****

Java Basic Concept

****

Java Programming Language:

Java is a versatile programming language used to create software applications. It's popular because it's platform-independent, meaning it runs on any device. Java code is compiled into bytecode, which can run on any system with a Java Virtual Machine (JVM). It's commonly used for web development, mobile apps, and enterprise software.

Java is developed by James Gosling with his team at Sun Microsystems in the year of 1991.
The first name given by James Gosling was OAK. First time officially announced Java 1995 by Sun Microsystem. In the year 2009, Oracle Corporation had takeover from Sun Microsystem.

Java Versions:

Java versions refer to different releases of the Java programming language and its associated development kit, the JDK (Java Development Kit).
Each major release typically includes updates to the language syntax, libraries, and performance improvements. Additionally, newer versions may deprecate or remove outdated features and APIs.

In addition to major releases, Java also has minor and interim updates. Minor updates, denoted by a decimal point (e.g., Java 8u271), mainly include bug fixes and security patches.

In new version, new features introduce, enhancements and also bug fixes. Java follows a versioning scheme where major releases are numbered, such as Java 5, Java 8, Java 11, Java 15, Java 17, Java 22 etc.

Java's Editions:

Each edition of Java has different capabilities.
There are three editions of Java:
JSE means Java Standard Editions. It's a platform for developing and running Java applications on desktops and servers. It provides core libraries and tools essential for Java development, making it a foundation for many software projects. We are creating application for a desktop computer.

JEE stands for Java Enterprise Edition. It's a platform for developing large-scale, distributed enterprise applications in Java. It provides APIs and runtime environments for building web services, enterprise-level applications, and scalable solutions for businesses.

JME stands for Java Micro Edition. It's a platform for developing applications for small devices like mobile phones and embedded systems. It provides a scaled-down version of Java, optimized for resource-constrained environments.

Java Applications:

We can make four kinds of Java application with Java programming:
Java standalone applications are programs that run independently on a computer without needing a web browser or internet connection. They are self-contained and typically interact with the user through a graphical user interface (GUI) or command-line interface (CLI).

Enterprise Applications:
Enterprise applications are software programs designed to support large-scale business processes and operations within an organization.
They often encompass a wide range of functionalities such as customer relationship management (CRM), enterprise resource planning (ERP), supply chain management (SCM), and more.
These applications are typically complex, highly scalable, and may involve multiple interconnected modules or components. They are crucial for managing various aspects of business operations efficiently and effectively.

Web Applications:
Java web applications are software programs that run on web servers and are accessed through a web browser over the internet.
They are designed to provide interactive content or services to users, such as online banking systems, social media platforms, e-commerce websites, and more. Java web applications use technologies like servlets, Java Server Pages (JSP), and frameworks like Spring MVC to handle user requests, process data, and generate dynamic web pages.

Mobile Applications:
Java mobile applications are software programs designed to run on mobile devices like smartphones and tablets. They offer various functionalities such as games, productivity tools, communication apps, and more. These applications are typically developed using Java ME (Micro Edition) or other mobile development frameworks and are distributed through app stores or directly downloaded onto the mobile device.

Java Main Features

Java is renowned for its powerful features that make it a popular choice among developers. Some key features:
Platform Independence:
Java application can run on any device platform(windows/Linux) with help of Java Virtual Machine (JVM. This "write once, run anywhere" capability allows developers to create software that can run on diverse platforms without modification.

Object-Oriented:
Java is object-oriented, meaning it revolves around objects that encapsulate data and behaviour. This paradigm promotes code reusability, modularity, and easier maintenance.

Simple and Familiar Syntax:
Java's syntax is derived from C and C++, making it familiar to developers of these languages. It's designed to be easy to learn and read, with clear and straightforward syntax.

Automatic Memory Management (Garbage Collection):
Java automatically manages memory allocation and deallocation, relieving developers from manual memory management tasks. The Garbage Collector identifies and removes unused objects, reducing memory leaks and enhancing application stability.

Robustness and Security:
Java's robustness is evident in its built-in exception handling, strong type checking, and runtime environment that ensures reliable program execution. Java's security features include a robust security model, class loading verification, and sandboxing to prevent unauthorized access and execution of malicious code.

Multi-threading Support:
Java supports concurrent programming through its multi-threading capabilities. Developers can create applications that execute multiple threads concurrently, enhancing performance and responsiveness.

Rich Standard Library:
Java comes with a vast standard library (Java API) that provides ready-to-use classes and methods for common programming tasks, such as networking, I/O operations, data structures, and more. This extensive library reduces development time and effort.

