NAME:-NILESH FATE\_JH

ASSIGNMENT\_02\_(SANDEEP\_SIR)

Q1.1. Working with java.lang.Boolean

A. **a.** Explore the [Java API documentation for java.lang.Boolean](https://docs.oracle.com/javase/8/docs/api/java/lang/Boolean.html) and observe its modifiers and super types.

The java.lang.Boolean class in Java is a wrapper class that represents the primitive boolean data type as an object. It is part of the java.lang package, which is automatically imported into all Java programs. Let's explore the API documentation for java.lang.Boolean and observe its modifiers, super types, and some other characteristics

1. Modifiers

* Class Modifier: public final
  + The Boolean class is declared as public, meaning it is accessible from anywhere in the program.
  + The class is also final, meaning it cannot be subclassed (extended). This is typical for wrapper classes in Java, as they are immutable and represent primitive types.
* Field Modifiers:
  + public static final: For constants like TRUE and FALSE, which represent the Boolean values true and false.
  + Example:

**public static final Boolean TRUE = new Boolean(true);**

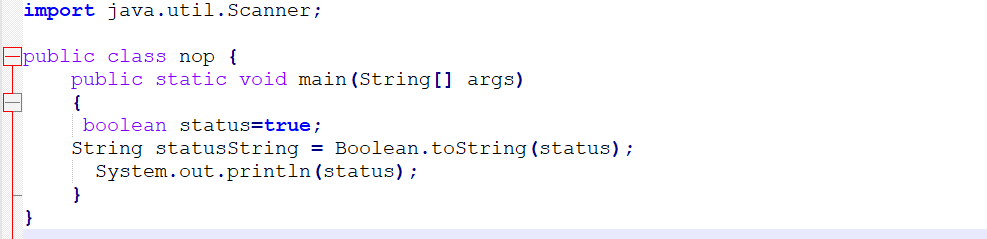
**public static final Boolean FALSE = new Boolean(false);**

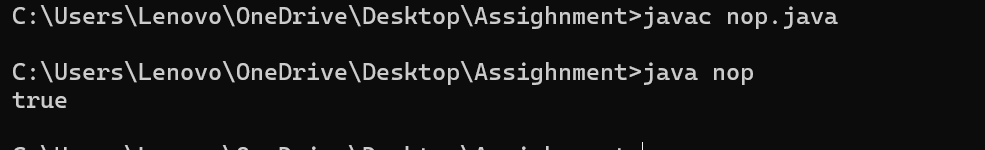
* + These constants are provided for convenience, and they refer to singleton instances of the Boolean class.
* Method Modifiers:
  + Most methods in the Boolean class are declared public static (like parseBoolean()), since they are utility methods that don't require an instance of Boolean to be called.
  + There are also public instance methods like booleanValue() which returns the primitive boolean value of a Boolean object.

2. Super Types

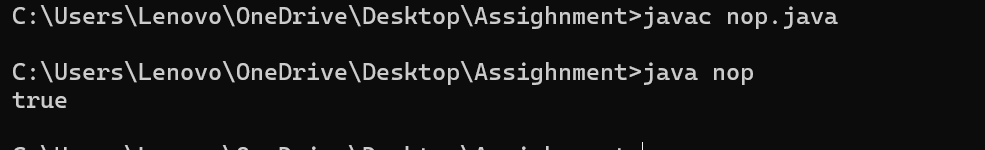
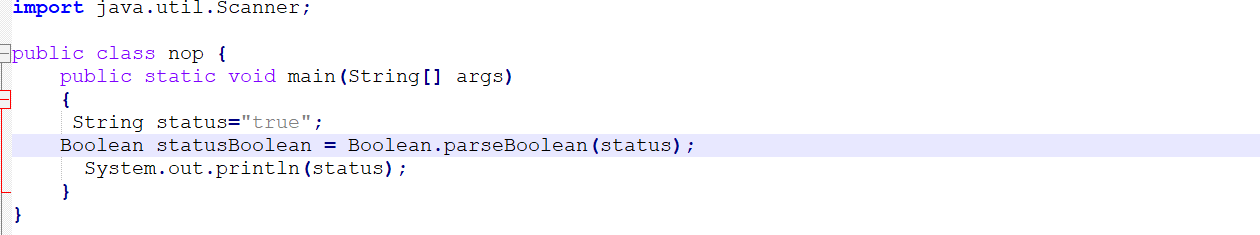
* The Boolean class extends java.lang.Object and implements two interfaces:
  + java.io.Serializable: This allows Boolean objects to be serialized (converted into a byte stream) for purposes such as saving to a file or transferring over a network.
  + java.lang.Comparable<Boolean>: This interface allows Boolean objects to be compared with one another. It defines the compareTo() method, which compares Boolean objects in terms of true (greater than) and false (less than).

**b.** Declare a method-local variable status of type boolean with the value true and convert it to a String using the toString method. (Hint: Use Boolean.toString(Boolean) ).

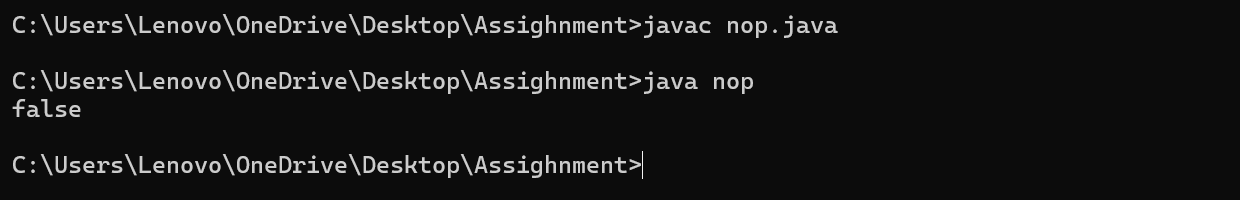
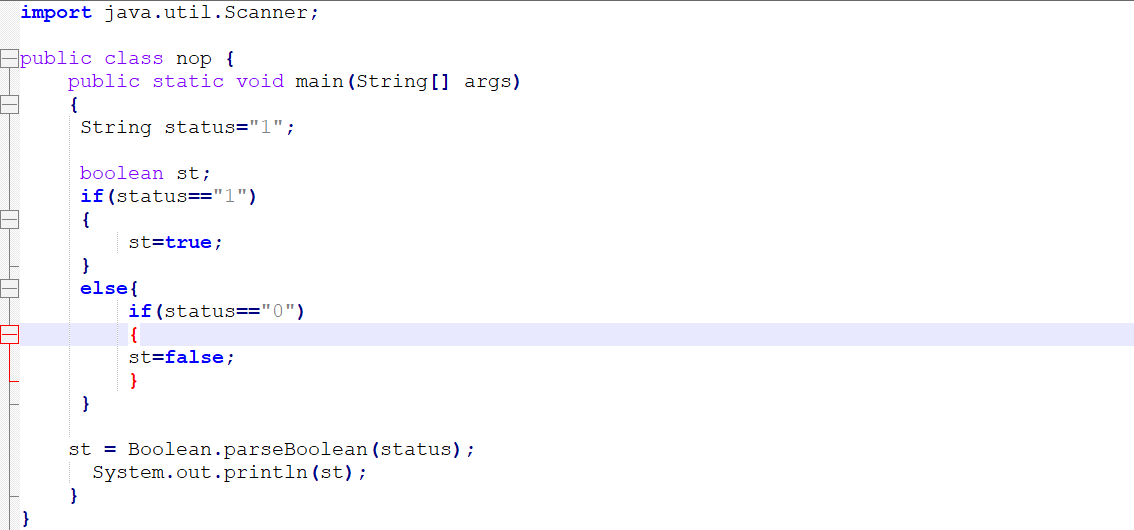




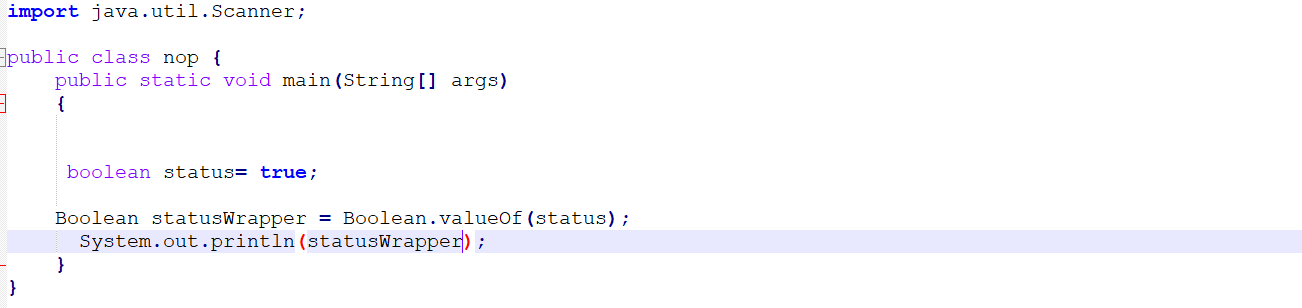
**c.** Declare a method-local variable strStatus of type String with the value "true" and convert it to a boolean using the parseBoolean method. (Hint: Use Boolean.parseBoolean(String)).

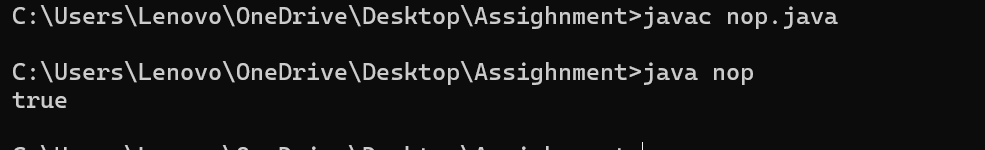


**d.** Declare a method-local variable strStatus of type String with the value "1" or "0" and attempt to convert it to a boolean. (Hint: parseBoolean method will not work as expected with "1" or "0").

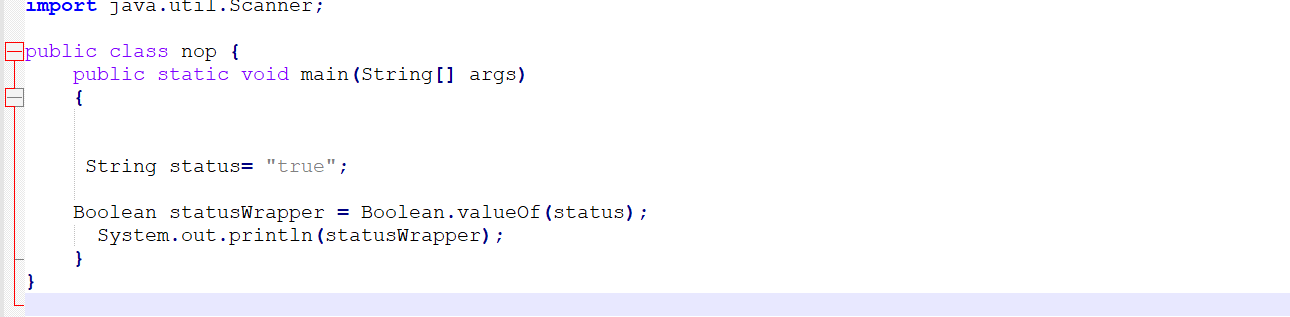


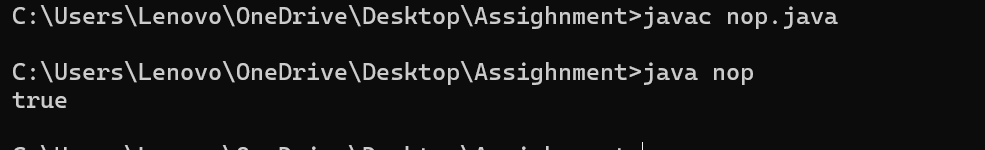
e. Declare a method-local variable status of type boolean with the value true and convert it to the corresponding wrapper class using Boolean.valueOf(). (Hint: Use Boolean.valueOf(boolean)).



