymorphism. Polymorphism means that multiple objects can be treated as if they were of the same type, even if they are actually different classes. By defining a common interface for a group of classes, you can write code that works with any of them.

জাভাতে ইন্টারফেস ব্যবহার করার অন্যতম প্রধান কারণ হল পলিমরফিজমের অনুমতি দেওয়া। পলিমরফিজম মানে হল যে একাধিক বস্তুকে একই ধরণের হিসাবে বিবেচনা করা যেতে পারে, এমনকি যদি তারা আসলে বিভিন্ন শ্রেণীর হয়। ক্লাসের একটি গ্রুপের জন্য একটি সাধারণ ইন্টারফেস সংজ্ঞায়িত করে, আপনি কোড লিখতে পারেন যা তাদের যে কোনোটির সাথে কাজ করে।

Java supports multiple inheritance through interfaces because it avoids the "diamond problem" that arises when two superclasses have a method with the same name and signature. In Java, a class can only have one direct superclass, but it can implement multiple interfaces, each with its own set of methods. This allows for greater flexibility in designing complex systems, while still avoiding the conflicts that can arise with multiple inheritance.

জাভা ইন্টারফেসের মাধ্যমে একাধিক উত্তরাধিকার সমর্থন করে কারণ এটি "ডায়মন্ড সমস্যা" এড়িয়ে যায় যখন দুটি সুপারক্লাস একই নাম এবং স্বাক্ষর সহ একটি পদ্ধতি থাকে। জাভাতে, একটি ক্লাসে শুধুমাত্র একটি সরাসরি সুপারক্লাস থাকতে পারে, তবে এটি একাধিক ইন্টারফেস প্রয়োগ করতে পারে, প্রতিটি নিজস্ব পদ্ধতির সেট সহ। এটি জটিল সিস্টেম ডিজাইন করার ক্ষেত্রে বৃহত্তর নমনীয়তার অনুমতি দেয়, যদিও একাধিক উত্তরাধিকারের সাথে উদ্ভূত দ্বন্দ্বগুলি এড়িয়ে যায়।

Q-8: Define method overriding with suitable example? What are the rules for method overriding?

Answer:

Method overriding is the act of defining a method in a subclass that has the same name, return type, and parameters as a method in its superclass. When a method is called on an object of the subclass, the method defined in the subclass will be called instead of the method defined in the superclass.

Here's an example:

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

class Dog extends Animal {
  @Override
  public void makeSound() {
    System.out.println("The dog barks");
  }
}
```

In this example, we have a superclass Animal and a subclass Dog. The Dog class overrides the makeSound() method from the Animal class with its own implementation. When we call makeSound() on an instance of the Dog class, it will print out "The dog barks" instead of "The animal makes a sound".

The rules for method overriding are:

- ➢ The name of the method must be the same in both the superclass and the subclass.
- ➢ The parameter types and number of parameters must be the same in both the superclass and the subclass.
- ➢ The return type of the overriding method must be the same or a subtype of the return type of the overridden method.
- ➢ The access level of the overriding method cannot be more restrictive than the access level of the overridden method.

Here are some additional benefits of method overriding:

- ✓ It allows us to provide different implementations of a method for different types of objects.
- ✓ It allows us to add new features to existing classes without breaking compatibility with existing code.
- ✓ It allows us to implement the concept of polymorphism in Java.

Q-9: What is Byte code in java?

Answer:

Bytecode in Java is a binary format that is used to represent compiled Java code. When you write Java code, it is compiled into bytecode, which can then be executed on any platform that has a Java Virtual Machine (JVM) installed.

The advantage of using bytecode is that it provides a platform-independent way of representing code. Since the same bytecode can be executed on any platform with a JVM implementation, you don't need to recompile your code for each platform.

Bytecode is also more compact than machine code, making it easier to distribute over the network. Java applets and applications, for example, are distributed as bytecode over the internet, which is then executed by a JVM on the end user's system.

In summary, Bytecode is an intermediate representation of code that is generated by the Java compiler and can be executed on any platform that has a JVM installed.

Q-10: What do you mean by abstract class? Why it is needed? Explain with example.

Answer:

In Java, an abstract class is a class that cannot be instantiated and is designed to be subclassed by other classes. An abstract class can contain both abstract and non-abstract (concrete) methods, as well as instance variables, constructors, and other features of a standard Java class.

An abstract class is needed because it allows you to define a common interface or behavior for a group of related classes without having to repeat code in each individual class. This makes your code more modular and easier to maintain since changes made to the abstract class will automatically propagate to all of its subclasses.

Here's an example of an abstract class in Java:

```
public abstract class Shape {
   protected double area;
   protected double perimeter;

   public abstract void calculateArea();

   public abstract void calculatePerimeter();

   public void display() {
      System.out.println("Area: " + area);
      System.out.println("Perimeter: " + perimeter);
   }
}
```

In this example, we define an abstract class called Shape which has two abstract methods: calculateArea and calculatePerimeter. Any subclass of Shape must implement these two methods, which define the basic properties of any geometric shape. For example, we could create a Rectangle subclass like this:

```
public class Rectangle extends Shape {
   private double length;
   private double width;

   public Rectangle(double length, double width) {
       this.length = length;
       this.width = width;
   }

   @Override
   public void calculateArea() {
       area = length * width;
   }

   @Override
   public void calculatePerimeter() {
       perimeter = 2 * (length + width);
   }
}
```

Polymorphism & Keyword:

Q-1: What is polymorphism? How polymorphism is achieved at compile time and run time?

Answer:

Polymorphism is the ability of an object to take on multiple forms. In object-oriented programming, polymorphism allows objects of different classes to be treated as if they are objects of the same class.

পলিমরফিজম হল একটি বস্তুর একাধিক রূপ নেওয়ার ক্ষমতা। অবজেক্ট-ওরিয়েন্টেড প্রোগ্রামিং-এ, পলিমরফিজম বিভিন্ন শ্রেণীর বস্তুকে একই শ্রেণীর বস্তু হিসাবে বিবেচনা করার অনুমতি দেয়।

Polymorphism can be achieved in two ways: at compile time and at runtime.

Compile-Time Polymorphism (Static Polymorphism):

Compile-time polymorphism is achieved through method overloading. Method overloading means that you define several methods with the same name but different parameters in a single class. The compiler determines which method to call based on the number and type of arguments passed to it. For example:

কম্পাইল-টাইম পলিমরফিজম পদ্ধতি ওভারলোডিংয়ের মাধ্যমে অর্জন করা হয়। মেথড ওভারলোডিং এর অর্থ হল আপনি একই নামের সাথে একাধিক পদ্ধতি সংজ্ঞায়িত করেন কিন্তু একটি একক ক্লাসে বিভিন্ন পরামিতি। কম্পাইলার এটিতে পাস করা আর্গুমেন্টের সংখ্যা এবং প্রকারের উপর ভিত্তি করে কোন পদ্ধতিতে কল করতে হবে তা নির্ধারণ করে। উদাহরণ স্বরূপ:

