JAVA

▼ Java notes Beginner to OOP

Java Beginner to Intermediate Notes

1. Basic Concepts

1.1 Java Basics

- **Java**: A high-level, class-based, object-oriented programming language designed to have as few implementation dependencies as possible.
- **JDK (Java Development Kit)**: A software development kit required to develop Java applications. It includes the JRE and development tools.
- **JRE (Java Runtime Environment)**: Provides libraries, Java Virtual Machine (JVM), and other components to run Java applications.
- **JVM (Java Virtual Machine)**: An engine that provides a runtime environment to drive the Java code or applications.

1.2 Data Types

- **Primitive Data Types**: These are the most basic data types in Java:
  - `int` : Integer data type, e.g., `int number = 5;`
  - `double` : Floating-point number, e.g., `double price = 9.99;`
  - `char` : Character, e.g., `char letter = 'A';`
  - `boolean` : True or false, e.g., `boolean isJavaFun = true;`

    - Other primitives include `byte`, `short`, `long`, and `float`.
- **Non-Primitive Data Types**: These are objects, including Strings, Arrays, Classes, etc.
    - `String`: A sequence of characters, e.g., `String name = "Java";`

1.3 Variables

- **Variables**: Containers for storing data values.
    - **Declaration**: `int age;`
    - **Initialization**: `int age = 25;`

1.4 Operators

- **Arithmetic Operators**: `+`, ` `, ` `, `/`, `%`
- **Relational Operators**: `==`, `!=`, `>`, `<`, `>=`, `<=`
- **Logical Operators**: `&&` (AND), `||` (OR), `!` (NOT)
- **Increment/Decrement**: `++` (increment by 1), `-` (decrement by 1)

1.5 Control Structures

- **If-Else**: Executes code based on conditions.

```
if (condition) {
    // code
} else {
    // code
}
```

- **Switch**: Allows a variable to be tested for equality against a list of values.

```
switch(variable) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // code
}
```

- **Loops**:
    - **For Loop**: Executes a block of code a certain number of times.

```
for (int i = 0; i < 5; i++) {
    // code
}
```

    - **While Loop**: Repeats code as long as a condition is true.

```
while (condition) {
    // code
}
```

    - **Do-While Loop**: Like a while loop, but checks the condition after executing the loop’s code.

```
do {
    // code
} while (condition);
```

1.6 Arrays

- **Arrays**: Containers that hold a fixed number of values of a single type.

```
int[] numbers = {1, 2, 3, 4, 5};
```

---

2. Intermediate Concepts

2.1 Methods

- **Methods**: Blocks of code designed to perform a particular task.
    - **Declaration**: `public int addNumbers(int a, int b)`
    - **Calling**: `int sum = addNumbers(5, 10);`
    - **Return Type**: Specifies what type of data the method will return.
    - **Parameters**: Inputs passed to the method.

2.2 Classes and Objects

- **Class**: A blueprint for creating objects (a particular data structure), defining its properties and behaviors.

```
class Car {
    String color;
    int speed;

    void accelerate() {
        // code
    }
}
```

- **Object**: An instance of a class.

  ```
  Car myCar = new Car();
  ```

2.3 Constructors

- **Constructors**: Special methods used to initialize objects.

  ```
  class Car {
      Car(String color, int speed) {
          this.color = color;
          this.speed = speed;
      }
  }
  ```

2.4 Inheritance

- **Inheritance**: Mechanism where one class acquires properties (fields and methods) of another class.

  ```
  class Animal {
      void eat() {
          // code
      }
  }

  class Dog extends Animal {
      void bark() {
          // code
      }
  }
  ```

2.5 Polymorphism

- **Polymorphism**: The ability to take many forms. It allows one interface to be used for a general class of actions.
  - **Method Overloading**: Same method name with different parameters.

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

    - **Method Overriding**: A subclass provides a specific implementation of a method already defined in its superclass.