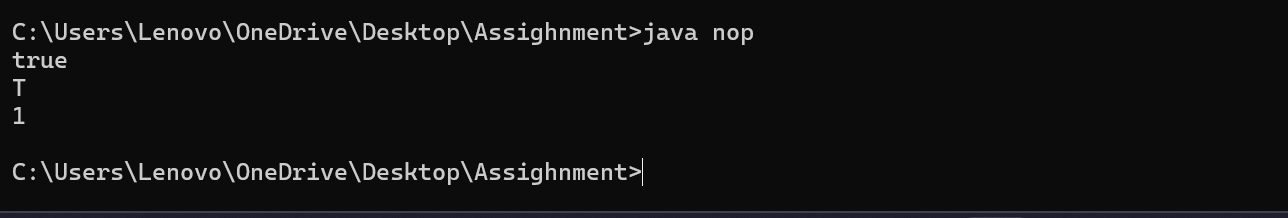
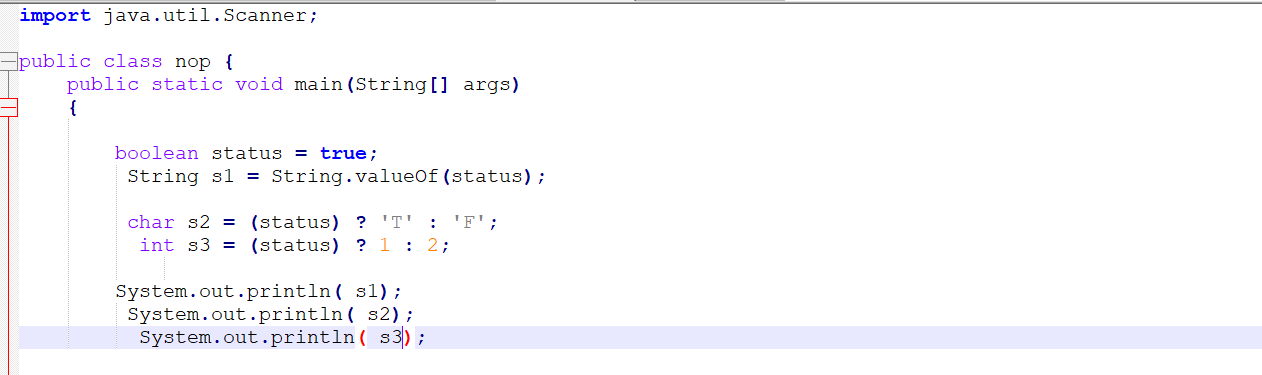


**f.** Declare a method-local variable strStatus of type String with the value "true" and convert it to the corresponding wrapper class using Boolean.valueOf(). (Hint: Use Boolean.valueOf(String)).





**g.** Experiment with converting a boolean value into other primitive types or vice versa and observe the results.



**2. Working with java.lang.Byte**

**a.** Explore the [Java API documentation for java.lang.Byte](https://docs.oracle.com/javase/8/docs/api/java/lang/Byte.html) and observe its modifiers and super types.

To explore the Java API documentation for java.lang.Byte, let's go over its key aspects, such as its modifiers, super types, and other relevant details:

**1. Class Declaration:**

The java.lang.Byte class is a **wrapper class** for the primitive type byte. It provides methods for working with a byte as an object, instead of as a primitive data type.

Here's the class declaration from the API:

java

Copy code

public final class Byte extends Number implements Comparable<Byte>

**2. Modifiers:**

* **public**: The class is accessible from anywhere.
* **final**: The class cannot be subclassed, meaning no other class can extend Byte.

**3. Super Types:**

The java.lang.Byte class extends and implements the following classes and interfaces:

* **Superclass**:
  + java.lang.Number: This is an abstract class that provides methods to convert the wrapped value into different numerical types (like int, long, float, etc.).
* **Interfaces**:
  + **Comparable<Byte>**: This interface allows Byte objects to be compared with one another. It enables methods like compareTo(Byte anotherByte), which can be used for sorting or ordering.

**4. Constructors:**

* **Byte(byte value)**: Constructs a Byte object with the given byte value.
* **Byte(String s)**: Constructs a Byte object by parsing the given String value as a byte.

**5. Common Methods:**

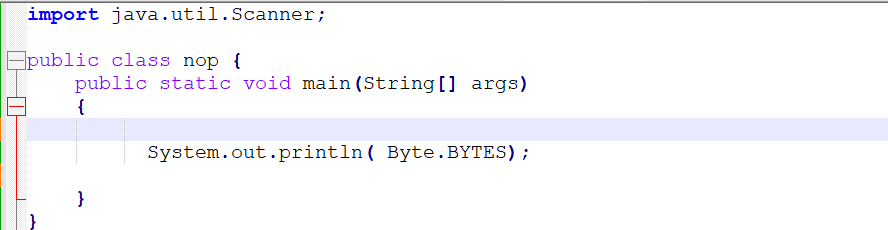
Here are some commonly used methods in the Byte class:

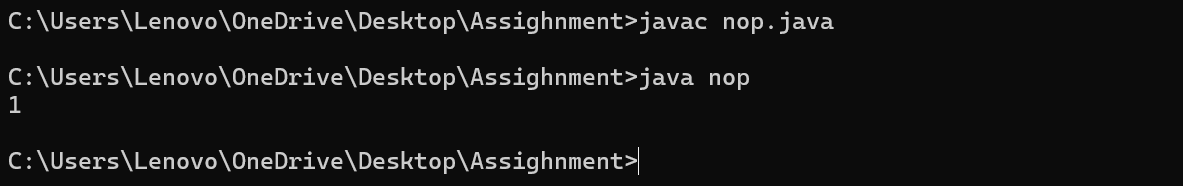
* **byteValue()**: Returns the value of this Byte object as a byte.
* **intValue()**: Returns the value of this Byte object as an int.
* **compareTo(Byte anotherByte)**: Compares two Byte objects.
* **valueOf(byte b)**: Returns a Byte instance representing the specified byte value.
* **parseByte(String s)**: Parses the String argument as a byte.
* **toString()**: Returns the String representation of the Byte object.

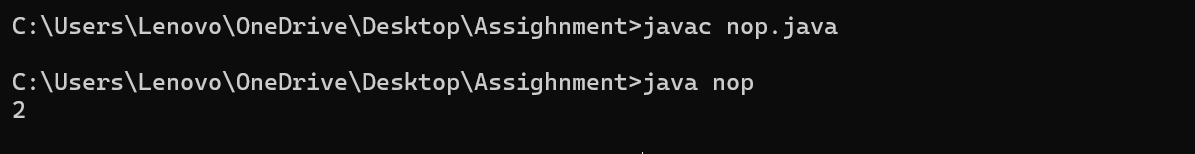
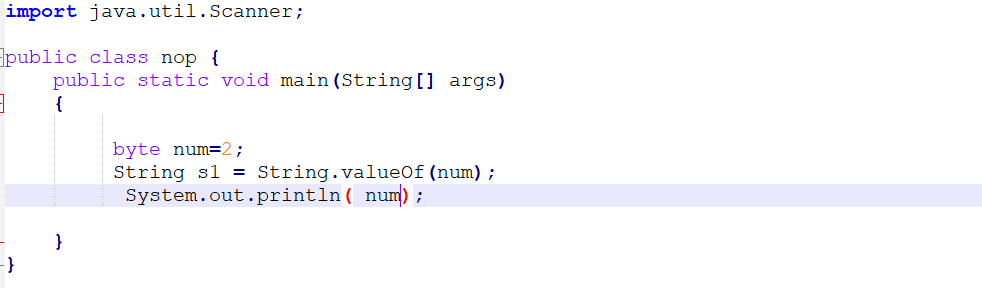
**6. Static Fields:**

* **MAX\_VALUE**: The maximum value a byte can hold, which is 127.
* **MIN\_VALUE**: The minimum value a byte can hold, which is -128.
* **SIZE**: The number of bits used to represent a byte value, which is 8.
* **Modifiers**: public final
* **Superclass**: java.lang.Number
* **Interfaces**: Comparable<Byte>
* **Key Methods**: byteValue(), intValue(), compareTo(), valueOf(), parseByte(), toString()
* **Static Fields**: MAX\_VALUE, MIN\_VALUE, SIZE

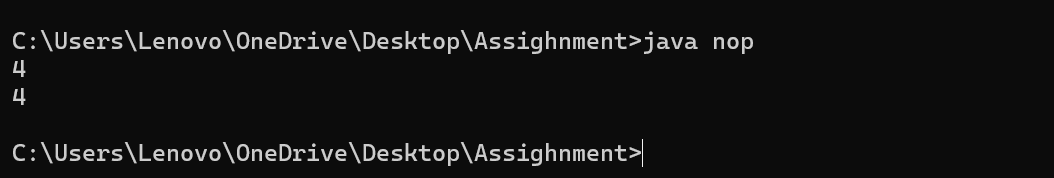
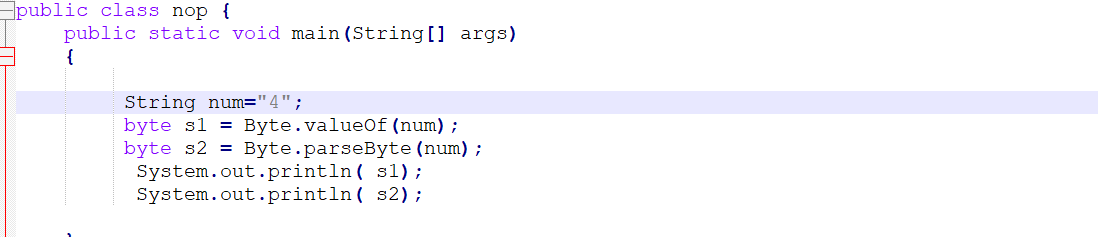
The Byte class allows manipulation of the primitive byte type as an object, adding functionality like comparison and type conversion, while also providing a convenient way to deal with byte values.

**b.** Write a program to test how many bytes are used to represent a byte value using the BYTES field. (Hint: Use Byte.BYTES). 

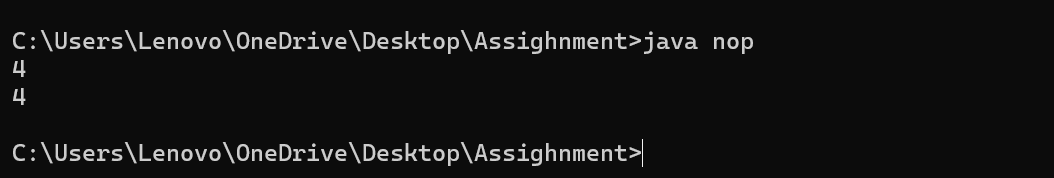
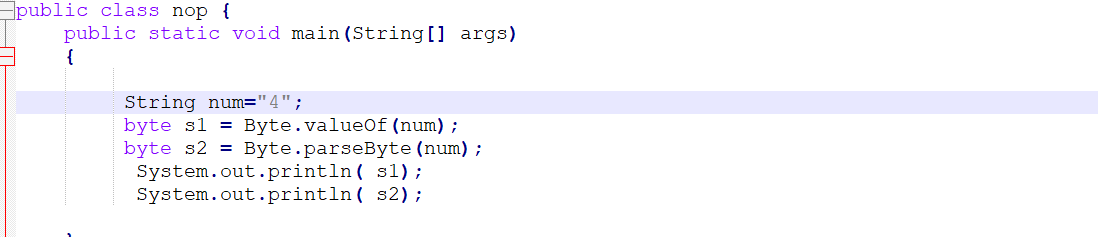


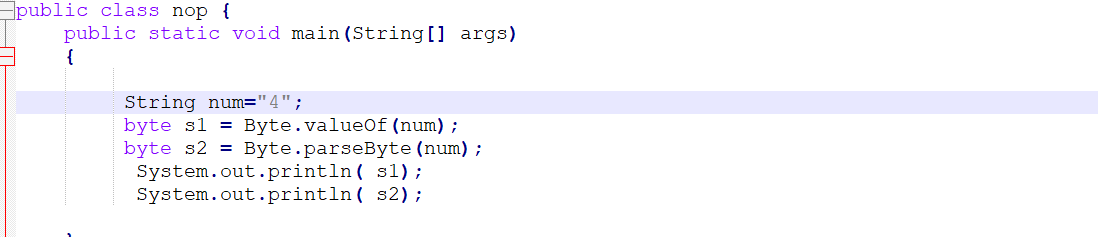
**c.** Write a program to find the minimum and maximum values of byte using the MIN\_VALUE and MAX\_VALUE fields. (Hint: Use Byte.MIN\_VALUE and Byte.MAX\_VALUE).