```python
class Math:
    def add(self, x, y):
        return x + y

    def add(self, x, y, z):
        return x + y + z
```

Runtime Polymorphism (Dynamic Polymorphism):

Runtime polymorphism, also known as dynamic polymorphism, is achieved through method overriding. Method overriding means that a subclass provides its own implementation of a method that is already provided by its parent class. For example:

রানটাইম পলিমরফিজম, যা ডাইনামিক পলিমরফিজম নামেও পরিচিত, মেথড ওভাররাইডিংয়ের মাধ্যমে অর্জন করা হয়। মেথড ওভাররাইডিং এর অর্থ হল একটি সাবক্লাস একটি পদ্ধতির নিজস্ব বাস্তবায়ন প্রদান করে যা ইতিমধ্যেই এর প্যারেন্ট ক্লাস দ্বারা সরবরাহ করা হয়েছে। উদাহরণ স্বরূপ:

```python
class Animal:
    def make_sound(self):
        print("Animal sound")

class Dog(Animal):
    def make_sound(self):
        print("Bark bark!")

class Cat(Animal):
    def make_sound(self):
        print("Meow meow!")
```

Q-2: Define Method overloading with suitable example? What are rules for method overriding?

Answer:

Method overloading is a feature in Java that allows you to define multiple methods with the same name but different parameters within the same class. The main advantage of method overloading is that it helps to improve the readability and reusability of your code.

Here's an example:

```
public class Calculator {

   public int add(int x, int y) {
      return x + y;
   }

   public double add(double x, double y) {
      return x + y;
   }

   public int add(int x, int y, int z) {
      return x + y + z;
   }
}
```

In the above example, we have three methods named add which are overloaded. The first method takes two integers as arguments, the second method takes two doubles as arguments, and the third method takes three integers as arguments.

Method Overloading Rules:

- Method overloading can only occur within the same class.
- The method signature must be different for each overloaded method.
- The method name must be the same for all overloaded methods.
- The return type of the method does not matter for method overloading.
- The access modifier of the method does not matter for method overloading.

Method overriding is a feature in Java that allows a subclass to provide its own implementation of a method that is already defined in its superclass.

The rules for method overriding are as follows:

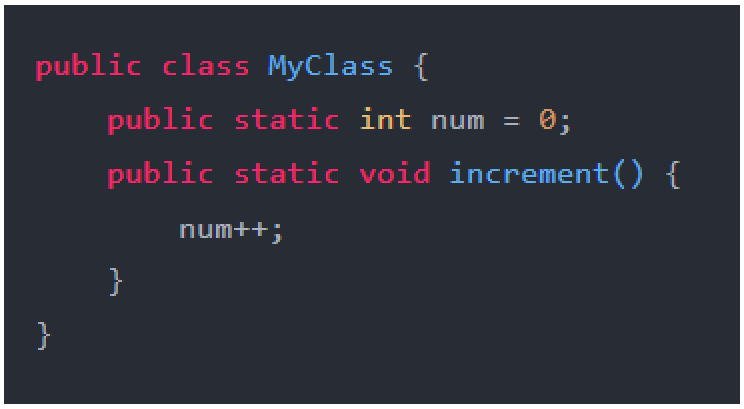
- ➢ The method in the subclass must have the same method signature (name and parameter types) as the method in the superclass.
- ➢ The method in the subclass must have the same or narrower return type than the method in the superclass.
- ➢ The access level of the method in the subclass cannot be more restrictive than the access level in the superclass.
- ➢ The method in the subclass cannot throw checked exceptions that are not specified in the throws clause of the method in the superclass.

Q-3: What is static keyword? what is final keyword? Why "this" keyword is used in Java language? explain with example

Answer:

Static keyword: In Java, the "static" keyword is used to define a class-level variable or method that can be accessed without creating an object of the class. When a variable or method is declared as static, it belongs to the class rather than to any instance of the class.

Example:



```java
public class MyClass {
    public static int num = 0;
    public static void increment() {
        num++;
    }
}
```

Final keyword: In Java, the "final" keyword is used to declare a constant that cannot be changed after it has been assigned a value. It is also used to make a method or class immutable.

Example:

```java
public class Circle {
    public final double PI = 3.14159;
    public final double radius;
    public Circle(double r) {
        radius = r;
    }
    public final double getArea() {
        return PI * radius * radius;
    }
}
```

"this" keyword 😟 (uses):

The "this" keyword in Java is a reference to the current object within an instance method or constructor. It is used to differentiate between instance variables and method parameters or local variables that have the same name. Here's an example to illustrate the usage of the "this" keyword:

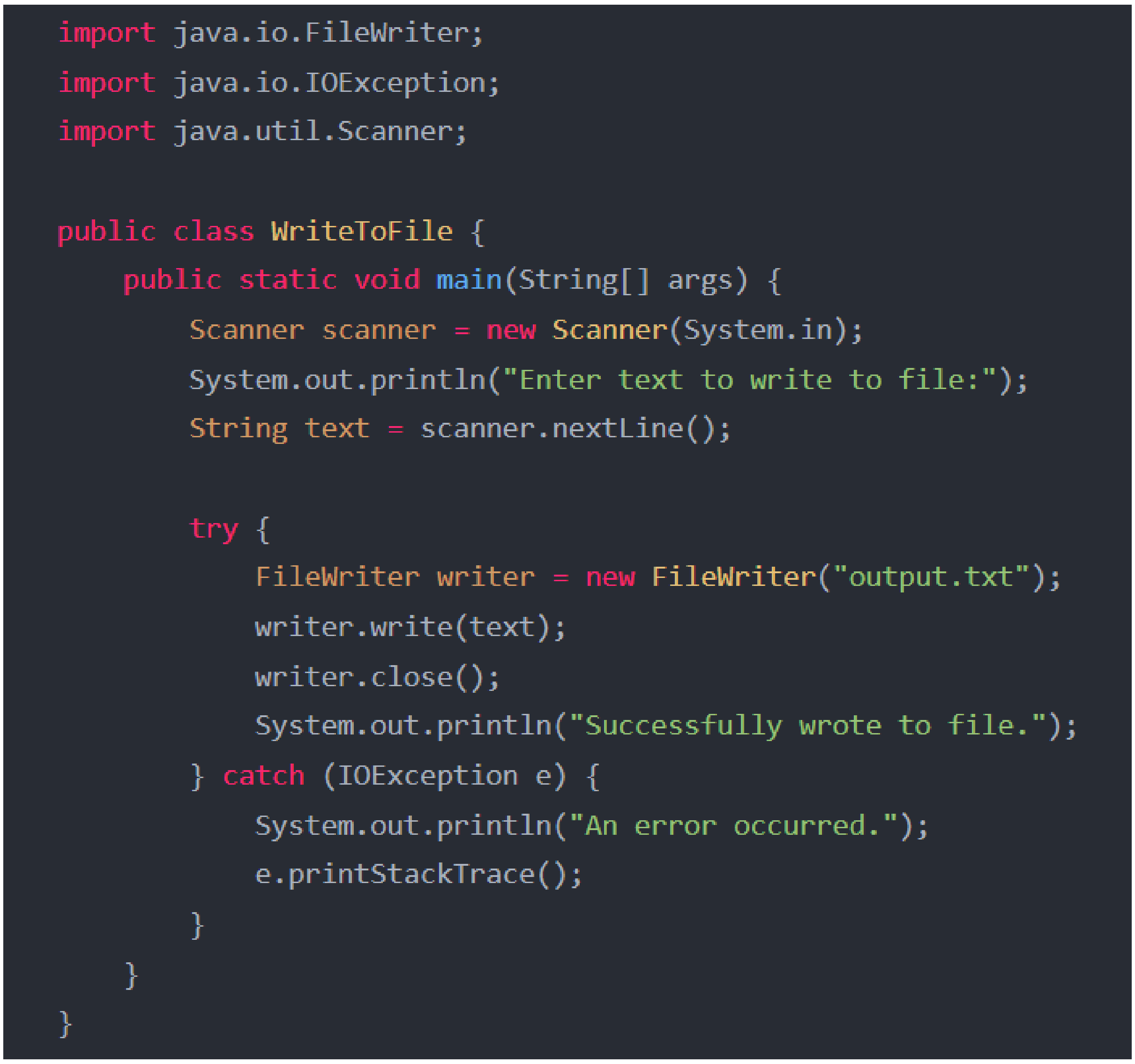
```java
public class Person {
    private String name;

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

    public void sayHello() {
        System.out.println("Hello, my name is " + this.name);
    }
}
```

In this code, the "this" keyword is used to refer to the "name" instance variable of the current object. In the constructor, "this.name = name;" sets the value of the "name" instance variable to the value passed in as a parameter. In the "sayHello" method, "this.name" is used to get the value of the "name" instance variable of the current object and print it out.

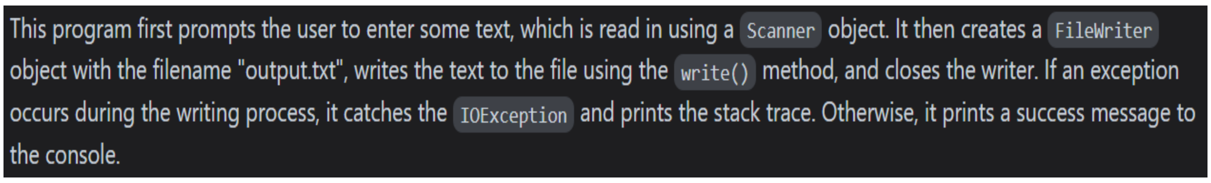
It's important to note that using "this" is optional in many cases, but can improve code readability and reduce ambiguity, especially in cases where local variables have the same name as instance varia

Q-4: Write a simple java program to read data from keyboard and write into file.

Answer:



```java
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class WriteToFile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter text to write to file:");
        String text = scanner.nextLine();

        try {
            FileWriter writer = new FileWriter("output.txt");
            writer.write(text);
            writer.close();
            System.out.println("Successfully wrote to file.");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```



This program first prompts the user to enter some text, which is read in using a `Scanner` object. It then creates a `FileWriter` object with the filename "output.txt", writes the text to the file using the `write()` method, and closes the writer. If an exception occurs during the writing process, it catches the `IOException` and prints the stack trace. Otherwise, it prints a success message to the console.

Exception handling

Q-1: What is an exception? How exception are handled in Java? Explain the mechanism with proper example.

Answer:

In Java, an exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. When an exception occurs, it may be handled by code that is written specifically to handle that type of exception.

জাভাতে, একটি ব্যতিক্রম হল এমন একটি ঘটনা যা একটি প্রোগ্রাম কার্যকর করার সময় ঘটে যা নির্দেশাবলীর স্বাভাবিক প্রবাহকে ব্যাহত করে। যখন একটি ব্যতিক্রম ঘটে, তখন এটি কোড দ্বারা পরিচালিত হতে পারে যা বিশেষভাবে সেই ধরনের ব্যতিক্রম পরিচালনা করার জন্য লেখা হয়।

Exception handling in Java is done using try-catch-finally blocks. The try block contains the code that may throw an exception. The catch block catches the exception and contains the code to handle it. The finally block is used to execute any code that needs to be executed regardless of whether or not an exception occurred.

জাভাতে ব্যতিক্রম হ্যান্ডলিং করা হয় ট্রাই-ক্যাচ-ফাইনালি ব্লক ব্যবহার করে। ট্রাই ব্লকে এমন কোড রয়েছে যা ব্যতিক্রম হতে পারে। ক্যাচ ব্লক ব্যতিক্রম ক্যাচ করে এবং এটি পরিচালনা করার জন্য কোড ধারণ করে। পরিশেষে ব্লক ব্যবহার করা হয় যেকোন কোড এক্সিকিউট করার জন্য যা এক্সিকিউট করা দরকার তা নির্বিশেষে ব্যতিক্রম ঘটেছে কিনা।

In computer programming, an exception is an event that occurs during program execution that interrupts the normal flow of the program's instructions. When an exception occurs, the program stops executing its regular instructions and jumps to a specific block of code called an exception handler. The exception handler may then perform some special processing to handle the exception and allow the program to continue executing.