      ```
      class Animal {
          void sound() {
              // code
          }
      }

      class Dog extends Animal {
          void sound() {
              // specific code
          }
      }
      ```

2.6 Encapsulation

- **Encapsulation**: Bundling the data (variables) and code (methods) that manipulates the data into a single unit, or class. It also involves restricting access to some of the object's components.

  ```
  class Person {
      private String name;

      public String getName() {
          return name;
      }

      public void setName(String newName) {
          name = newName;
      }
  }
  ```

2.7 Abstraction

- **Abstraction**: Hiding the implementation details and showing only the essential features of the object.
    - **Abstract Class**: A class that cannot be instantiated and is designed to be subclassed.

      ```
      abstract class Animal {
          abstract void sound();
      }

      class Dog extends Animal {
          void sound() {
              // code
      ```

```
    }
}
```

  - **Interface**: A reference type in Java, it is a collection of abstract methods.

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

    class Dog implements Animal {
        public void sound() {
            // code
        }
    }
    ```

2.8 Exception Handling

- **Exception Handling**: Mechanism to handle runtime errors, so the normal flow of the application can be maintained.
  - **Try-Catch Block**: Used to catch exceptions.

    ```
    try {
        // code that may throw an exception
    } catch (ExceptionType e) {
        // code to handle the exception
    }
    ```

Object-Oriented Programming (OOP) Concepts

Java is a robust, statically typed language with a strong emphasis on OOP principles, which allow for code reuse, scalability, and maintainability.

OOP Principles Recap:

1. **Encapsulation**: Keeping the data (attributes) and the methods (functions) that manipulate the data within a single unit, or class.
2. **Inheritance**: The mechanism of basing a class on another class to reuse code.
3. **Polymorphism**: The ability of different classes to respond to the same method call in different ways.
4. **Abstraction**: Hiding the complexity and only showing the essential features of an object.

These notes should help guide you from beginner to intermediate levels, emphasizing key concepts and practical examples.

Computer Program - is a process of writing statements or commands
that instruct the computer how to process data.

PROGRAM DEVELOPMENT LIFE CYCLE
1.problem definition
2.problem analysis
3.algorithm presentation
4.coding and debugging

TYPE OF ERRORs
1.Compile Time Errors
-->syntax error and compiler detect it

2.Runtime Error
-->compiler can't catch the errors

1 Variables- temporary data during programs runtime only.

Declaring Variables > specify the type and assign it a value. (using = sign)

```
ex :    type variableName = value;

         int x = 10;

        String name = “Deng”;
```

Final Variables > this declare the variables as a FINAL or CONSTANT / not UNCHANGEABLE .

ex: final int myNum = 12;

Display Variables > println()

2 Data types

These are the primitive Data Types:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits

double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores 2 VALUES (true or false values)
char	2 bytes	Stores a single character/letter or ASCII values

Non-Primitive Data Types:

>created by the programmer and not defined by Java

>starts with Upper case   (ex: Strings, Arrays , Classes)

3 Identifiers - name of the variable that the programmer indicated. (STORAGE)

>unique names and can be short (x,y,z) and can be descriptive names (age,sum,totalPrice).

ex: int age = 12;

int y = 2;

The general rules for naming variables are:

- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names should start with a lowercase letter, and cannot contain whitespace
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case-sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like Java keywords, such as `int` or `boolean`) cannot be used as names

TYPES OF CASTING IN JAVA

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type `double` -> `float` -> `long` -> `int` -> `char` -> `short` -> `byte`
  - must be done manually by placing the type in parentheses `()` in front of the value

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

- Widening Casting (automatically) - converting a smaller type to a larger type size . `byte` -> `short` -> `char` -> `int` -> `long` -> `float` -> `double`

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Arithmetic Operators

▼ are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Java Assignment Operators

> are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5

<table>
<tr><th></th><th></th><th></th></tr>
<tr><td>+=</td><td>x += 3</td><td>x = x + 3</td></tr>
<tr><td>-=</td><td>x -= 3</td><td>x = x - 3</td></tr>
<tr><td>*=</td><td>x *= 3</td><td>x = x * 3</td></tr>
<tr><td>/=</td><td>x /= 3</td><td>x = x / 3</td></tr>
<tr><td>%=</td><td>x %= 3</td><td>x = x % 3</td></tr>
<tr><td>&=</td><td>x &= 3</td><td>x = x & 3</td></tr>
<tr><td>|=</td><td>x |= 3</td><td>x = x | 3</td></tr>
<tr><td>^=</td><td>x ^= 3</td><td>x = x ^ 3</td></tr>
<tr><td>>>=</td><td>x >>= 3</td><td>x = x >> 3</td></tr>
<tr><td><<=</td><td>x <<= 3</td><td>x = x << 3</td></tr>
</table>

Java Comparison Operators

Comparison operators are used to compare two values (or variables).