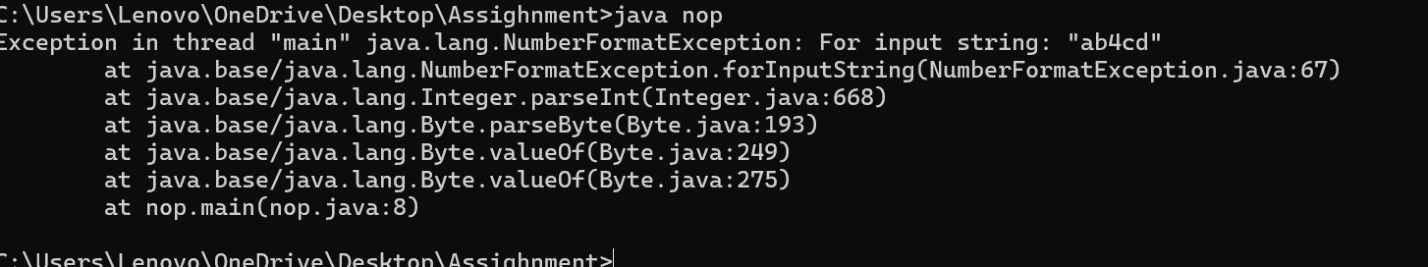
**d.** Declare a method-local variable number of type byte with some value and convert it to a String using the toString method. (Hint: Use Byte.toString(byte)).

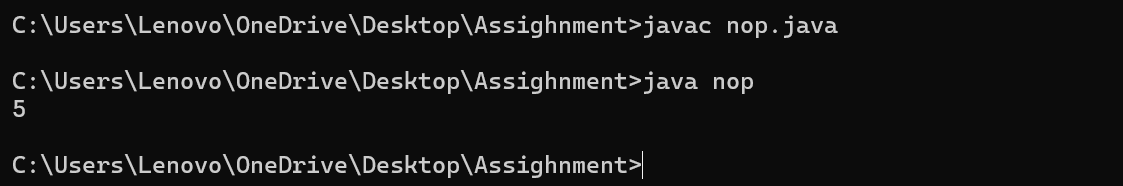
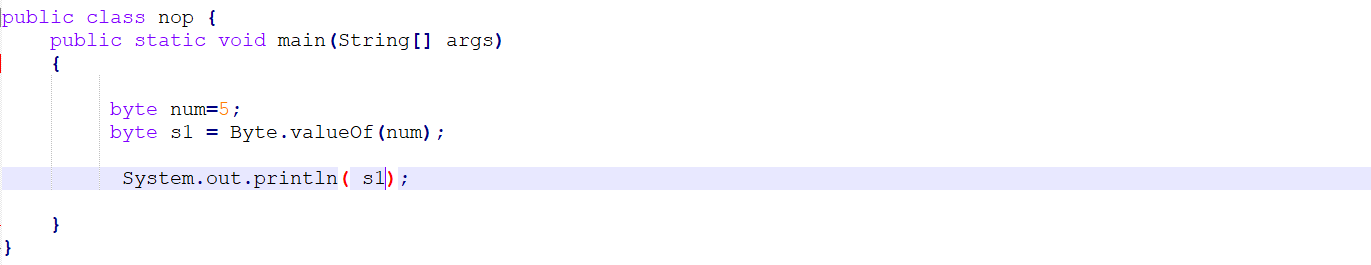


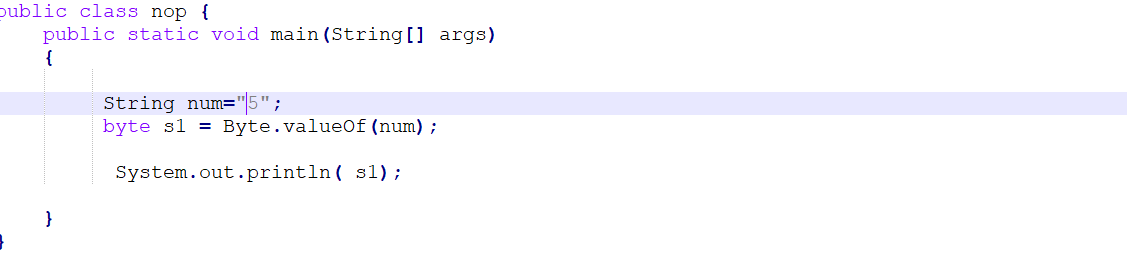
**e.** Declare a method-local variable strNumber of type String with some value and convert it to a byte value using the parseByte method. (Hint: Use Byte.parseByte(String)).

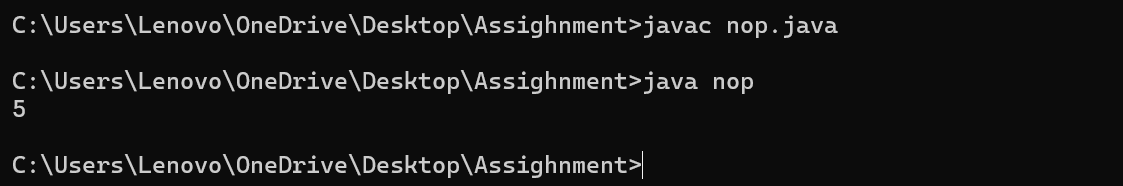


**f.** Declare a method-local variable strNumber of type String with the value "Ab12Cd3" and attempt to convert it to a byte value. (Hint: parseByte method will throw a NumberFormatException). 

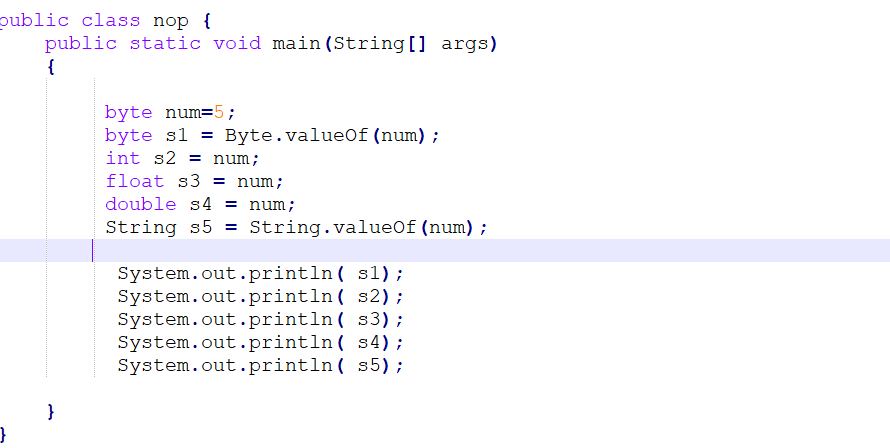


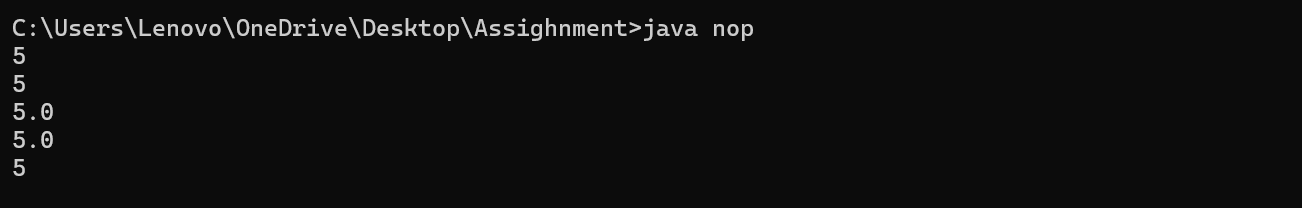
**g.** Declare a method-local variable number of type byte with some value and convert it to the corresponding wrapper class using Byte.valueOf(). (Hint: Use Byte.valueOf(byte)). 

**h.** Declare a method-local variable strNumber of type String with some byte value and convert it to the corresponding wrapper class using Byte.valueOf(). (Hint: Use Byte.valueOf(String)). 



**i.** Experiment with converting a byte value into other primitive types or vice versa and observe the results.





**3. Working with java.lang.Short**

**a.** Explore the [Java API documentation for java.lang.Short](https://docs.oracle.com/javase/8/docs/api/java/lang/Short.html) and observe its modifiers and super types.

To explore the java.lang.Short class, let’s look at its modifiers, super types, and other important characteristics:

**1. Class Declaration:**

The java.lang.Short class is a **wrapper class** for the primitive data type short. It allows you to treat short values as objects, with added methods for conversion and manipulation.

java

Copy code

public final class Short extends Number implements Comparable<Short>

**2. Modifiers:**

* **public**: The class is accessible from anywhere.
* **final**: The class cannot be subclassed, meaning no other class can extend Short.

**3. Super Types:**

The Short class extends and implements the following:

* **Superclass**:
  + java.lang.Number: This is an abstract class that provides methods to convert the numeric value into different types (like int, long, float, etc.).
* **Interfaces**:
  + **Comparable<Short>**: This interface allows Short objects to be compared with one another, enabling comparison operations like compareTo(Short anotherShort).

**4. Fields (Static Constants):**

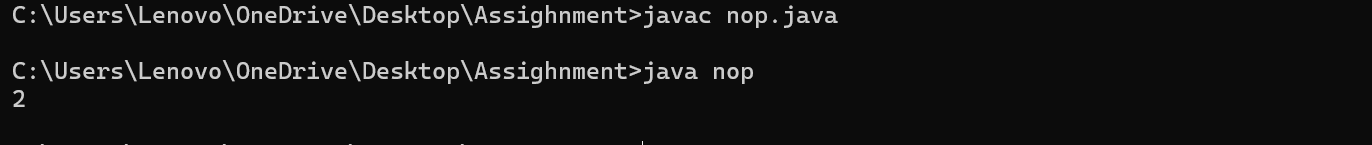
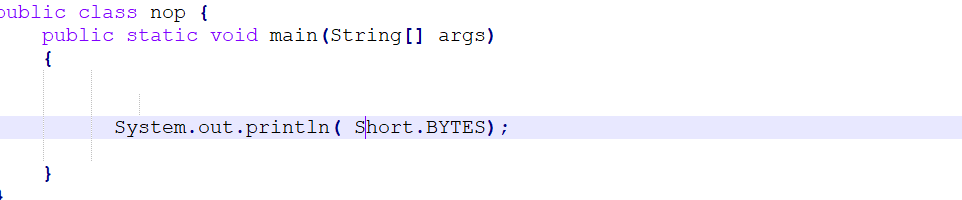
* **MIN\_VALUE**: The minimum value a short can have, which is -32,768.
* **MAX\_VALUE**: The maximum value a short can have, which is 32,767.
* **SIZE**: The number of bits used to represent a short, which is 16.
* **BYTES**: The number of bytes used to represent a short, which is 2.