কম্পিউটার প্রোগ্রামিংয়ে, ব্যতিক্রম হল এমন একটি ঘটনা যা প্রোগ্রাম এক্সিকিউশনের সময় ঘটে যা প্রোগ্রামের নির্দেশাবলীর স্বাভাবিক প্রবাহকে বাধাগ্রস্ত করে। যখন একটি ব্যতিক্রম ঘটে, প্রোগ্রামটি তার নিয়মিত নির্দেশাবলী কার্যকর করা বন্ধ করে দেয় এবং একটি ব্যতিক্রম হ্যান্ডলার নামক কোডের একটি নির্দিষ্ট ব্লকে চলে যায়। ব্যতিক্রম হ্যান্ডলার তারপর ব্যতিক্রম পরিচালনা করার জন্য কিছু বিশেষ প্রক্রিয়াকরণ করতে পারে এবং প্রোগ্রামটিকে কার্যকর করা চালিয়ে যাওয়ার অনুমতি দেয়।

Here's an example to illustrate the mechanism of exception handling in Java:

```
public class Example {
  public static void main(String[] args) {
    try {
      int a = 10 / 0; // This will throw an ArithmeticException
    } catch (ArithmeticException e) {
      System.out.println("An arithmetic exception occurred: " + e.getMessage());
    } finally {
      System.out.println("This code will always be executed.");
    }
  }
}
```

In this example, we're trying to divide the integer value 10 by zero, which is not allowed and will throw an Arithmetic Exception. We've put this code inside a try block.

When we run this code, the output will be:

```
An arithmetic exception occurred: / by zero
This code will always be executed.
```

Q-2: What happens if an exception is not caught? Write down the difference between exception and errors.

Answer:

If an exception is not caught, it will propagate up the call stack until it reaches the top-level of the program, at which point it will cause the program to terminate and display an error message that includes a traceback of the call stack.

যদি একটি ব্যতিক্রম ধরা না হয়, এটি কল স্ট্যাকটি প্রচার করবে যতক্ষণ না এটি প্রোগ্রামের শীর্ষ স্তরে পৌঁছায়, এই সময়ে এটি প্রোগ্রামটিকে বন্ধ করে দেবে এবং একটি ত্রুটি বার্তা প্রদর্শন করবে যাতে কল স্ট্যাকের একটি ট্রেসব্যাক অন্তর্ভুক্ত থাকে।

Errors	Exception
A lack of system resources typically causes errors in a program, and the program that a programmer writes is not designed to detect such issues.	An exception occurs mainly due to issues in programming such as bugs, typos, syntax errors etc.
Errors usually happen at runtime. Therefore they're of the unchecked type.	Exceptions can arise at both runtime and compile time.
System crashes and out of memory errors are two instances of errors.	SQLException is an example of exceptions in Java
Errors belong to java.lang.error.	Errors belong to java.lang.Exception.
Errors are irrecoverable and lead to abnormal termination of the program.	Exceptions are recoverable and can be handled by exception handling techniques.

Q-3: Explain how you can create your own java exception with a proper example.

Answer:
<u>Steps to create a Custom Exception with an Example</u>

CustomException class is the custom exception class this class is extending Exception class.

Create one local variable message to store the exception message locally in the class object.

We are passing a string argument to the constructor of the custom exception object. The constructor set the argument string to the private string message.

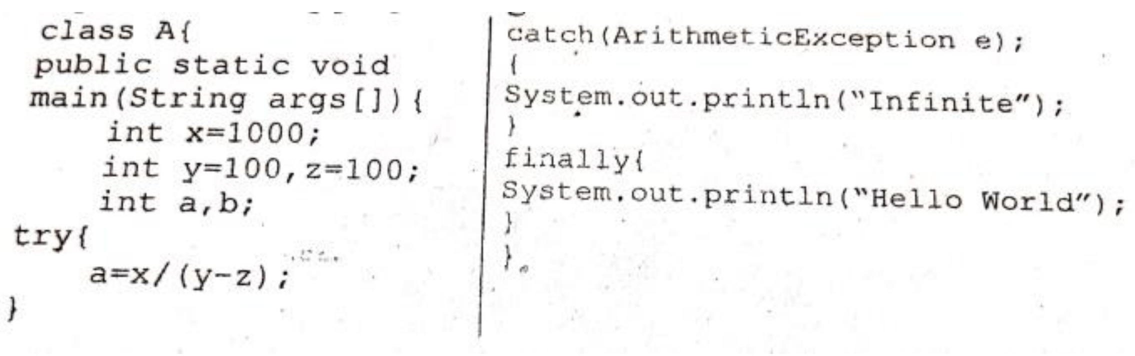
toString () method is used to print out the exception message.

We are simply throwing a CustomException using one try-catch block in the main method and observe how the string is passed while creating a custom exception. Inside the catch block, we are printing out the message.

To create your own Java exception, you need to create a class that extends the Exception class or one of its subclasses. Here's an example:

```
public class MyException extends Exception {
  public MyException(String message) {
    super(message);
  }
}
```
In this example, we've created a custom exception called MyException that extends the base Exception class. We've also defined a constructor that takes a message as a parameter and calls the superclass constructor with that message.

Q-4: What are exception handling in java? What will be the output of the following program?



```
 class A{
 public static void
 main(String args[]){
     int x=1000;
     int y=100,z=100;
     int a,b;
try{
    a=x/(y-z);
}
catch(ArithmeticException e);
{
System.out.println("Infinite");
}
finally{
System.out.println("Hello World");
}
}.
```

Answer:

Exception handling in Java is a mechanism that allows us to handle runtime errors gracefully. The try-catch block is used to catch and handle exceptions in Java. Here's an example program that demonstrates exception handling:

```
public class ExceptionHandlingExample {
  public static void main(String[] args) {
    int num1, num2;
    try {
      num1 = 0;
      num2 = 10 / num1;
      System.out.println(num2);
      System.out.println("End of try block");
    } catch (ArithmeticException e) {
      System.out.println("You can't divide by zero");
    }

    System.out.println("After try-catch block");
  }
}
```

Q-5:How to perform two tasks by two treads? What is the purpose of join method?

Answer:

To perform two tasks by two threads, you can create two threads and start them simultaneously using the start() method. Here's an example:

```
Thread t1 = new Thread(new Task1());

Thread t2 = new Thread(new Task2());

t1.start();

t2.start();
```

In this example, we've created two threads, t1 and t2, and started them using the start() method. Each thread performs a different task, as defined by the Task1 and Task2 classes.

The join() method is used to wait for a thread to complete before continuing with the execution of the program. When a thread calls the join() method on another thread, it blocks until that thread completes its execution. The purpose of the join() method is to ensure that the threads complete their tasks in the correct order.

Q-6: What is multithreading? Explain the procedure to create multiple threads in java language with examples.

Answer:

Multithreading in Java refers to the concurrent execution of multiple threads within a single program. It allows different parts of a program to execute simultaneously, improving performance and responsiveness. Each thread represents an independent flow of execution, allowing multiple tasks to be performed concurrently.