>helps us to find answers and make decisions.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Java Logical Operators

▼ are used to determine the logic between variables or values:

<table>
<tr><th>Operator</th><th>Name</th><th>Description</th><th>Example</th></tr>
<tr><td>&&</td><td>Logical and</td><td>Returns true if both statements are true</td><td>x < 5 && x < 10</td></tr>
<tr><td>||</td><td>Logical or</td><td>Returns true if one of the statements is true</td><td>x < 5 || x < 4</td></tr>
<tr><td>!</td><td>Logical not</td><td>Reverse the result, returns false if the result is true</td><td>!(x < 5 && x < 10)</td></tr>
</table>

Java Strings

Strings are used for storing text.

String METHODS:

1) `length()` method

> String in Java is a object, which contain methods that can perform certain operations on strings.

For example, the length of a string can be found with the `length()` method:

String txt = “AKOAKOAKO”;

System.out.println (“The length of txt string is :” + txt.length());

2) `toUpperCase()` and `toLowerCase()`

```
String txt = "Hello World";
System.out.println(txt.toUpperCase());   // Outputs "HELLO WORLD"
System.out.println(txt.toLowerCase());   // Outputs "hello world"
```

3) indexOf()

```
String txt = "Please locate where 'locate' occurs!";
System.out.println(txt.indexOf("locate"));
```

String Concatenation

The `+` operator can be used between strings to combine them. This is called **concatenation**:

example 1:   (using “+”)

`System.out.println(firstName + " " + lastName);`

example 2:   (using “concat()”)

`System.out.println(firstName.concat(lastName));`

Strings - Special Characters

Because strings must be written within quotes, Java will misunderstand this string, and generate an error:

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (`\`) escape character turns special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Code	Result
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed

▼ ***All String Methods***

The String class has a set of built-in methods that you can use on strings.

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
codePointAt()	Returns the Unicode of the character at the specified index	int
codePointBefore()	Returns the Unicode of the character before the specified index	int
codePointCount()	Returns the number of Unicode values found in a string.	int
compareTo()	Compares two strings lexicographically	int
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int
concat()	Appends a string to the end of another string	String
contains()	Checks whether a string contains a sequence of characters	boolean
contentEquals()	Checks whether a string contains the exact same sequence of characters of the specified CharSequence or StringBuffer	boolean
copyValueOf()	Returns a String that represents the characters of the character array	String
endsWith()	Checks whether a string ends with the specified character(s)	boolean
equals()	Compares two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
format()	Returns a formatted string using the specified locale, format string, and arguments	String
getBytes()	Converts a string into an array of bytes	byte[]
getChars()	Copies characters from a string to an array of chars	void
hashCode()	Returns the hash code of a string	int
indexOf()	Returns the position of the first found occurrence of specified characters in a string	int
intern()	Returns the canonical representation for the string object	String
isEmpty()	Checks whether a string is empty or not	boolean
join()	Joins one or more strings with a specified separator	String
lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string	int
length()	Returns the length of a specified string	int
matches()	Searches a string for a match against a regular expression, and returns the matches	boolean

offsetByCodePoints()	Returns the index within this String that is offset from the given index by codePointOffset code points	int
regionMatches()	Tests if two string regions are equal	boolean
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
replaceAll()	Replaces each substring of this string that matches the given regular expression with the given replacement	String
replaceFirst()	Replaces the first occurrence of a substring that matches the given regular expression with the given replacement	String
split()	Splits a string into an array of substrings	String[]
startsWith()	Checks whether a string starts with specified characters	boolean
subSequence()	Returns a new character sequence that is a subsequence of this sequence	CharSequence
substring()	Returns a new string which is the substring of a specified string	String
toCharArray()	Converts this string to a new character array	char[]
toLowerCase()	Converts a string to lower case letters	String
toString()	Returns the value of a String object	String
toUpperCase()	Converts a string to upper case letters	String
trim()	Removes whitespace from both ends of a string	String
valueOf()	Returns the string representation of the specified value	String