High Performance:
Despite being interpreted initially, modern Java implementations, such as Oracle's Hotspot JVM, offer high performance through Just-In-Time (JIT) compilation, adaptive optimization, and other techniques.

Portability:
Java's platform independence and bytecode compilation enable portability across different operating systems and hardware architectures, allowing applications to run unchanged on various devices and environments.

Community and Ecosystem:
Java boasts a large and active developer community, along with a vibrant ecosystem of tools, frameworks (e.g., Spring, Hibernate), and libraries (e.g., Apache Commons, Guava) that extend its capabilities and support various development needs

Java Is Well Known For OOPs:

Java embraces the principles of Object-Oriented Programming (OOP), facilitating modular, scalable, and maintainable codebases. The Key OOP concepts in Java include:

Class and Object:
Classes serve as blueprints defining data members and methods, while objects represent instances of these classes, embodying unique states and behaviours.

Encapsulation:
Bundling data and methods within a single unit, encapsulation fosters data security and code modularity, crucial for building robust applications.
Encapsulation involves bundling data (attributes) and methods (behaviour) within a single unit, known as a class. This shields the internal state of an object from external interference and ensures data integrity. For instance, consider a "Car" class with attributes like "model," "colour," and methods like "startEngine()" and "stopEngine()".

Inheritance:
Enabling code reuse and hierarchy, inheritance allows classes to acquire properties and behaviours of parent classes, promoting code scalability and extensibility.
It allows a sub class to inherit attributes and methods from super class. It promotes code reusability and initiates an "is-a" relationship between classes.
For example, a "Vehicle" class can serve as a superclass for "Car," "Truck," and "Motorcycle" classes, inheriting common attributes and methods.

Polymorphism:
Facilitating flexibility and extensibility, enhancing code adaptability and versatility across various contexts.
Polymorphism enables objects to take on multiple forms or behaviours based on their context.
In Java, it can be achieved through method overriding (re-defining parent class method in the child class) and method overloading (with in same class writing multiple method having same name but different number of parameters or their different data types).
For instance, a "Shape" superclass can have a "draw()" method overridden by subclasses like "Circle" and "Rectangle," each implementing its drawing logic.

Abstraction:
Concealing implementation details, involves hiding complex implementation details and exposing only essential features of an object, enhancing code clarity and maintainability.
In Java, it can be achieved through abstract classes and interfaces.
Abstract classes define incomplete methods that must be implemented by subclasses, while interfaces specify a contract that implementing classes must adhere to.
For example, an "Animal" abstract class can define abstract methods like "eat()" and "sleep()," which concrete subclasses like "Dog" and "Cat" must implement eat() and sleep().

Java Data Types

Java offers various data types tailored to different use cases, each with its own space occupancy and suitability for specific scenarios. Let's explore some common Java data types along with their uses and examples:

Primitive Data Types:

byte:
Used for small integer values within the range of -128 to 127. Occupies 1 byte of memory. Ideal for conserving memory in large arrays or when dealing with raw binary data.
byte num = 10;

short:
Suitable for small integer values ranging from -32,768 to 32,767. Occupies 2 bytes of memory. Often used in scenarios where memory conservation is essential but a wider range than byte is required.
short num = 1000;

int:
One of the most commonly used data types, representing integer values from -2^31 to 2^31 - 1. Occupies 4 bytes of memory. Ideal for general-purpose integer arithmetic.
int num = 10000;

long:
Used for large integer values within the range of -2^63 to 2^63 - 1. Occupies 8 bytes of memory. Suitable for scenarios requiring a wider range or higher precision than int.
long num = 10000000000L;

float:
Represents single-precision floating-point numbers, suitable for decimal values with moderate precision. Occupies 4 bytes of memory. Ideal for scientific calculations or situations where memory usage needs to be minimized.
float num = 3.14f;

double:
Offers double-precision floating-point numbers, providing greater precision compared to float. Occupies 8 bytes of memory. Commonly used for financial calculations, where precision is crucial.
double num = 3.14159265359;

boolean:
Represents a boolean value, either true or false. Occupies 1 bit of memory but typically implemented as 1 byte. Widely used in conditional statements and control flow.
boolean flag = true;

char:
Used to store a single character, such as letters, digits, or symbols, represented in Unicode format. Occupies 2 bytes of memory. Essential for handling textual data and character manipulation.
char letter = 'A';

Reference/Non-primitive Data Types:

String:
Represents a sequence of characters, widely used for storing textual data. Strings are immutable objects in Java. They occupy memory based on the length of the string and the characters it contains.
String greeting = "Hello, World!";

Arrays:
Used to store a collection of elements of the same data type. Arrays in Java are reference types, and their memory occupancy depends on the number of elements and the size of each element.
int[] rollNumbers = {11, 12, 13, 14, 15};

Classes and Objects:
Java allows the creation of custom classes and objects, enabling developers to model real-world entities and encapsulate data and behavior. Memory occupancy for classes and objects depends on their size and the data they encapsulate.

```
class Person {
   String name;
   int age;
}
Person person1 = new Person();
```

Selecting the appropriate data type in Java is crucial for optimizing memory usage and ensuring the efficient execution of programs. Understanding the characteristics and use cases of different data types enables developers to make informed decisions when designing and implementing Java applications.