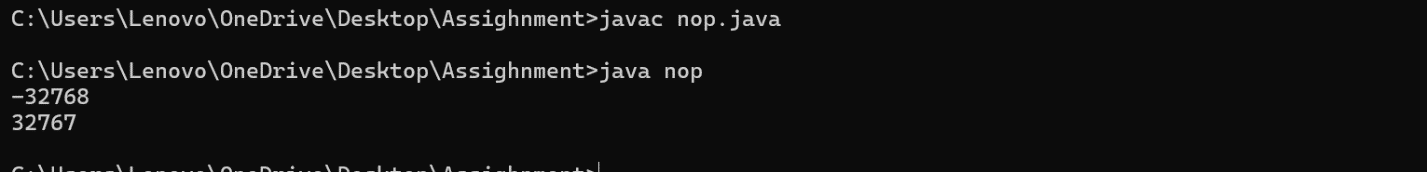
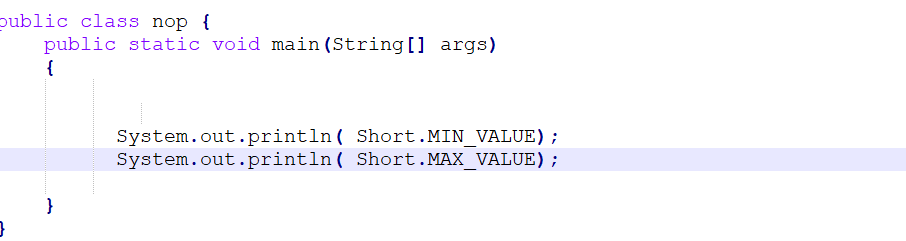
**5. Constructors:**

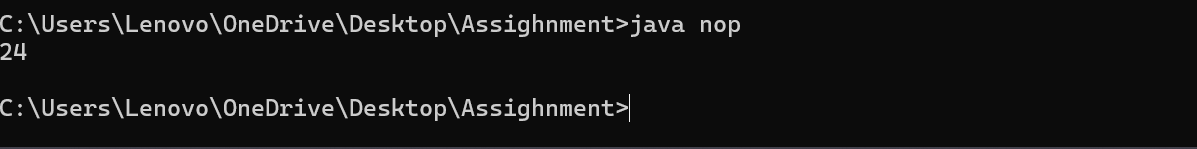
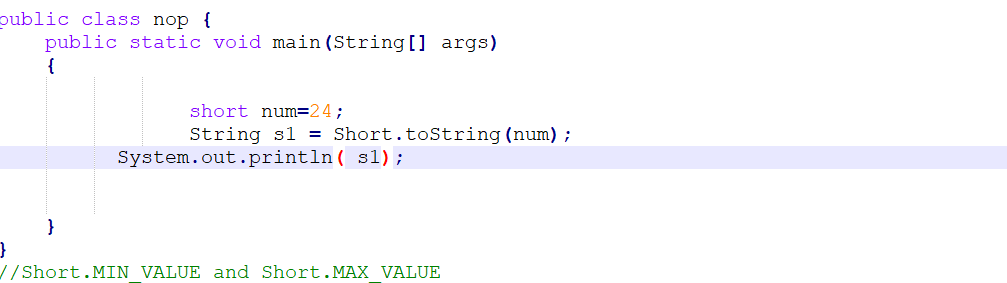
* **Short(short value)**: Constructs a new Short object with the specified short value.
* **Short(String s)**: Constructs a Short object by parsing the given string as a short value.

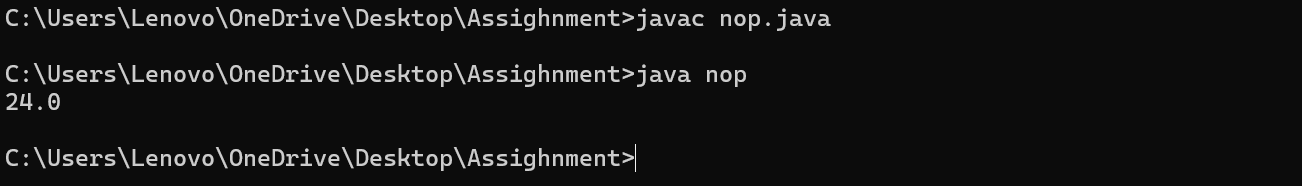
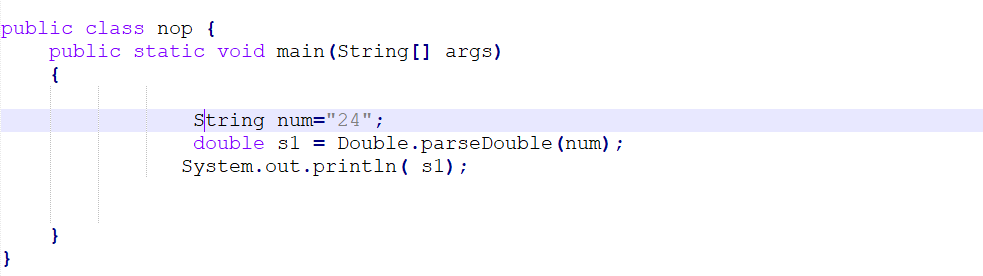
**6. Common Methods:**

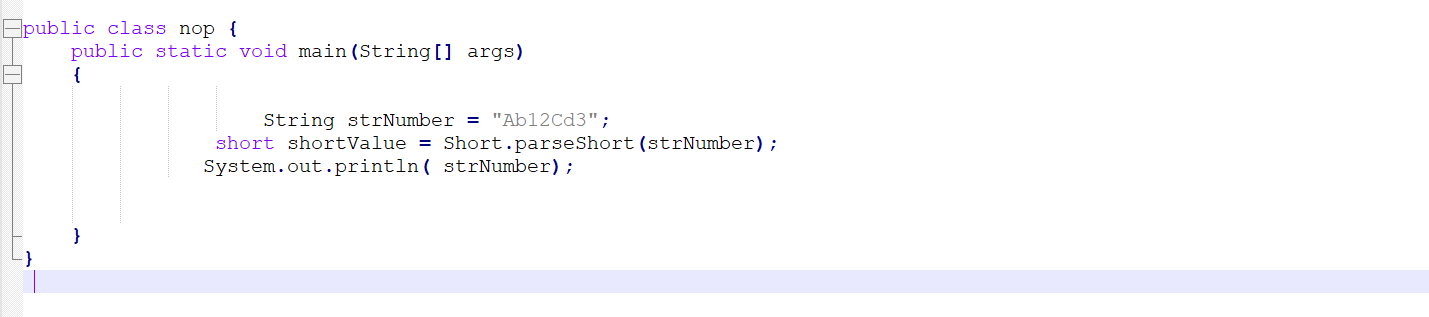
* **shortValue()**: Returns the value of this Short object as a short.
* **intValue()**: Returns the value of this Short object as an int.
* **longValue()**: Returns the value of this Short object as a long.
* **compareTo(Short anotherShort)**: Compares this Short object with another.
* **valueOf(short s)**: Returns a Short instance representing the specified short value.
* **parseShort(String s)**: Parses the string argument as a short value.
* **toString()**: Returns the String representation of the Short object.

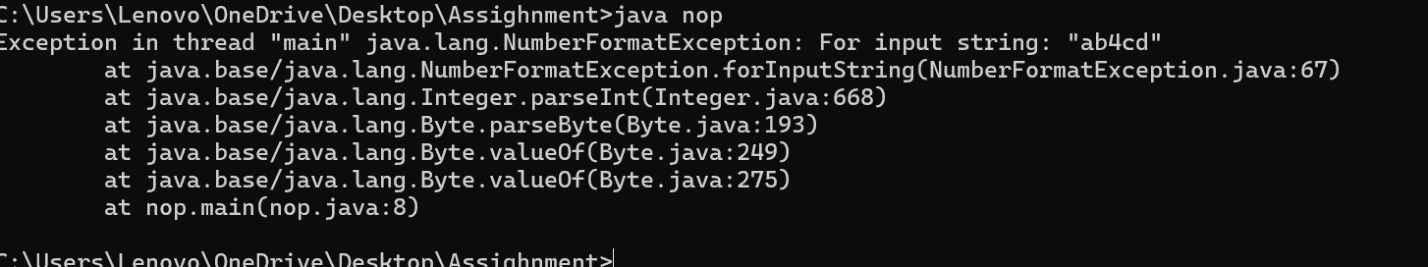
**b.** Write a program to test how many bytes are used to represent a short value using the BYTES field. (Hint: Use Short.BYTES).

**c.** Write a program to find the minimum and maximum values of short using the MIN\_VALUE and MAX\_VALUE fields. (Hint: Use Short.MIN\_VALUE and Short.MAX\_VALUE).

**d.** Declare a method-local variable number of type short with some value and convert it to a String using the toString method. (Hint: Use Short.toString(short)). 

**e.** Declare a method-local variable strNumber of type String with some value and convert it to a short value using the parseShort method. (Hint: Use Short.parseShort(String)). 

**f.** Declare a method-local variable strNumber of type String with the value "Ab12Cd3" and attempt to convert it to a short value. (Hint: parseShort method will throw a NumberFormatException).



**g.** Declare a method-local variable number of type short with some value and convert it to the corresponding wrapper class using Short.valueOf(). (Hint: Use Short.valueOf(short)).

**public class Main {**

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

**short number = 10;**

**Short wn = Short.valueOf(number);**

**Short.valueOf(short)**

**System.out.println("Wrapper class value (Short): " + wn);**

**}**

**}**

**h.** Declare a method-local variable strNumber of type String with some short value and convert it to the corresponding wrapper class using Short.valueOf(). (Hint: Use Short.valueOf(String)).

**public class Main {**

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

**String strNumber = "20";**

**Short String = Short.valueOf(strNumber);**

**System.out.println("String (Short): " + String);**

**}**

**}**

* 1. Experiment with converting a short value into other primitive types or vice versa and observe the results.

**public class Main {**

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

**short sv = 15;**

**int iv = sv;**

**System.out.println("short to int: " + iv);**

**long lv = sv;**

**System.out.println("short to long: " + lv);**

**float fv= sv;**

**System.out.println("short to float: " + fv);**

**double dv = sv;**

**System.out.println("short to double: " + dv);**

**short newShortFromInt = (short) intValue;**

**System.out.println("int to short: " + newShortFromInt);**

**short newShortFromLong = (short) longValue;**

**System.out.println("long to short: " + newShortFromLong);**

**}**

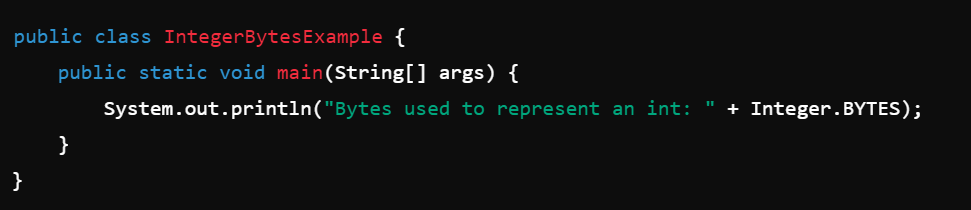
**}**

**4. Working with java.lang.Integer**

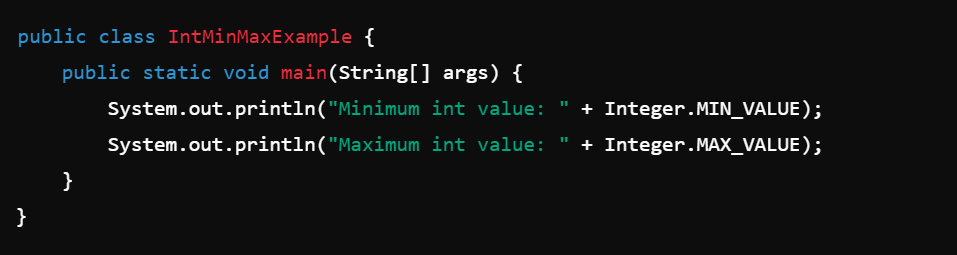
**a. Explore the Java API documentation for java.lang.Integer**

The java.lang.Integer class is a wrapper class for the primitive int data type. It inherits from the Number class and implements the Comparable<Integer> interface. The class provides utility methods for converting between integers and strings, parsing integers, and performing other operations related to the int data type.

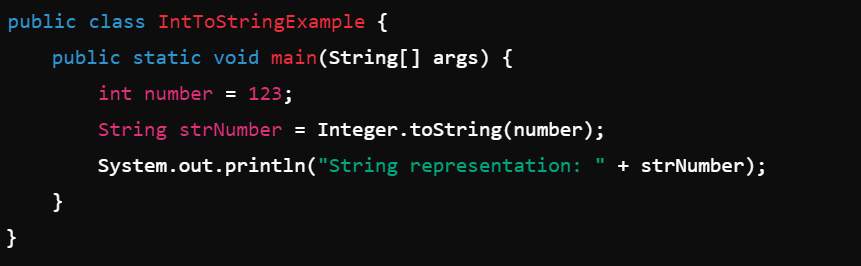
**b.** Write a program to test how many bytes are used to represent an int value using the BYTES field. (Hint: Use Integer.BYTES).



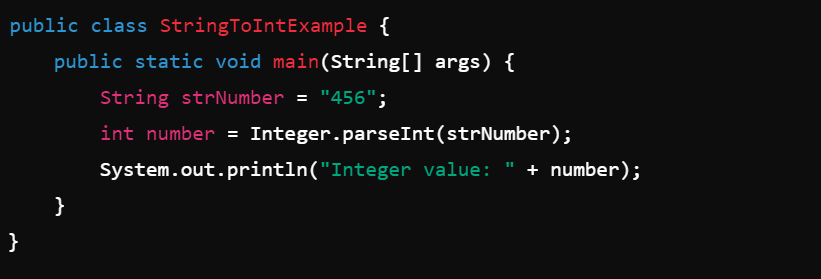
**c.** Write a program to find the minimum and maximum values of int using the MIN\_VALUE and MAX\_VALUE fields. (Hint: Use Integer.MIN\_VALUE and Integer.MAX\_VALUE).