▼ **JAVA RESERVED WORDS**

Java Reserved Keywords

Java has a set of keywords that are reserved words that cannot be used as variables, methods, classes, or any other identifiers:

Keyword	Description
abstract	A non-access modifier. Used for classes and methods: An abstract class cannot be used to create objects (to access it, it must be inherited from another class). An abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from)
assert	For debugging
boolean	A data type that can only store true or false values
break	Breaks out of a loop or a switch block
byte	A data type that can store whole numbers from -128 and 127
case	Marks a block of code in switch statements
catch	Catches exceptions generated by try statements
char	A data type that is used to store a single character
class	Defines a class

continue	Continues to the next iteration of a loop
const	Defines a constant. Not in use - use final instead
default	Specifies the default block of code in a switch statement
do	Used together with while to create a do-while loop
double	A data type that can store fractional numbers from 1.7e−308 to 1.7e+308
else	Used in conditional statements
enum	Declares an enumerated (unchangeable) type
exports	Exports a package with a module. New in Java 9
extends	Extends a class (indicates that a class is inherited from another class)
final	A non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override)
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
float	A data type that can store fractional numbers from 3.4e−038 to 3.4e+038
for	Create a for loop
goto	Not in use, and has no function
if	Makes a conditional statement
implements	Implements an interface
import	Used to import a package, class or interface
instanceof	Checks whether an object is an instance of a specific class or an interface
int	A data type that can store whole numbers from -2147483648 to 2147483647
interface	Used to declare a special type of class that only contains abstract methods
long	A data type that can store whole numbers from -9223372036854775808 to 9223372036854775808
module	Declares a module. New in Java 9
native	Specifies that a method is not implemented in the same Java source file (but in another language)
new	Creates new objects
package	Declares a package
private	An access modifier used for attributes, methods and constructors, making them only accessible within the declared class
protected	An access modifier used for attributes, methods and constructors, making them accessible in the same package and subclasses
public	An access modifier used for classes, attributes, methods and constructors, making them accessible by any other class
requires	Specifies required libraries inside a module. New in Java 9
return	Finished the execution of a method, and can be used to return a value from a method
short	A data type that can store whole numbers from -32768 to 32767
static	A non-access modifier used for methods and attributes. Static methods/attributes can be accessed without creating an object of a class
strictfp	Obsolete. Restrict the precision and rounding of floating point calculations

super	Refers to superclass (parent) objects
switch	Selects one of many code blocks to be executed
synchronized	A non-access modifier, which specifies that methods can only be accessed by one thread at a time
this	Refers to the current object in a method or constructor
throw	Creates a custom error
throws	Indicates what exceptions may be thrown by a method
transient	Used to ignore an attribute when serializing an object
try	Creates a try...catch statement
var	Declares a variable. New in Java 10
void	Specifies that a method should not have a return value
volatile	Indicates that an attribute is not cached thread-locally, and is always read from the "main memory"
while	Creates a while loop

▼ ***ALL MATH METHODS***

Method	Description	Return Type
abs(x)	Returns the absolute value of x	double|float|int|long
acos(x)	Returns the arccosine of x, in radians	double
addExact(x, y)	Returns the sum of x and y	int|long
asin(x)	Returns the arcsine of x, in radians	double
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians	double
atan2(y,x)	Returns the angle theta from the conversion of rectangular coordinates (x, y) to polar coordinates (r, theta).	double
cbrt(x)	Returns the cube root of x	double
ceil(x)	Returns the value of x rounded up to its nearest integer	double
copySign(x, y)	Returns the first floating point x with the sign of the second floating point y	double|float
cos(x)	Returns the cosine of x (x is in radians)	double
cosh(x)	Returns the hyperbolic cosine of a double value	double
decrementExact(x)	Returns x-1	int|long
exp(x)	Returns the value of Ex	double
expm1(x)	Returns ex -1	double
floor(x)	Returns the value of x rounded down to its nearest integer	double
floorDiv(x, y)	Returns the division between x and y rounded down	int|long
floorMod(x, y)	Returns the remainder of a division between x and y where the result of the division was rounded down	int|long
getExponent(x)	Returns the unbiased exponent used in x	int
hypot(x, y)	Returns sqrt(x2 +y2) without intermediate overflow or underflow	double
IEEEremainder(x, y)	Computes the remainder operation on x and y as prescribed by the IEEE 754 standard	double