Keywords In Java

Reserved words or keywords in Java are predefined terms that have specific meanings and purposes within the language. They are reserved for special use and cannot be used as identifiers such as class, interface, method, or variable names. The importance of reserved words lies in maintaining the consistency and integrity of the Java language syntax and ensuring that code is clear, readable, and unambiguous.

We should avoid using keywords as identifiers because of following reasons:

Clarity and Readability:
Using keywords as identifiers can lead to confusion and make code harder to understand for other developers who are familiar with the standard Java conventions.

Compiler Errors:
If you attempt to use a keyword as an identifier, the Java compiler will generate an error. This is because keywords have predefined meanings and cannot be redefined as identifiers.

Maintainability:
Avoiding the use of keywords ensures that your code remains maintainable. If keywords were used as identifiers, it could be difficult to distinguish between language constructs and user-defined entities.

Future Compatibility:
Java may introduce new keywords in future versions of the language. By avoiding the use of keywords as identifiers, you prevent potential conflicts with new language features.

Compiler generates error:
If you were to use a keyword as an identifier, the Java compiler would generate a syntax error, indicating that a keyword cannot be used in that context.
For example, try to name a variable "class" or "int", you would encounter a compilation error:

```
int class = 10; // Error: 'class' is a keyword and cannot be used as an identifier
```

Keywords possible categorization:
Understanding these categories and their respective keywords is crucial for mastering Java programming. It allows developers to utilize the language effectively, write clear and concise code, and leverage Java's powerful features for building robust and scalable applications.

Java keywords can be categorized into several groups based on their functionality.
Let's categorize them and explain each group:

Primitive Types Keywords:
boolean: Represents a boolean value, either true or false.
byte: It represents an 8-bit signed integer.
short: It represents a 16-bit signed integer.
int: It represents a 32-bit signed integer.
long: It represents a 64-bit signed integer.
float: It represents a single-precision 32-bit IEEE 754 floating point.
double: It represents a double-precision 64-bit IEEE 754 floating point.
char: It represents a 16-bit Unicode character.

Flow Control Keywords:
if, else: Used for conditional branching.
switch, case, default: Used for multi-branch selection.
for, while, do: Used for looping.
break, continue: Used for altering control flow within loops.
return: Used to exit a method and return a value.
try, catch, finally: Used for exception handling.
throw, throws: Used to explicitly throw or declare exceptions.

Class, Method, and Package Keywords:
class: Declares a class.
interface: Declares an interface.
extends: Indicates inheritance.
implements: Indicates interface implementation.
package: Declares a package.
import: Imports classes, interfaces, and packages.
this: Refers to the current object.
super: Refers to the superclass.
static: Indicates a class-level entity.
final: Indicates a constant value, or that a variable, method, or class cannot be overridden or subclassed.

Visibility Keywords:
private, default, protected, public : Specifies the visibility of classes, methods, and variables.

Object Creation and Management Keywords:
new: It creates new objects.
instanceof: It checks if an object is an instance of a class.
null: It represents the absence of a value.

Miscellaneous Keywords:
void: It indicates that a method does not return a value.
true, false: Literal values representing boolean states.
enum: Declares an enumerated type.
synchronized: Specifies that a method can only be accessed by one thread at a time.
volatile: It indicates that a variable may be modified asynchronously.
transient: It indicates that a variable should not be serialized.
assert: Asserts a condition to be true.

Variable & Constants

Variables and constants are fundamental concepts in programming languages like Java, but they serve different purposes.

Variables:
Definition: Variables are named storage locations in a program's memory, used to hold data that can change during the program's execution.

Variables Declaration in Java
We can declare variables in Java as pictorially depicted below as a visual aid. Variables are declared using a specific data type and an identifier (name) that uniquely identifies the storage location.

Mutable:
Variables can be assigned new values multiple times during the program's execution.

Initializing Variables in Java
It can be perceived with the help of 3 components that are as follows:
datatype: Type of data that can be stored in this variable.
variable_name: Name given to the variable.
value: It is the initial value stored in the variable.