To create multiple threads in Java, you have several options. Here's an example using the Thread class:

```
public class MultithreadingExample {
    public static void main(String[] args) {
        Thread thread1 = new Thread(new Task1());
        Thread thread2 = new Thread(new Task2());
        thread1.start();
        thread2.start();
    }
}
class Task1 implements Runnable {
    @Override
    public void run() {
        // Task 1 logic here
        System.out.println("Task 1 is executing.");
    }
}

class Task2 implements Runnable {
    @Override
    public void run() {
        // Task 2 logic here
        System.out.println("Task 2 is executing.");
    }
}
```

In this example, we create two threads (thread1 and thread2) by instantiating the Thread class and passing the corresponding task objects (Task1 and Task2) to their constructors. Each task is defined as a separate class implementing the Runnable interface.

After creating the threads, we start their execution by calling the start() method. This will initiate the concurrent execution of both threads, allowing their respective tasks to execute simultaneously.

Q-7:Using exception handling mechanism write a program to manage arrayIndexOutOfBoundsException in java.

Answer:

```
public class ArrayOutOfBoundsExceptionExample {
   public static void main(String[] args) {
      int[] array = {1, 2, 3};
      try {
         System.out.println(array[3]);
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Error: " + e.getMessage());
      }
   }
}
```

In this example, we've created an array of size 3 and tried to access the element at index 3, which doesn't exist. This will result in an ArrayIndexOutOfBoundsException being thrown. We catch the exception using the try-catch block and print out an error message.

Q-8: Distinguish between process based and thread based multitasking. Write the states of life cycle of a thread and draw state transition diagram.

Answer:

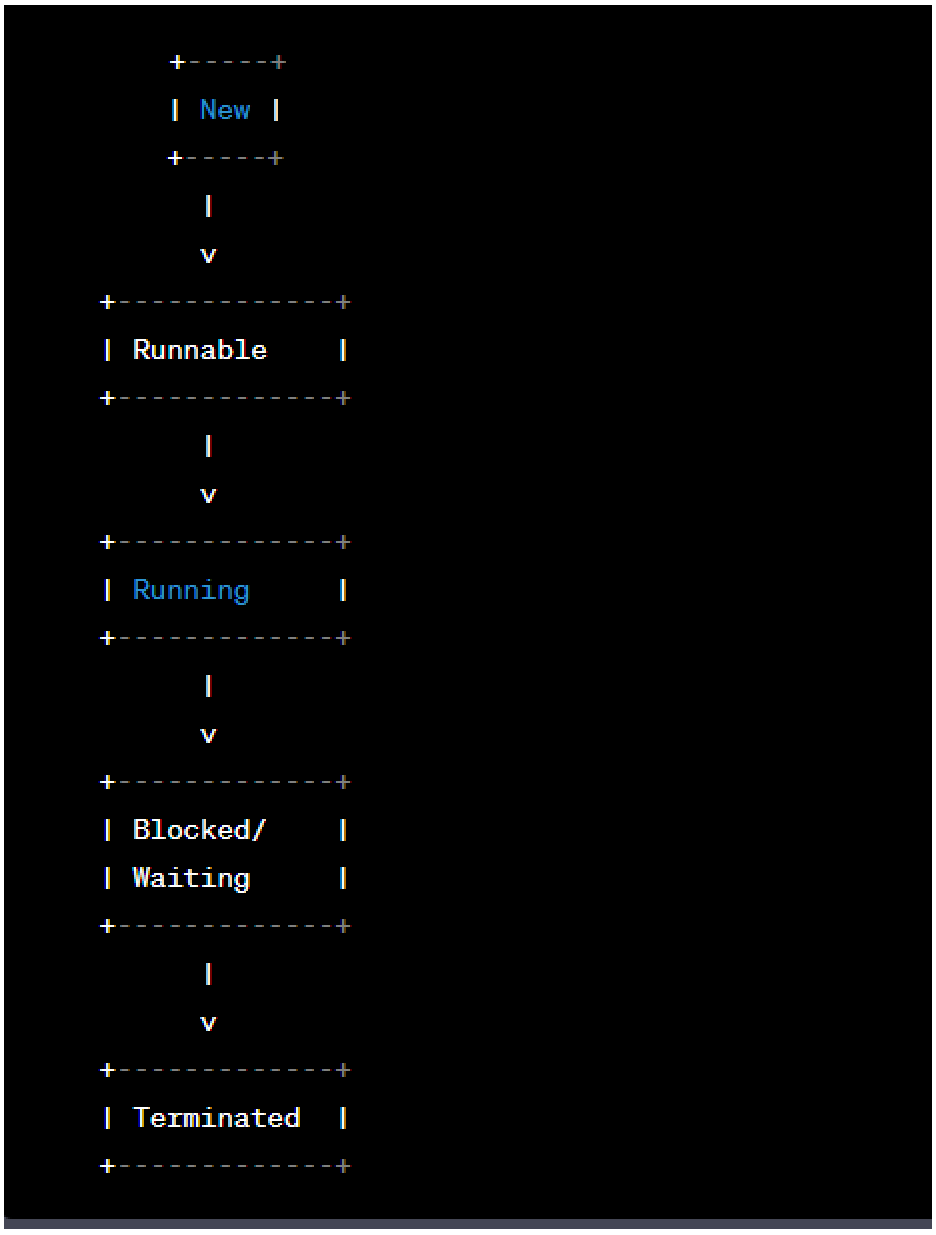
Process-based multitasking involves executing multiple processes simultaneously, where each process has its own memory space and resources. Each process runs independently and can perform different tasks.

Thread-based multitasking, on the other hand, involves executing multiple threads within a single process. Threads share the same memory space and resources of the process. Multiple threads within a process can execute concurrently, allowing for concurrent execution of tasks.

States of the life cycle of a thread in Java:

- New: The thread is in the new state before it is started.
- Runnable: The thread is ready to run and waiting for its turn to be allocated processor time.
- Running: The thread is currently being executed.
- Blocked/Waiting: The thread is blocked or waiting for a specific condition to be met.
- Terminated: The thread has finished its execution and reached the end of its life cycle.

Here's a state transition diagram for the life cycle of a thread:



```
          +-----+
          | New |
          +-----+
             |
             v
      +-------------+
      | Runnable    |
      +-------------+
             |
             v
      +-------------+
      | Running     |
      +-------------+
             |
             v
      +-------------+
      | Blocked/    |
      | Waiting     |
      +-------------+
             |
             v
      +-------------+
      | Terminated  |
      +-------------+
```

Q-9:What is an exception? Discuss how try-catch-finally blocks works in java.