incrementExact(x)	Returns x+1	int|double
log(x)	Returns the natural logarithm (base E) of x	double
log10(x)	Returns the base 10 logarithm of x	double
log1p(x)	Returns the natural logarithm (base E) of the sum of x and 1	double
max(x, y)	Returns the number with the highest value	double|float|int|long
min(x, y)	Returns the number with the lowest value	double|float|int|long
multiplyExact(x, y)	Returns the result of x multiplied with y	int|long
negateExact(x)	Returns the negation of x	int|long
nextAfter(x, y)	Returns the floating point number adjacent to x in the direction of y	double|float
nextDown(x)	Returns the floating point value adjacent to x in the negative direction	double|float
nextUp(x)	Returns the floating point value adjacent to x in the direction of positive infinity	double|float
pow(x, y)	Returns the value of x to the power of y	double
random()	Returns a random number between 0 and 1	double
rint(x)	Returns the double value that is closest to x and equal to a mathematical integer	double
round(x)	Returns the value of x rounded to its nearest integer	long|int
scalb(x, y)	Returns x multiplied by 2 to the power of y	double|float
signum(x)	Returns the sign of x	double|float
sin(x)	Returns the sine of x (x is in radians)	double
sinh(x)	Returns the hyperbolic sine of a double value	double
sqrt(x)	Returns the square root of x	double
subtractExact(x, y)	Returns the result of x minus y	int|long
tan(x)	Returns the tangent of an angle	double
tanh(x)	Returns the hyperbolic tangent of a double value	double
toDegrees(x)	Converts an angle measured in radians to an approx. equivalent angle measured in degrees	double
toIntExact(x)	Converts a long value to an int	int
toRadians(x)	Converts an angle measured in degrees to an approx. angle measured in radians	double
ulp(x)	Returns the size of the unit of least precision (ulp) of x	double|float

▼ **Java Output Methods**

Output Methods

The `System.out` stream, short for "**output**", is used together with different methods to output values or print text to the console:

Method	Description
print()	Prints text or values the console
printf()	Prints formatted text or values to the console

println()	Prints text or values to the console, followed by a new line

Java Math Methods

The Java Math class has many methods that allows you to perform mathematical tasks on numbers.

Note: All Math methods are `static`.

SCANNER

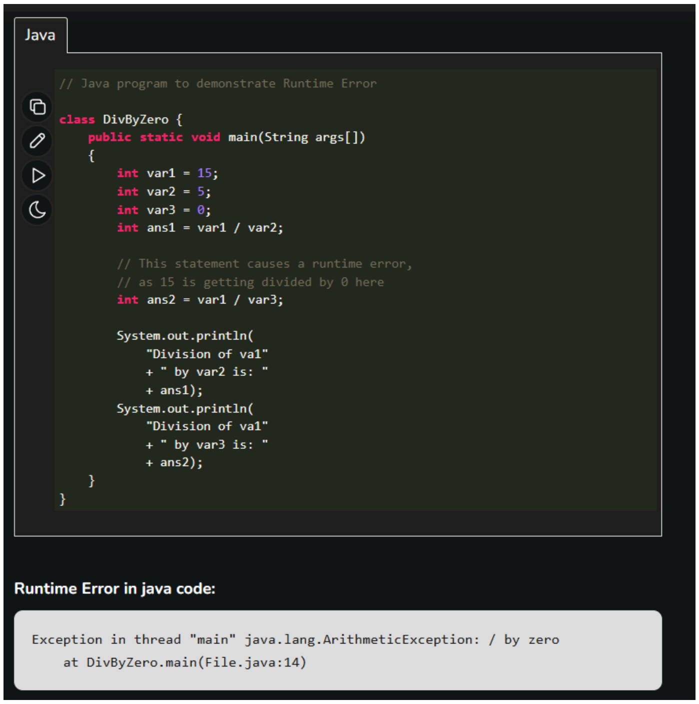
> USER INPUT :

- nextLine()
- nextInt()
- nextShort()
- nextLong()
- nextByte()
- nextBoolean()
- nextDouble()
- nextFloat()

REVIEWER JAVA!!!