Example:
In Java, a variable named "count" of type integer (int) can be declared and assigned a value:
int count = 5;

Scenario for Use:
Variables are used when the value stored may change over time, such as user input, intermediate results in calculations, or values retrieved from external sources like databases.

Constants:
Definition: Constants are named values that remain unchanged throughout the program's execution.

Immutable: Once defined, the value of a constant cannot be modified during runtime.

Declaration: Constants are typically declared using the "final" keyword in Java, indicating that their value is fixed and cannot be altered.

Example: In Java, a constant named "PI" representing the mathematical constant π can be declared:
final double PI = 3.14159;

Scenario for Use: Constants are used when a value is known and will not change during the program's execution, such as mathematical constants, configuration settings, or fixed values used throughout the program.

Differentiation between Variables & Constants:

Use variables when dealing with data that may change during program execution, such as user input, intermediate calculations, or dynamic data.
Use constants for values that are known and will not change during program execution, such as mathematical constants, configuration settings, or fixed values used throughout the program.

Choosing between variables and constants depends on whether the value is expected to change or remain constant during the program's execution. Properly selecting and managing variables and constants contribute to code clarity, maintainability, and reliability.

Types of Variables based on availabilities in Java

Different types of variables which are listed as follows:

Local Variables:
A variable defined within a block or method or constructor is called a local variable.

These variables are created when the block is entered, or the function is called and destroyed after exiting from the block or when the call returns from the function.

The scope of these variables exists only within the block in which the variables are declared, i.e., we can access these variables only within that block.

**Initialization of the local variable is mandatory before using it in the defined scope.*

Instance Variables:
Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.

*Instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.

*Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier, then the default access specifier will be used.

**Initialization of an instance variable is not mandatory. Its default value is dependent on the data type of variable.*
String/reference is null,
float is 0.0f,
int is 0,
long is 0l,
boolean is false

Wrapper classes (Integer, Float, Double etc) is null
*Instance variables can be accessed only by creating objects.
*Instance variable can be initializing constructors.
*May also use instance blocks.
*Can be modify/accessed using setters/getters

Static Variables:
Static variables are also known as class variables.
*They are declared similarly to instance variables.
*Static variables are declared using the static keyword.
*They are declared in a class but outside of any method, constructor, or block.
*Unlike instance variables, we can only have one copy of a static variable per class, irrespective of n many objects we create.
*Static variables are created at the start of program execution and destroyed automatically when execution ends.
*Initialization of a static variable is not mandatory.
*Its default value is dependent on the data type of variable. String is null,
float is 0.0f,
int it is 0,
Wrapper classes like Integer it is null etc.

If we access a static variable like an instance variable (through an object), the compiler will show a warning message, which won't halt the program. The compiler will replace the object name with the class name automatically.

If we access a static variable without the class name, the compiler will automatically append the class name. But for accessing the static variable of a different class, we must mention the class name as 2 different classes might have a static variable with the same name.

Static variables cannot be declared locally inside an instance method.

Static blocks can be used to initialize static variables.

Differences Between the instance and static Variables:
Each object will have its own copy of an instance variable, whereas we can only have one copy of a static variable per class, irrespective of how many objects we create. Thus, static variables are good for memory management.

Changes made in an instance variable using one object will not be reflected in other objects as each object has its own copy of the instance variable.
In the case of a static variable, changes will be reflected in other objects as static variables are common to all objects of a class.

We can access instance variables through object references, and static variables can be accessed directly using the class name.

Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. Static variables are created when the program starts and destroyed when the program stops.

Scope of Variables in Java:

Modifier	Package	Subclass	Other World
public	Yes	Yes	Yes
protected	Yes	Yes	No
Default (no modifier)	Yes	No	No
private	No	No	No

Wrapper Classes in Java:

A Wrapper class in Java is a class whose object wraps or contains primitive data types.

When we create an object to a wrapper class, it contains a field and, in this field, we can store primitive data types.

In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes:
There are certain needs for using the Wrapper class in Java as mentioned below:

They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).

The classes in java.util package handles only objects and hence wrapper classes help in this case also.

Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
An object is needed to support synchronization in multithreading.

Advantages of Wrapper Classes:
Collections allowed only object data.
On object data we can call multiple methods compareTo(), equals(), toString()
Cloning process only objects
Object data allowed null values.
Serialization can allow only object data.

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Autoboxing and Unboxing
Autoboxing:
The automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing.
For example – conversion of int to Integer, long to Long, double to Double, etc.

Unboxing:
It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double, etc

Why use the wrapper class in Java?
The wrapper class in Java is used to convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method.