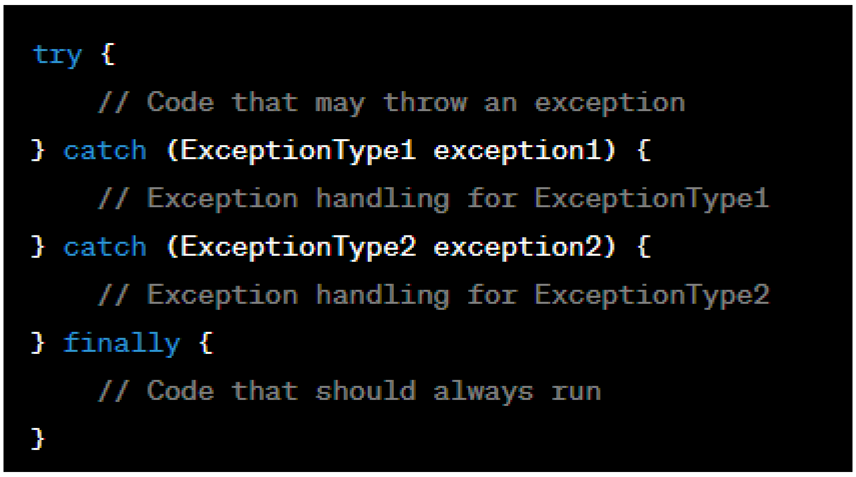
Answer:

An exception in Java is an event that occurs during the execution of a program that disrupts the normal flow of instructions. It represents an error or an exceptional condition that needs to be handled.

The try-catch-finally blocks in Java are used for exception handling. Here's how they work:

- ➢ The try block is used to enclose the code that may throw an exception. If an exception occurs within the try block, the normal flow of the program is interrupted, and the execution jumps to the corresponding catch block.
- ➢ The catch block is used to catch and handle the thrown exception. It specifies the type of exception it can handle, and if a matching exception occurs, the code within the catch block is executed. Multiple catch blocks can be used to handle different types of exceptions.
- ➢ The finally block is optional and is used to specify code that should always run, regardless of whether an exception occurred or not. It is typically used to release resources or perform cleanup operations.

The general syntax of try-catch-finally blocks in Java is as follows:

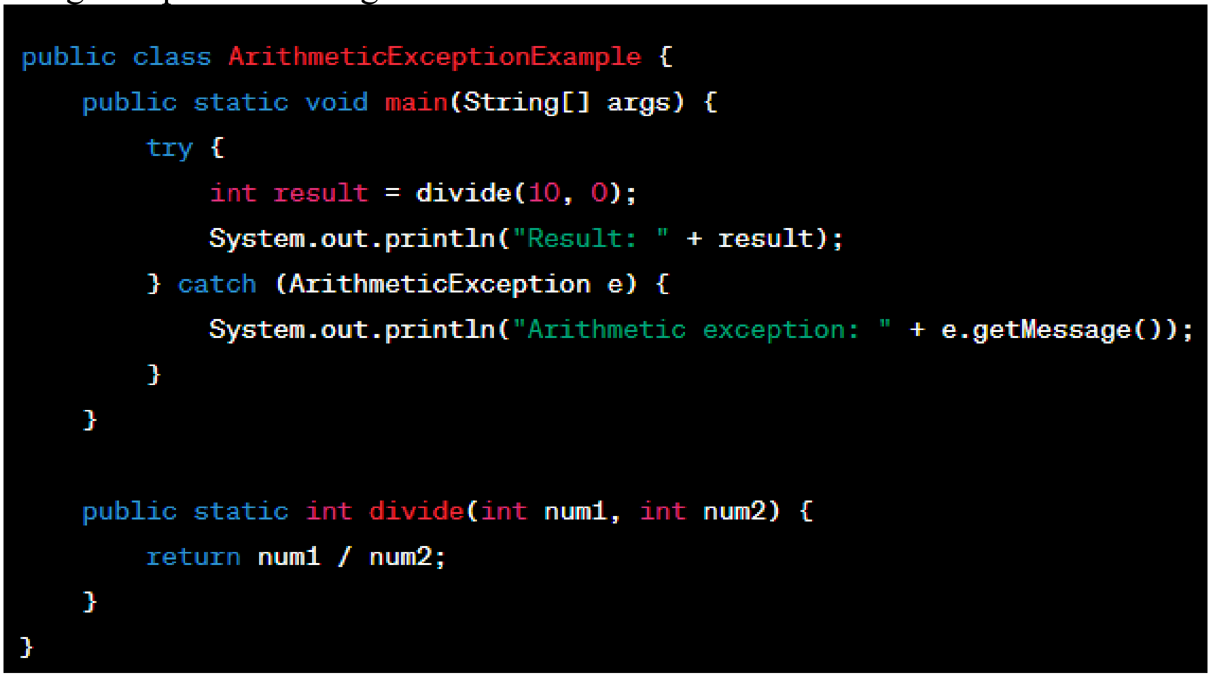


```
try {
    // Code that may throw an exception
} catch (ExceptionType1 exception1) {
    // Exception handling for ExceptionType1
} catch (ExceptionType2 exception2) {
    // Exception handling for ExceptionType2
} finally {
    // Code that should always run
}
```

Q-10: What is exception handling? Using exception handling mechanism write a program to manage Arithmetic exception in java.

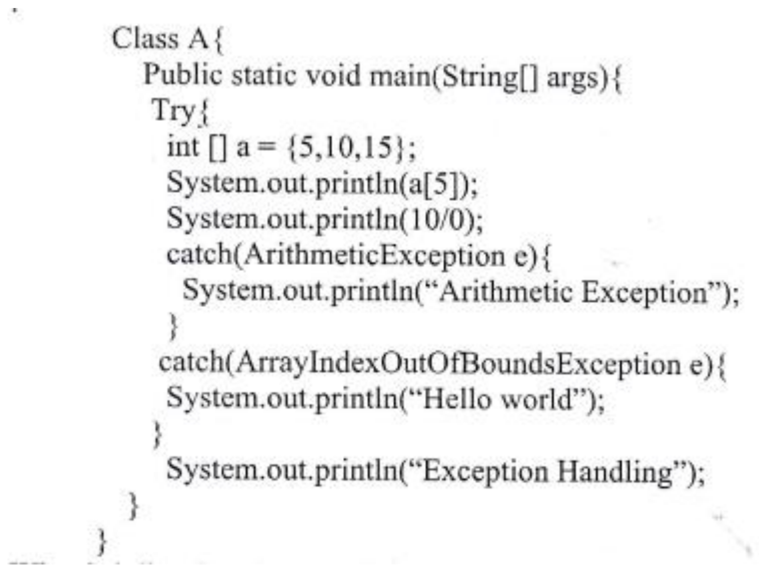
Answer:
Exception handling is a mechanism in Java that allows for the proper handling of exceptions that occur during the execution of a program. It involves identifying, catching, and handling exceptions to prevent program termination and provide appropriate error handling.