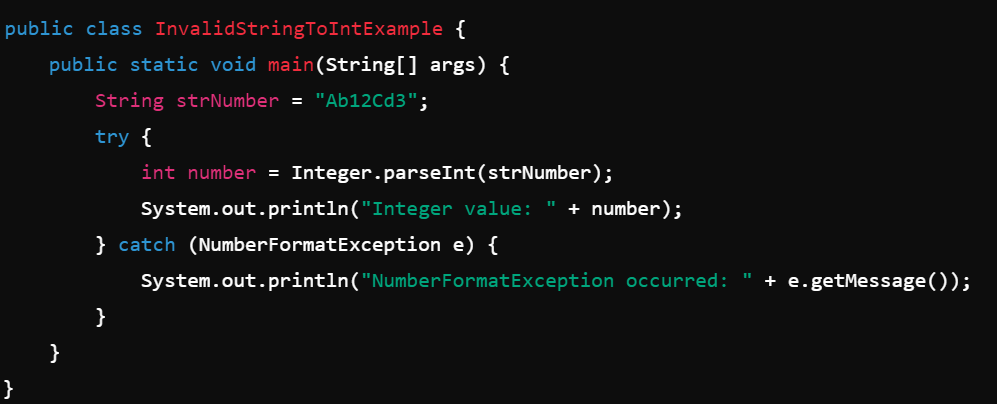
**d.** Declare a method-local variable number of type int with some value and convert it to a String using the toString method. (Hint: Use Integer.toString(int)).



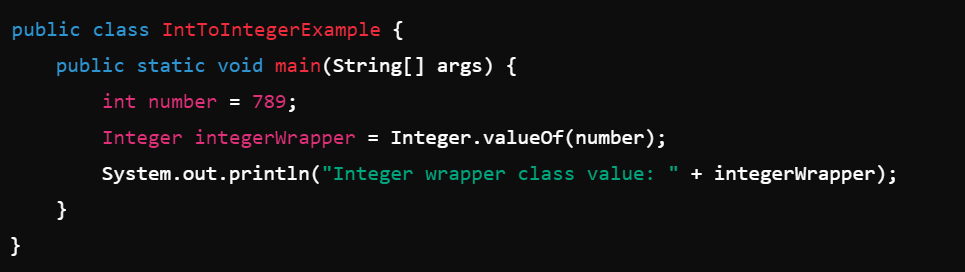
**e.** Declare a method-local variable strNumber of type String with some value and convert it to an int value using the parseInt method. (Hint: Use Integer.parseInt(String)).



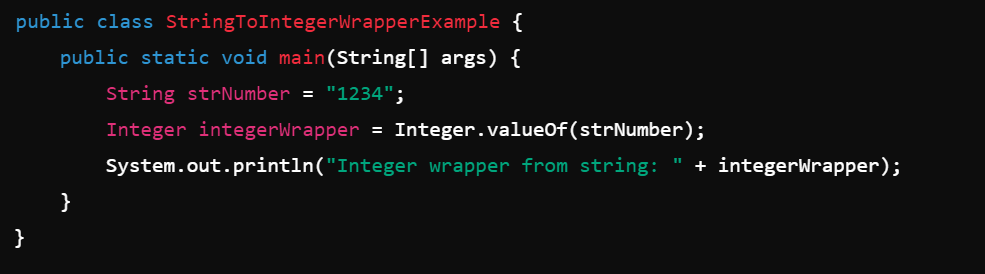
**f.** Declare a method-local variable strNumber of type String with the value "Ab12Cd3" and attempt to convert it to an int value. (Hint: parseInt method will throw a NumberFormatException).



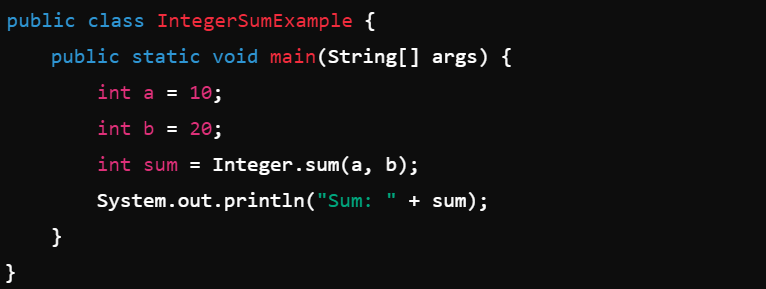
**g.** Declare a method-local variable number of type int with some value and convert it to the corresponding wrapper class using Integer.valueOf(). (Hint: Use Integer.valueOf(int)).



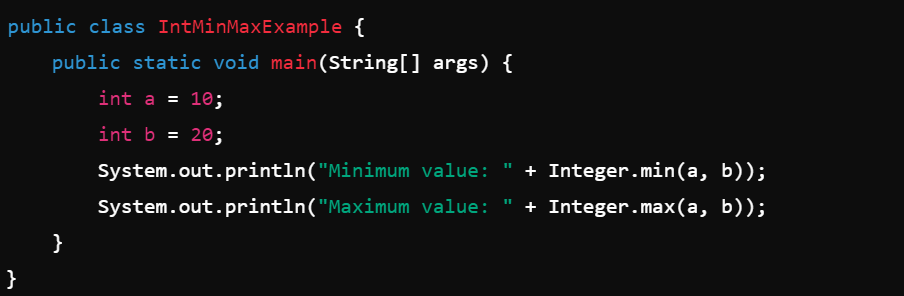
**h.** Declare a method-local variable strNumber of type String with some integer value and convert it to the corresponding wrapper class using Integer.valueOf(). (Hint: Use Integer.valueOf(String)).



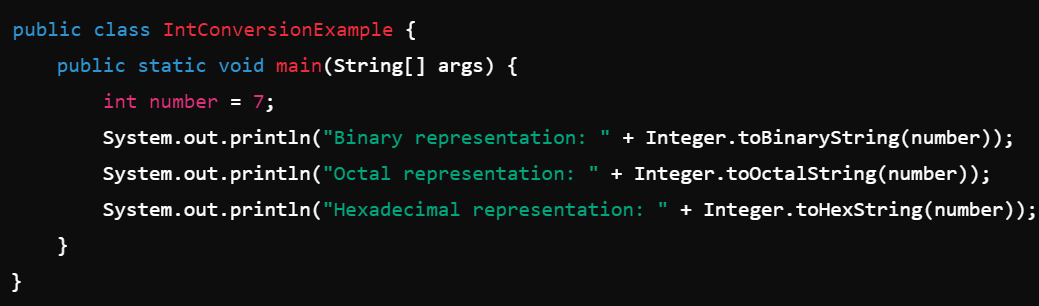
**i.** Declare two integer variables with values 10 and 20, and add them using a method from the Integer class. (Hint: Use Integer.sum(int, int)).



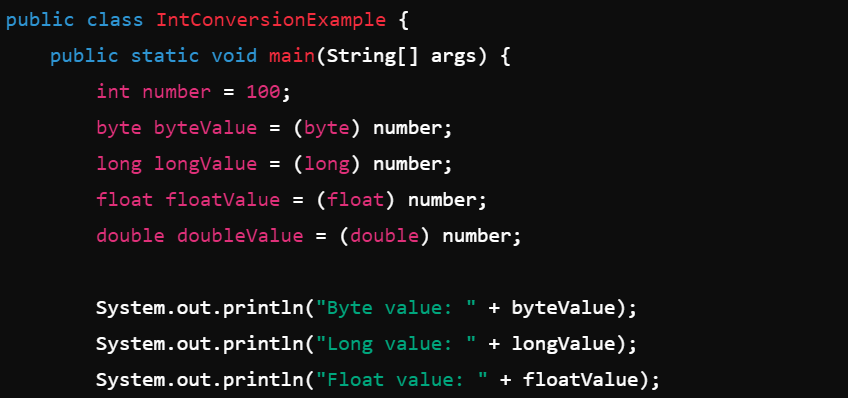
**j.** Declare two integer variables with values 10 and 20, and find the minimum and maximum values using the Integer class. (Hint: Use Integer.min(int, int) and Integer.max(int, int)).



**k.** Declare an integer variable with the value 7. Convert it to binary, octal, and hexadecimal strings using methods from the Integer class. (Hint: Use Integer.toBinaryString(int), Integer.toOctalString(int), and Integer.toHexString(int)).



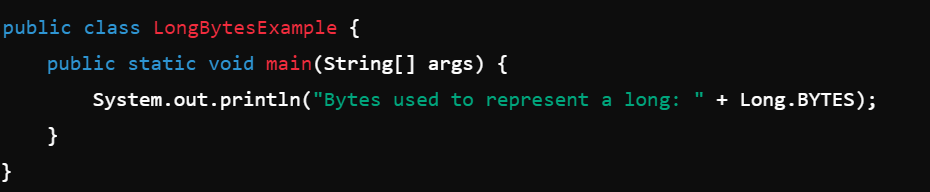
**l.** Experiment with converting an int value into other primitive types or vice versa and observe the results.

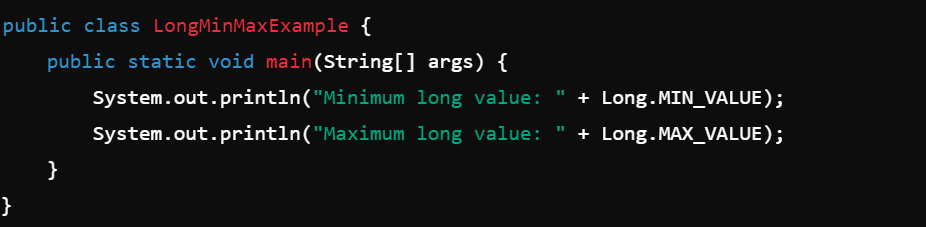


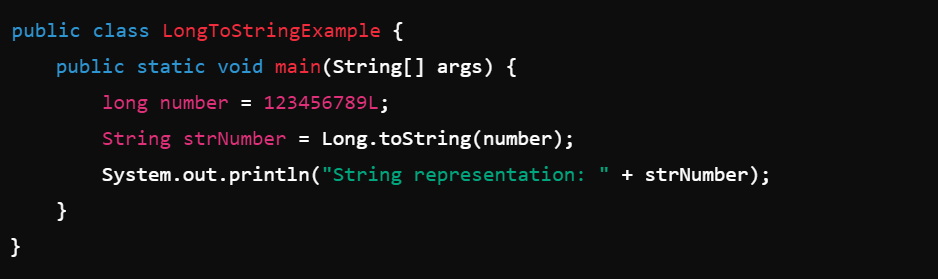
**5. Working with java.lang.Long**

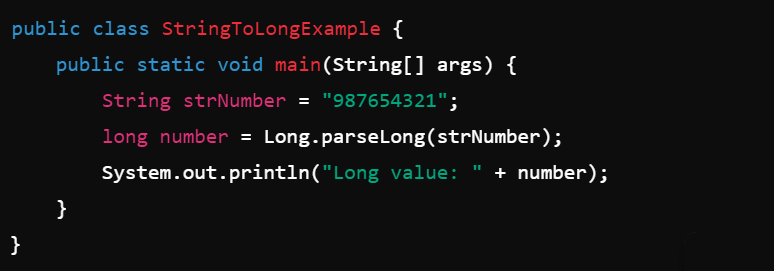
**a. Explore the Java API documentation for java.lang.Long**

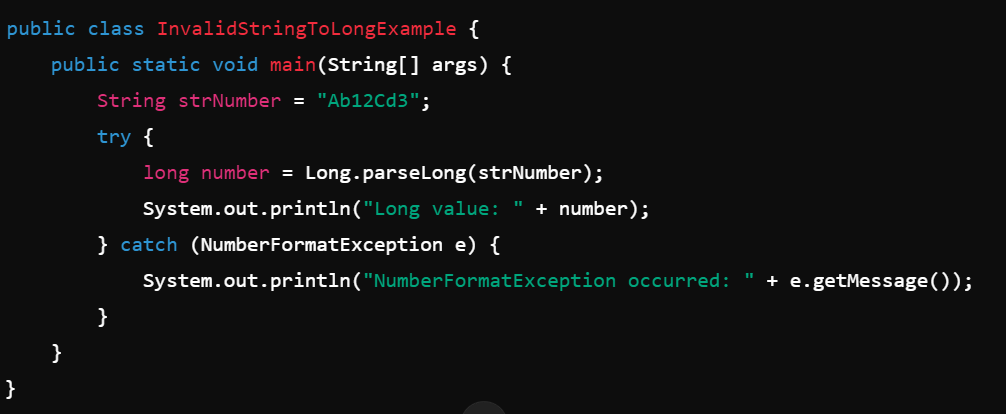
The java.lang.Long class is a wrapper class for the primitive long data type. It extends Number and implements the Comparable<Long> interface. It provides utility methods for converting between long and String, parsing long values, and other useful features. Some important methods include Long.parseLong(String), Long.toString(long), Long.sum(long, long), and more.