TYPES OF ERRORS IN JAVA

1. Run time error - detected during the execution of the program.

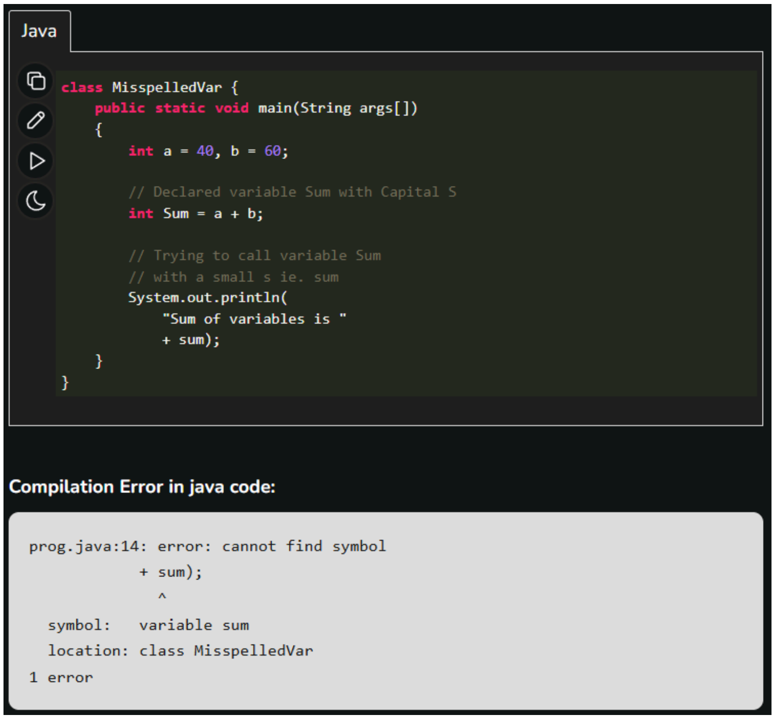


```java
// Java program to demonstrate Runtime Error

class DivByZero {
    public static void main(String args[])
    {
        int var1 = 15;
        int var2 = 5;
        int var3 = 0;
        int ans1 = var1 / var2;

        // This statement causes a runtime error,
        // as 15 is getting divided by 0 here
        int ans2 = var1 / var3;

        System.out.println(
            "Division of va1"
            + " by var2 is: "
            + ans1);
        System.out.println(
            "Division of va1"
            + " by var3 is: "
            + ans2);
    }
}
```