Here's an example of how you can manage the ArithmeticException in Java using exception handling:



```java
public class ArithmeticExceptionExample {
    public static void main(String[] args) {
        try {
            int result = divide(10, 0);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic exception: " + e.getMessage());
        }
    }

    public static int divide(int num1, int num2) {
        return num1 / num2;
    }
}
```

Q-11:What happens if an exception is not caught? What will be the output of the following program:



```
Class A{
   Public static void main(String[] args){
    Try{
     int [] a = {5,10,15};
     System.out.println(a[5]);
     System.out.println(10/0);
     catch(ArithmeticException e){
       System.out.println("Arithmetic Exception");
     }
    catch(ArrayIndexOutOfBoundsException e){
     System.out.println("Hello world");
    }
     System.out.println("Exception Handling");
  }
}
```

Answer:

If an exception is not caught and handled, it will propagate up the call stack until it reaches an appropriate exception handler or the program terminates. If no exception handler is found, the default exception handler provided by the Java runtime environment will handle the exception, which typically results in printing a stack trace and terminating the program.

Q-12:What is inline function? Write down the advantages and limitations of inline function.

Answer:

An inline function is a function that is expanded in place at the point where it is called, rather than being executed as a separate function call. In Java, the inline keyword does not exist, but some methods can be considered inline if they are marked final or private and are small enough to be optimized by the compiler.

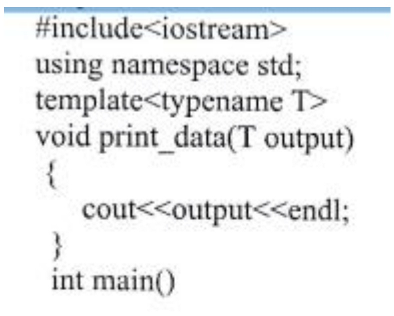
Advantages of inline functions:

- Reduces function call overhead, resulting in faster code execution.
- Can improve cache locality by reducing the number of memory accesses required.

Limitations of inline functions:

- Increases code size, which can decrease performance due to increased cache misses and instruction pipeline stalls.
- Can cause code bloat if used excessively, making the program larger and harder to maintain.
- May not be suitable for all functions, such as those that are complex or require error handling.

Q-13:What is template? What would be the output of the following program :

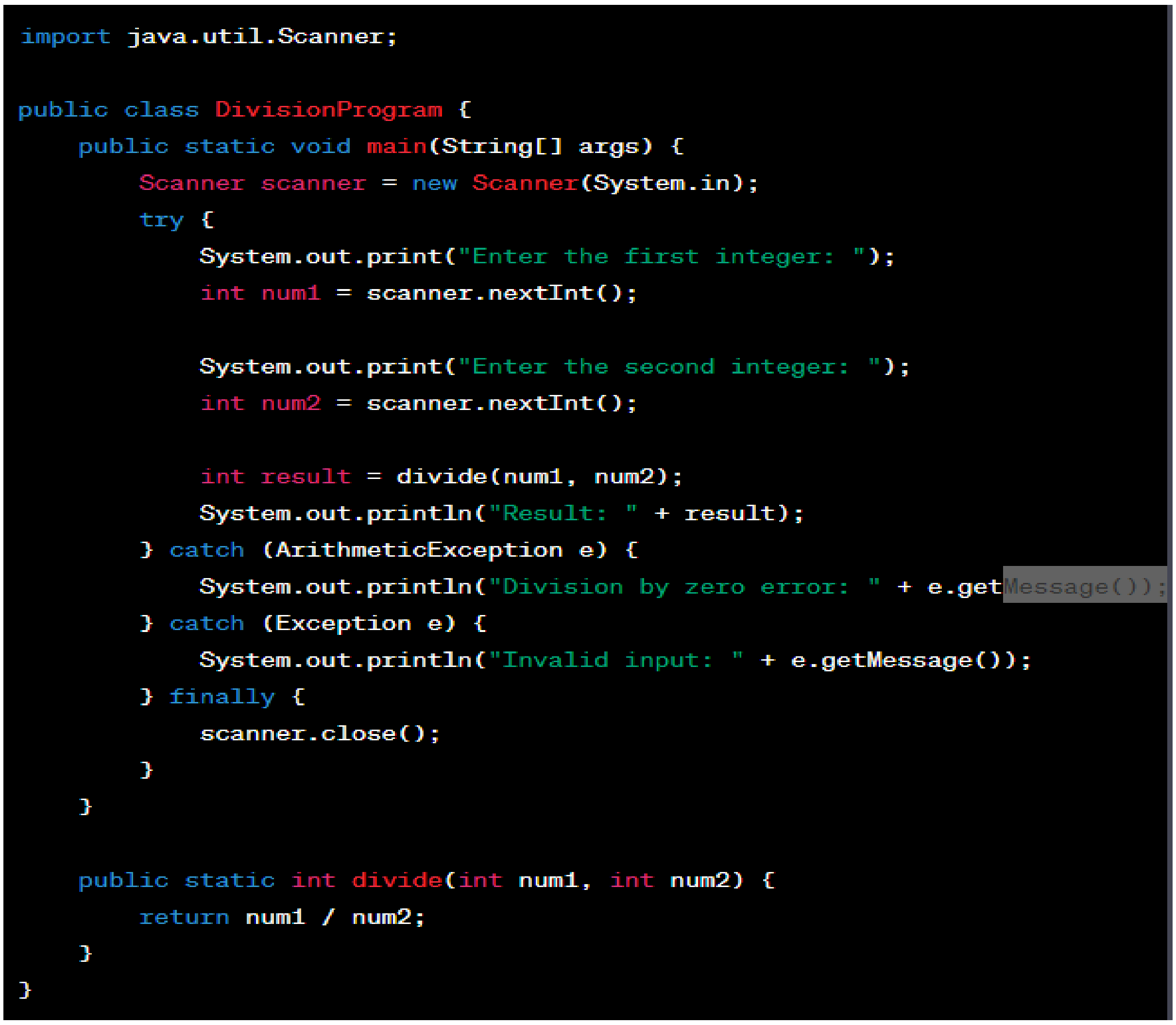


```
#include<iostream>
using namespace std;
template<typename T>
void print_data(T output)
 {
    cout<<output<<endl;
 }
 int main()
```

Answer: A template is a generic class or method that can work with any data type. Templating is a technique that allows you to write a single implementation of a class or method that can operate on multiple types.