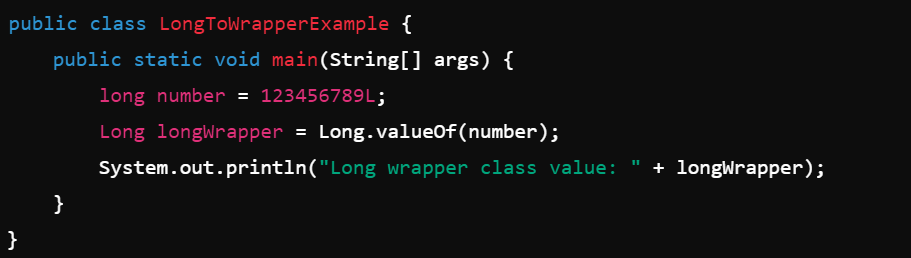
**b.** Write a program to test how many bytes are used to represent a long value using the BYTES field. (Hint: Use Long.BYTES). 

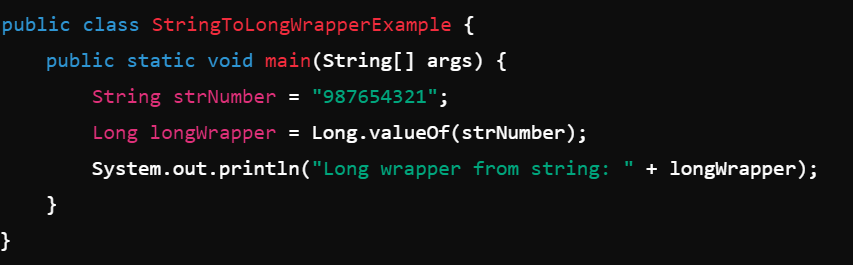
**c.** Write a program to find the minimum and maximum values of long using the MIN\_VALUE and MAX\_VALUE fields. (Hint: Use Long.MIN\_VALUE and Long.MAX\_VALUE). 

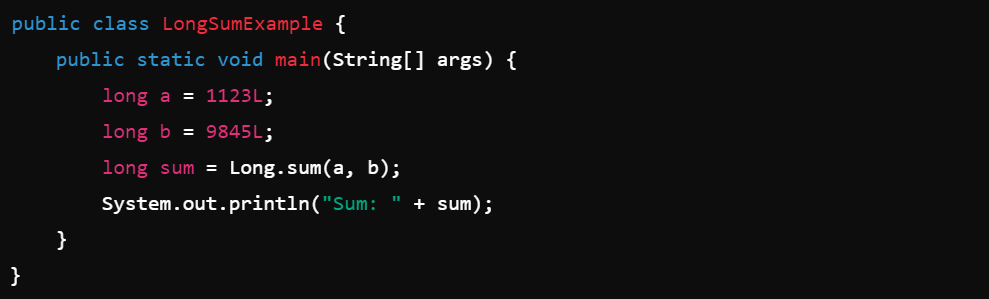
**d.** Declare a method-local variable number of type long with some value and convert it to a String using the toString method. (Hint: Use Long.toString(long)). 

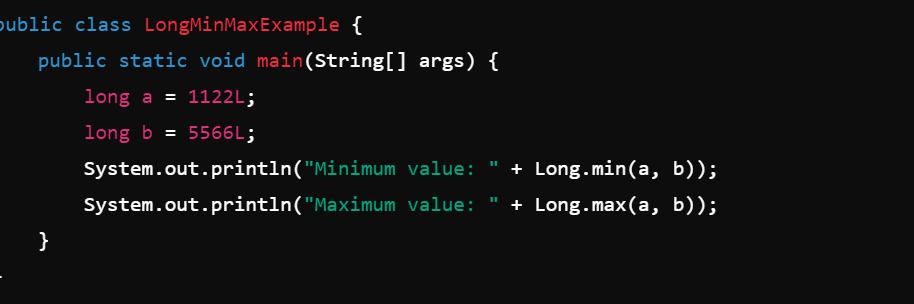
**e.** Declare a method-local variable strNumber of type String with some value and convert it to a long value using the parseLong method. (Hint: Use Long.parseLong(String)). 

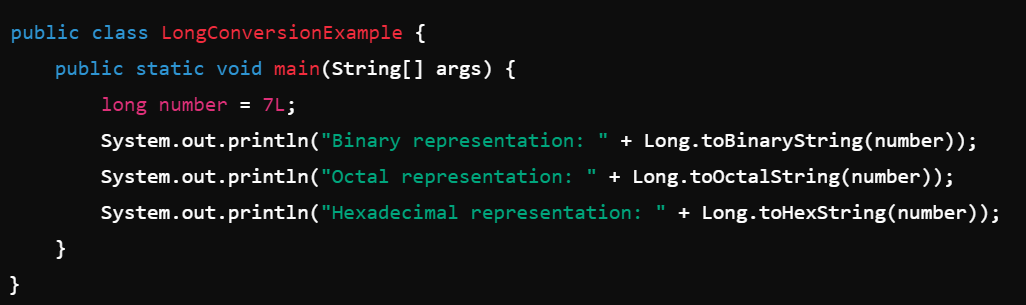
**f.** Declare a method-local variable strNumber of type String with the value "Ab12Cd3" and attempt to convert it to a long value. (Hint: parseLong method will throw a NumberFormatException). 

**g.** Declare a method-local variable number of type long with some value and convert it to the corresponding wrapper class using Long.valueOf(). (Hint: Use Long.valueOf(long)). 

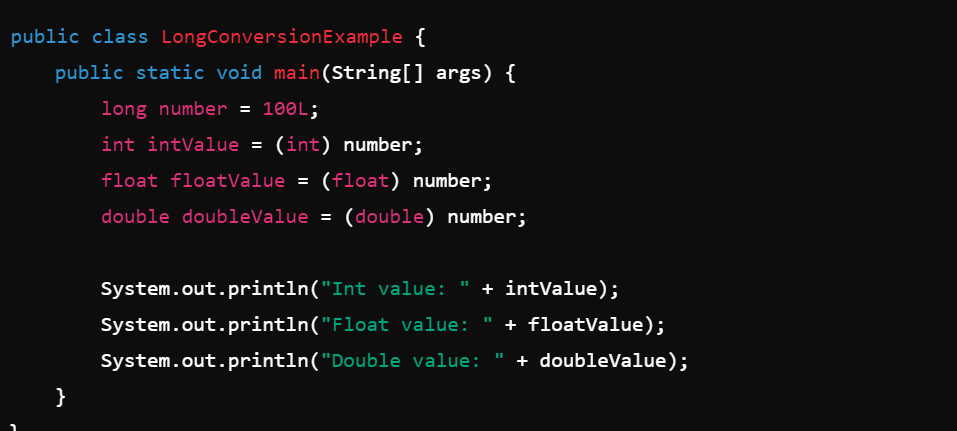
**h.** Declare a method-local variable strNumber of type String with some long value and convert it to the corresponding wrapper class using Long.valueOf(). (Hint: Use Long.valueOf(String)). 

**i.** Declare two long variables with values 1123 and 9845, and add them using a method from the Long class. (Hint: Use Long.sum(long, long)). 

**j.** Declare two long variables with values 1122 and 5566, and find the minimum and maximum values using the Long class. (Hint: Use Long.min(long, long) and Long.max(long, long)). 

**k.** Declare a long variable with the value 7. Convert it to binary, octal, and hexadecimal strings using methods from the Long class. (Hint: Use Long.toBinaryString(long), Long.toOctalString(long), and Long.toHexString(long)). 

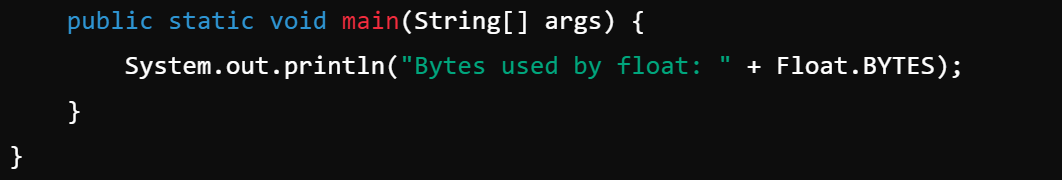
**l.** Experiment with converting a long value into other primitive types or vice versa and observe the results.

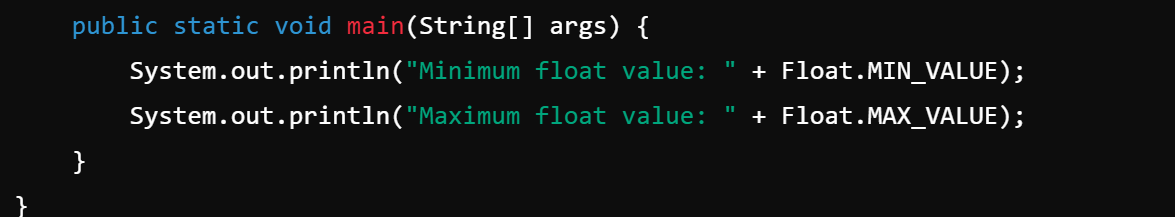


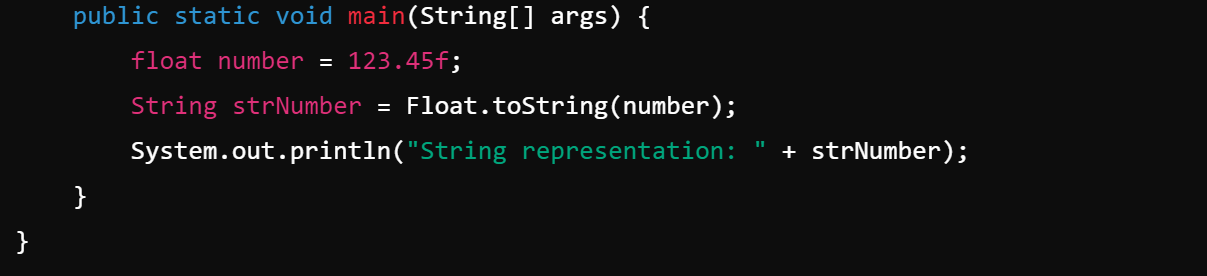
**6. Working with java.lang.Float**

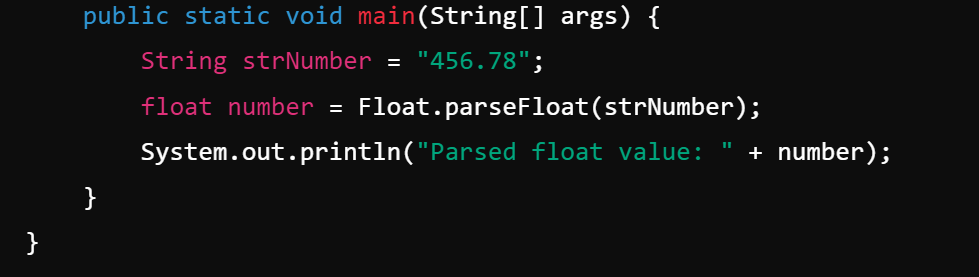
**a. Explore the Java API documentation for java.lang.Float**

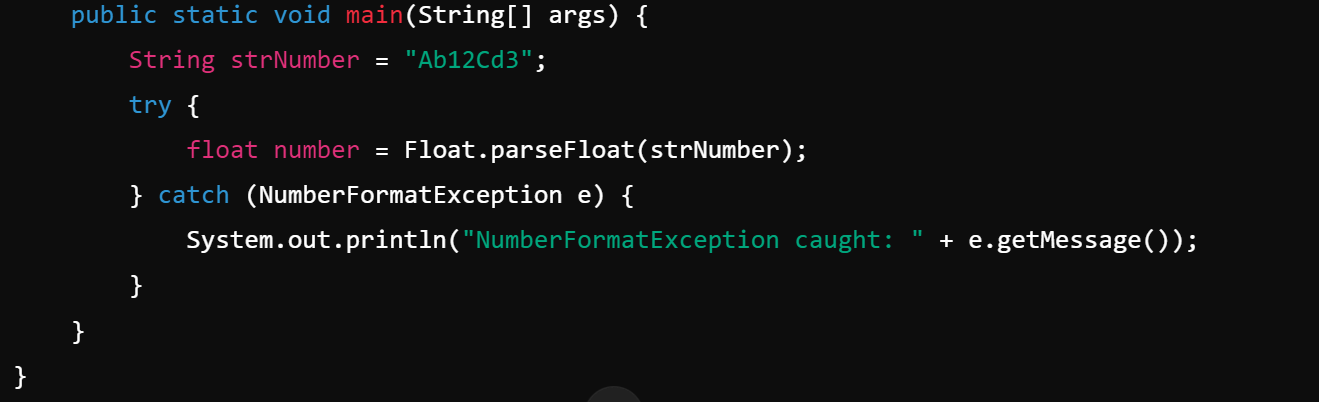
You can explore the java.lang.Float class documentation [here](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Float.html). This class is a wrapper for the primitive float type and provides methods to convert float to String and vice versa. It also contains constants like MIN\_VALUE, MAX\_VALUE, and fields like BYTES.