Runtime Error in java code:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at DivByZero.main(File.java:14)
```

2. Compile Time Error - prevent the code from running because of incorrect syntax such as grammatical errors, misspelling variables or names , missing semi-colon; {} () [] / Also called Syntax Error.

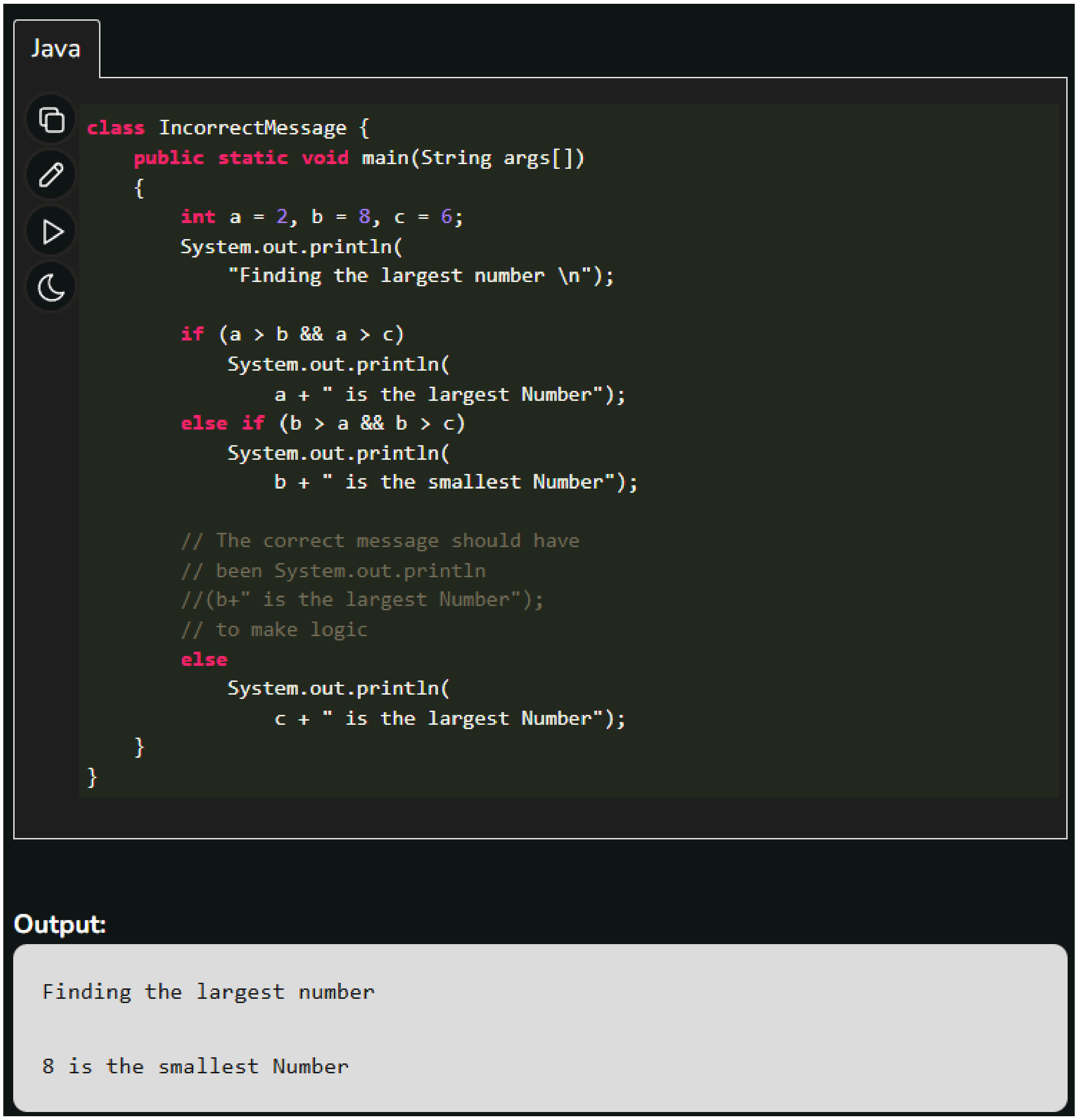


```java
class MisspelledVar {
    public static void main(String args[])
    {
        int a = 40, b = 60;

        // Declared variable Sum with Capital S
        int Sum = a + b;

        // Trying to call variable Sum
        // with a small s ie. sum
        System.out.println(
            "Sum of variables is "
            + sum);
    }
}
```

Compilation Error in java code:

```
prog.java:14: error: cannot find symbol
            + sum);
              ^
  symbol:   variable sum
  location: class MisspelledVar
1 error
```

3. Logical Error - when your program works, but does the wrong thing or returns an incorrect result or no output when it should be returning an output . It is caused due to an incorrect idea or concept used by the programmer / Also called Semantic Error .



```java
class IncorrectMessage {
    public static void main(String args[])
    {
        int a = 2, b = 8, c = 6;
        System.out.println(
            "Finding the largest number \n");

        if (a > b && a > c)
            System.out.println(
                a + " is the largest Number");
        else if (b > a && b > c)
            System.out.println(
                b + " is the smallest Number");

        // The correct message should have
        // been System.out.println
        //(b+" is the largest Number");
        // to make logic
        else
            System.out.println(
                c + " is the largest Number");
    }
}
```

Output:

```
Finding the largest number

8 is the smallest Number
```