Q-16:Write a program that takes two integers from keyboard to perform division operation. A try block to throw an exception when a wrong type of data is keyed or division by zero is occurred. Also write appropriate catch block to handle the exception thrown.

Answer: Here's an example program that takes two integers from the keyboard and performs a division operation, using a try block to handle exceptions:



```java
import java.util.Scanner;

public class DivisionProgram {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.print("Enter the first integer: ");
            int num1 = scanner.nextInt();

            System.out.print("Enter the second integer: ");
            int num2 = scanner.nextInt();

            int result = divide(num1, num2);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Division by zero error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Invalid input: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    public static int divide(int num1, int num2) {
        return num1 / num2;
    }
}
```

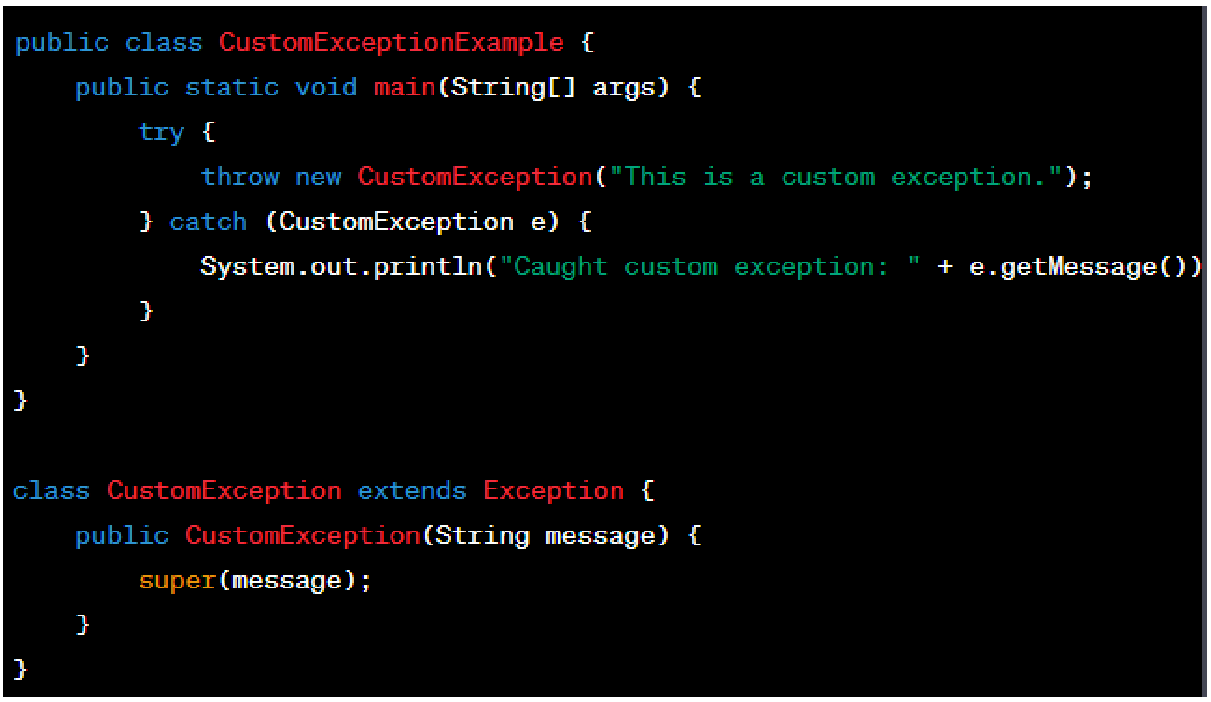
In this program, we use the Scanner class to read two integers from the keyboard. The division operation is performed within the try block. If a division by zero occurs, an ArithmeticException is thrown. If the user enters an invalid input, such as a non-integer value, an InputMismatchException is thrown.

We have separate catch blocks to handle the ArithmeticException and the general Exception (which includes InputMismatchException). The appropriate error messages are displayed for each exception. The finally block is used to close the Scanner object to release any associated resources.

Q-16:How can you define your own exception type in java? Explain with example.

Answer:

you can define your own exception types by creating a custom exception class that extends the `Exception` class or one of its subclasses. Here's an example:



```
public class CustomExceptionExample {
    public static void main(String[] args) {
        try {
            throw new CustomException("This is a custom exception.");
        } catch (CustomException e) {
            System.out.println("Caught custom exception: " + e.getMessage())
        }
    }
}

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

In this example, we define a custom exception class CustomException that extends the Exception class. The CustomException class has a constructor that takes a String parameter representing the exception message. Inside the main() method, we throw an instance of CustomException using the throw keyword.

In the catch block, we catch the CustomException and print the exception message. By defining a custom exception class, we can create and handle exceptions specific to our application's requirements.

Q-17:What happens when a class member is declared as static? What restrictions are applied to a method that is declared as static?

Answer:

When a class member is declared as static in Java, it means that the member belongs to the class itself rather than an instance of the class. Here's what happens when a class member (field or method) is declared as static:

Field (Static Variable):

- A static field is shared among all instances of the class. It is stored in a single memory location and can be accessed directly through the class name.
- It is initialized only once when the class is loaded into memory and retains its value throughout the program's execution.
- All instances of the class share the same value of the static field.

Method (Static Method):

- A static method belongs to the class and can be invoked using the class name, without creating an instance of the class.
- It can access only static members of the class, including static fields and other static methods. It cannot access non-static (instance) members directly.
- Static methods cannot be overridden in the subclass, but they can be redefined in the subclass with a method of the same signature (hiding the superclass's static method).
- They are commonly used for utility methods or methods that operate on class-level data.

Restrictions applied to a method that is declared as static:

- Static methods cannot access non-static (instance) members directly. They can only access other static members (fields or methods) of the class.
- Static methods cannot use the this keyword, as there is no instance associated with them.
- Static methods cannot be overridden in the subclass. However, they can be redefined with a method of the same signature, thus hiding the superclass's static method.
- Static methods can be called directly using the class name, without creating an instance of the class.

It's important to note that static members are shared across all instances of the class and are not tied to any specific instance. They are useful for defining behaviour's or data that are common to all instances or for utility methods that do not require access to instance-specific data.

MD RAIHAN ALI_63
CSE_02
Faculty of Engineering & Technology
University of Dhaka (NITER)