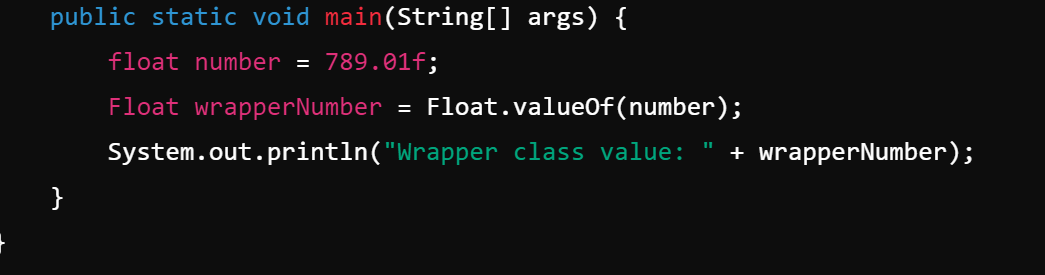
**b.** Write a program to test how many bytes are used to represent a float value using the BYTES field. (Hint: Use Float.BYTES). 

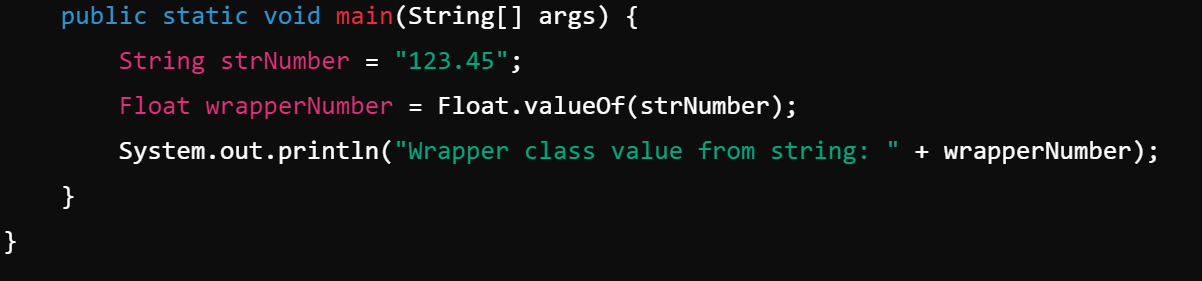
**c.** Write a program to find the minimum and maximum values of float using the MIN\_VALUE and MAX\_VALUE fields. (Hint: Use Float.MIN\_VALUE and Float.MAX\_VALUE). 

**d.** Declare a method-local variable number of type float with some value and convert it to a String using the toString method. (Hint: Use Float.toString(float)). 

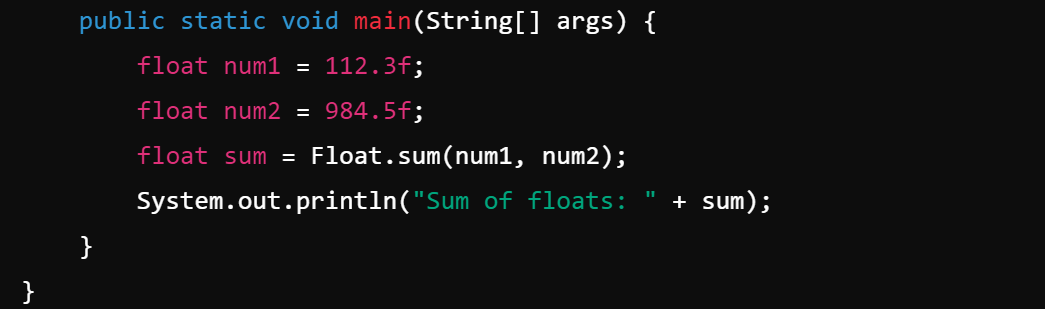
**e.** Declare a method-local variable strNumber of type String with some value and convert it to a float value using the parseFloat method. (Hint: Use Float.parseFloat(String)). 

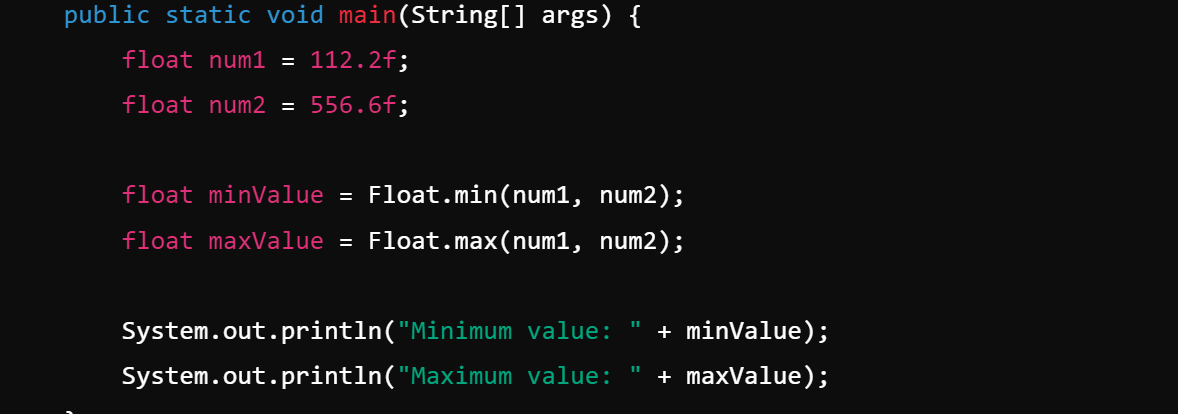
**f.** Declare a method-local variable strNumber of type String with the value "Ab12Cd3" and attempt to convert it to a float value. (Hint: parseFloat method will throw a NumberFormatException). 

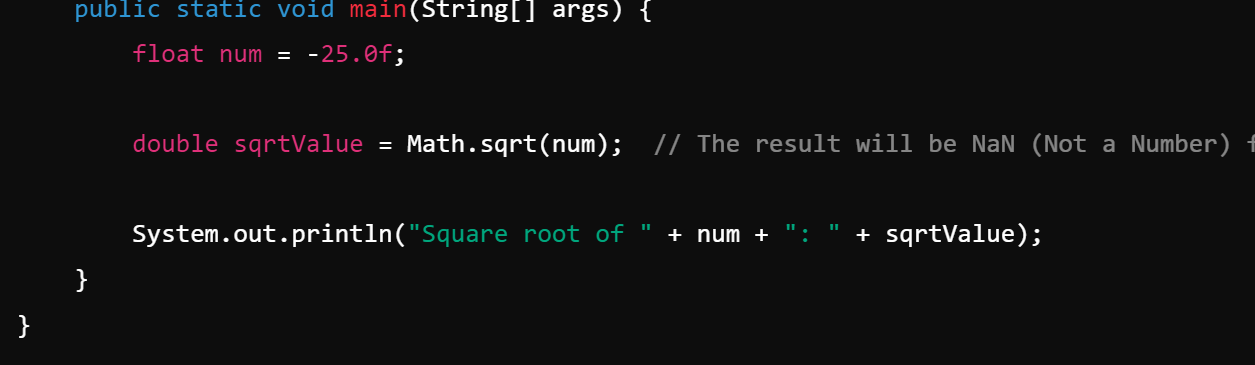
**g.** Declare a method-local variable number of type float with some value and convert it to the corresponding wrapper class using Float.valueOf(). (Hint: Use Float.valueOf(float)). 

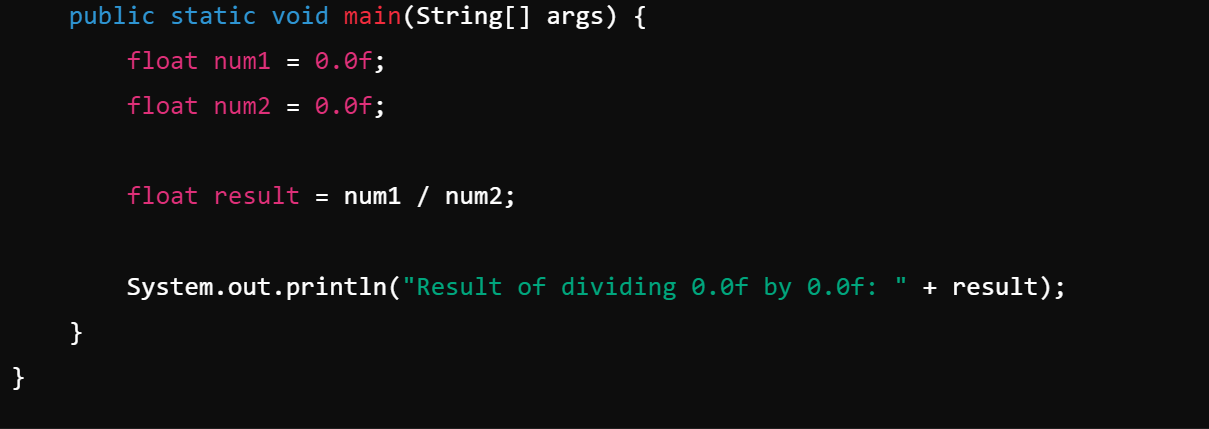
**h.** Declare a method-local variable strNumber of type String with some float value and convert it to the corresponding wrapper class using Float.valueOf(). (Hint: Use Float.valueOf(String)). 

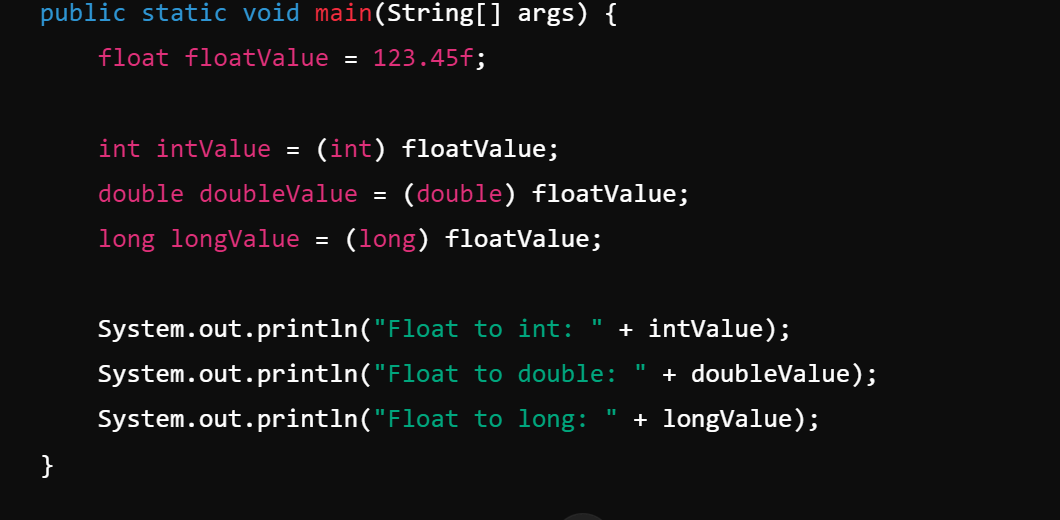
**i.** Declare two float variables with values 112.3 and 984.5, and add them using a method from the Float class. (Hint: Use Float.sum(float, float)).



**j.** Declare two float variables with values 112.2 and 556.6, and find the minimum and maximum values using the Float class. (Hint: Use Float.min(float, float) and Float.max(float, float)). 

**k.** Declare a float variable with the value -25.0f. Find the square root of this value. (Hint: Use Math.sqrt() method). 

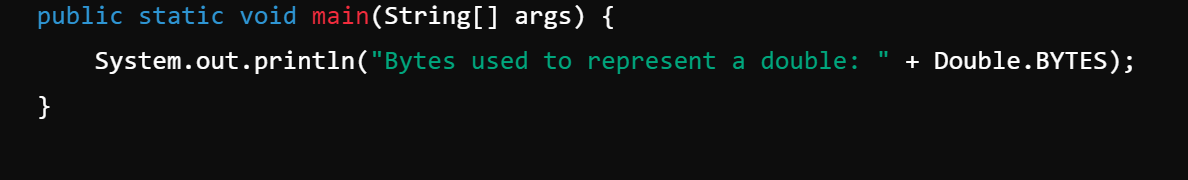
**l.** Declare two float variables with the same value, 0.0f, and divide them. (Hint: Observe the result and any special floating-point behavior). 

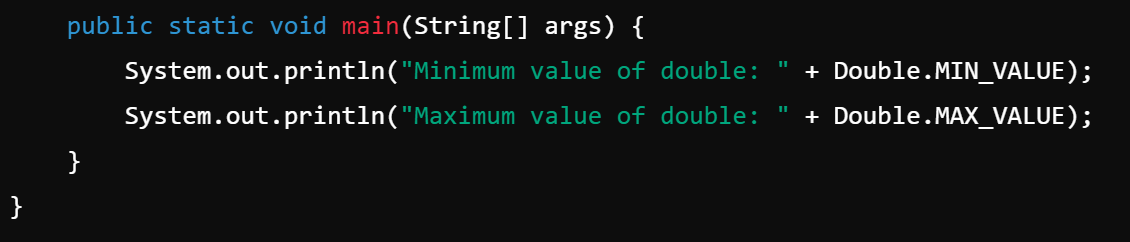
**m.** Experiment with converting a float value into other primitive types or vice versa and observe the results

**7. Working with** java.lang.Double

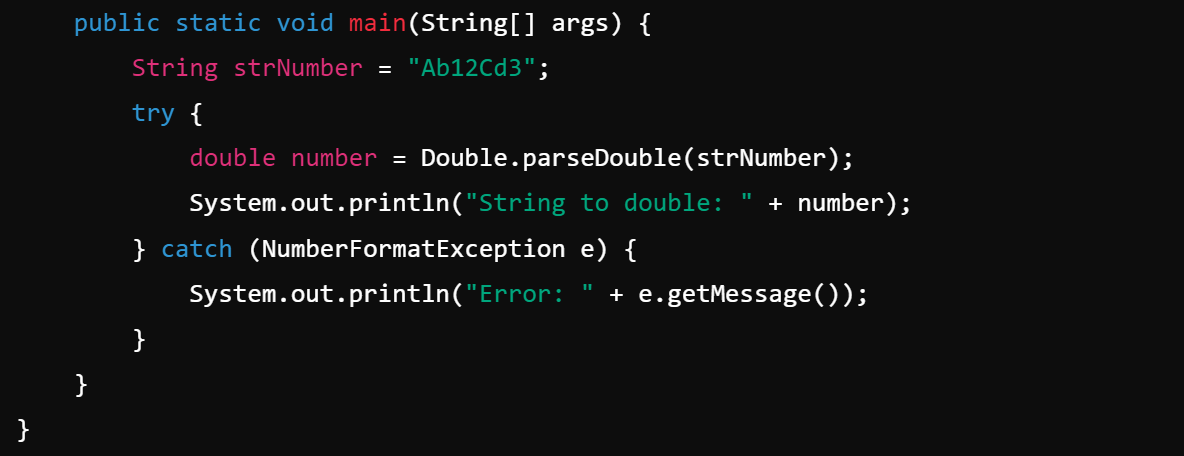
**a. Explore the Java API documentation for java.lang.Double and observe its modifiers and supertypes.**

* The Double class is a wrapper class for the primitive type double.
* Modifiers: public final class Double extends Number implements Comparable<Double>
* Supertypes: Object, Number, Comparable<Double>

**b.** Write a program to test how many bytes are used to represent a double value using the BYTES field. (Hint: Use Double.BYTES). 

**c.** Write a program to find the minimum and maximum values of double using the MIN\_VALUE and MAX\_VALUE fields. (Hint: Use Double.MIN\_VALUE and Double.MAX\_VALUE). **d.** Declare a method-local variable number of type double with some value and convert it to a String using the toString method. (Hint: Use Double.toString(double)).

**e.** Declare a method-local variable strNumber of type String with some value and convert it to a double value using the parseDouble method. (Hint: Use Double.parseDouble(String)).

**f.** Declare a method-local variable strNumber of type String with the value "Ab12Cd3" and attempt to convert it to a double value. (Hint: parseDouble method will throw a NumberFormatException). 

**g.** Declare a method-local variable number of type double with some value and convert it to the corresponding wrapper class using Double.valueOf(). (Hint: Use Double.valueOf(double)).

**public class Main {**

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

**double number = 10.5;**

**Double wn = Double.valueOf(number);**

**System.out.println("svalue (Double): " + wn);**

**}**

**}**

**h.** Declare a method-local variable strNumber of type String with some double value and convert it to the corresponding wrapper class using Double.valueOf(). (Hint: Use Double.valueOf(String)).

**public class Main {**

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

**String strNumber = "20.5";**

**Double wn = Double.valueOf(strNumber);**

**System.out.println("Wrapper class value from String (Double): " + wn);**

**}**

**}**

**i.** Declare two double variables with values 112.3 and 984.5, and add them using a method from the Double class. (Hint: Use Double.sum(double, double)).

**public class Main {**

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

**double num1 = 112.3;**

**double num2 = 984.5;**

**double wn = Double.sum(num1, num2);**

**System.out.println("Sum of num1 and num2: " + wn);**

**}**

**}**

**j.** Declare two double variables with values 112.2 and 556.6, and find the minimum and maximum values using the Double class. (Hint: Use Double.min(double, double) and Double.max(double, double)).

**public class Main {**

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

**double num1 = 112.2;**

**double num2 = 556.6;**

**double minValue = Double.min(num1, num2);**

**double maxValue = Double.max(num1, num2);**

**System.out.println("Minimum value: " + minValue);**

**System.out.println("Maximum value: " + maxValue);**

**}**

}

**k.** Declare a double variable with the value -25.0. Find the square root of this value. (Hint: Use Math.sqrt() method).

**public class Main {**

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

**double wn = -25.0;**

**double sqrtValue = Math.sqrt(wn);**

**System.out.println("Square root of -25.0: " + sqrtValue);**

**}**

}

**l.** Declare two double variables with the same value, 0.0, and divide them. (Hint: Observe the result and any special floating-point behavior).

**public class Main {**

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

**double wn1 = 0.0;**

**double wn2 = 0.0;**

**double result = wn1 / wn2;**

**System.out.println("Result of dividing 0.0 by 0.0: " + result);**

**}**

**}**

**m.** Experiment with converting a double value into other primitive types or vice versa and observe the results.

**public class Main {**

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

**double wn = 123.456;**

**int intValue = (int) wn;**

**System.out.println("double to int: " + intValue);**

**long longValue = (long) wn;**

**System.out.println("double to long: " + longValue);**

**float floatValue = (float) wn;**

**System.out.println("double to float: " + floatValue);**

**double newDoubleFromInt = intValue;**

**System.out.println("int to double: " + newDoubleFromInt);**

**double newDoubleFromLong = longValue;**

**System.out.println("long to double: " + newDoubleFromLong);**

**}**

**}**

**8. Conversion between Primitive Types and Strings**

Initialize a variable of each primitive type with a user-defined value and convert it into String:

* + First, use the toString method of the corresponding wrapper class. (e.g., Integer.toString()).
  + Then, use the valueOf method of the String class. (e.g., String.valueOf()).

**public class Main {**

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

**int iv = 10;**

**double dv = 20.5;**

**float fv = 5.75f;**

**long lv = 100L;**

**short sv = 5;**

**byte bv = 1;**

**boolean boolV = true;**

**char cv = 'A';**

**String ivToString = Integer.toString(iv);**

**String dvToString = Double.toString(dv);**

**String fvToString = Float.toString(fv);**

**String lvToString = Long.toString(lv);**

**String svToString = Short.toString(sv);**

**String bvToString = Byte.toString(bv);**

**String boolVToString = Boolean.toString(boolV);**

**String cvToString = Character.toString(cv);**

**String ivToStringVal = String.valueOf(iv);**

**String dvToStringVal = String.valueOf(dv);**

**String fvToStringVal = String.valueOf(fv);**

**String lvToStringVal = String.valueOf(lv);**

**String svToStringVal = String.valueOf(sv);**

**String bvToStringVal = String.valueOf(bv);**

**String boolVToStringVal = String.valueOf(boolV);**

**String cvToStringVal = String.valueOf(cv);**

**System.out.println("Using toString():");**

**System.out.println("int: " + ivToString + ", double: " + dvToString);**

**System.out.println("float: " + fvToString + ", long: " + lvToString);**

**System.out.println("short: " + svToString + ", byte: " + bvToString);**

**System.out.println("boolean: " + boolVToString + ", char: " + cvToString);**

**System.out.println("\nUsing String.valueOf():");**

**System.out.println("int: " + ivToStringVal + ", double: " + dvToStringVal);**

**System.out.println("float: " + fvToStringVal + ", long: " + lvToStringVal);**

**System.out.println("short: " + svToStringVal + ", byte: " + bvToStringVal);**

**System.out.println("boolean: " + boolVToStringVal + ", char: " + cvToStringVal);**

**}**

**}**

**9. Default Values of Primitive Types**

Declare variables of each primitive type as fields of a class and check their default values. (Note: Default values depend on whether the variables are instance variables or static variables).

**public class DefaultValues {**

**int intVar;**

**double doubleVar;**

**float floatVar;**

**long longVar;**

**short shortVar;**

**byte byteVar;**

**boolean boolVar;**

**char charVar;**

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

**DefaultValues obj = new DefaultValues();**

**System.out.println("Default values of primitive types:");**

**System.out.println("int: " + obj.intVar);**

**System.out.println("double: " + obj.doubleVar);**

**System.out.println("float: " + obj.floatVar);**

**System.out.println("long: " + obj.longVar);**

**System.out.println("short: " + obj.shortVar);**

**System.out.println("byte: " + obj.byteVar);**

**System.out.println("boolean: " + obj.boolVar);**

**System.out.println("char: '" + obj.charVar + "'");**

**}**

**10. Arithmetic Operations with Command Line Input**

Write a program that accepts two integers and an arithmetic operator (+, -, \*, /) from the command line. Perform the specified arithmetic operation based on the operator provided. (Hint: Use switch-case for operations).

**public class ArithmeticOperations {**

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

**if (args.length != 3) {**

**System.out.println("Usage: java ArithmeticOperations <int1> <operator> <int2>");**

**return;**

**}**

**int num1 = Integer.parseInt(args[0]);**

**String operator = args[1];**

**int num2 = Integer.parseInt(args[2]);**

**int result = 0;**

**switch (operator) {**

**case "+":**

**result = num1 + num2;**

**break;**

**case "-":**

**result = num1 - num2;**

**break;**

**case "\*":**

**result = num1 \* num2;**

**break;**

**case "/":**

**if (num2 != 0) {**

**result = num1 / num2;**

**} else {**

**System.out.println("Error: Division by zero is not allowed.");**

**return;**

**}**

**break;**

**default:**

**System.out.println("Error: Invalid operator. Use +, -, \*, or /.");**

**return;**

**}**

**System.out.println("Result: " + result);**

**}**

**}**