Identifiers

In Java, an identifier is a name given to various program elements such as classes, variables, methods, and packages. These names serve as labels to uniquely identify these elements within the program's scope. Identifiers follow specific rules and naming conventions in Java.

Significance of Naming Conventions:

Readability and Understandability:
Consistent and meaningful naming conventions enhance code readability and understandability. By adhering to naming conventions, developers can easily comprehend the purpose and functionality of different program elements without extensive documentation.

Maintainability:
Well-chosen identifiers contribute to code maintainability by making it easier for developers to navigate and modify code. Descriptive and intuitive names reduce the likelihood of confusion and errors when revisiting or updating code.

Collaboration and Teamwork:
Naming conventions facilitate collaboration among team members by establishing a common language and style for naming program elements. Consistency in naming conventions across the project ensures uniformity and reduces the learning curve for new team members.

Avoidance of Conflicts:
Following naming conventions helps avoid naming conflicts and ambiguities within the codebase. Clear and distinctive identifiers reduce the risk of unintended variable shadowing or method overloading, promoting code robustness and reliability.

Naming Conventions in Java:

CamelCase:
CamelCase is widely used for naming variables, methods, and parameters in Java. It involves starting the first word with a lowercase letter and capitalizing the first letter of subsequent words, without using spaces or punctuation. For example: firstName, calculateTotalPrice().

PascalCase:
PascalCase is commonly used for naming classes and interfaces in Java. It involves capitalizing the first letter of each word, including the first word, without using spaces or punctuation. For example: CustomerService, UserInterface.

UPPER_CASE:
UPPER_CASE is used for naming constants in Java. It involves using uppercase letters with underscores (_) to separate words. For example: MAX_SIZE, DEFAULT_TIMEOUT.

Package Naming:
Package names are typically lowercase, with multiple words separated by periods (.) to indicate hierarchy. For example: com.example.project.

Abbreviations:
Abbreviations should be avoided or used sparingly in identifiers to maintain clarity and readability. If used, they should be consistently applied and widely understood within the context of the project.

Operators

In Java, an operator is a symbol that performs specific operations on one or more operands to produce a result. Operators are essential for performing mathematical, logical, and relational computations in Java programs. Java supports various types of operators, each serving a distinct purpose.

Arithmetic Operators:
It performs basic mathematical operations such as addition, subtraction, multiplication, division, and modulus.
Example:

```
int a = 10;
int b = 5;
int sum = a + b; // Doing addition
int diff = a - b; // Doing subtraction
int prod = a * b; // Doing multiplication
int quotient = a / b; // Doing division
int remainder = a % b; // Doing modulus (remainder after division)
```

Assignment Operators:
It used to assign values to variables.
Example:

```
int x = 10;
int y = 5;
x += y; // Equivalent to: x = x + y;
```

Comparison Operators:
It compares two values and return a boolean result indicating whether the comparison is true or false.
Example:

```
int a = 10;
int b = 5;
boolean isEqual = (a == b); // Equality comparison
boolean isGreaterThan = (a > b); // Greater than comparison
boolean isLessThanOrEqual = (a <= b); // Less than or equal to comparison
```

Logical Operators:
It perform logical operations on boolean values and produce a boolean result.
Example:

```
boolean x = true;
boolean y = false;
boolean result = x && y; // Logical AND
boolean result2 = x || y; // Logical OR
boolean result3 = !x; // Logical NOT
```

Increment and Decrement Operators:
Increment (++) and decrement (--) operators are used to increase or decrease the value of a variable by
Example:

```
int count = 0;
count++; // Increment count by 1
count--; // Decrement count by 1
```

Bitwise Operators:
Bitwise operators perform bitwise operations on integer operands at the binary level.
Example:

```
int a = 5; // Binary: 0101
int b = 3; // Binary: 0011
int result = a & b; // Bitwise AND (result: 0001)
int result2 = a | b; // Bitwise OR (result: 0111)
```

Conditional Operator (Ternary Operator):
The conditional operator (? :) is a ternary operator that evaluates a boolean expression and returns one of two possible values based on the result.
Example:

```
int a = 10;
int b = 5;
int result = (a > b) ? a : b; // If a is greater than b, result = a; otherwise, result = b;
```

Understanding and utilizing these operators effectively allows Java developers to perform a wide range of operations, from basic arithmetic calculations to complex logical and bitwise manipulations in their programs.

Advantages of Operators in Java

The advantages of using operators in Java are mentioned below:

Expressiveness:
Operators in Java provide a concise and readable way to perform complex calculations and logical operations.

Time-Saving:
Operators in Java save time by reducing the amount of code required to perform certain tasks.

Improved Performance:
Using operators can improve performance because they are often implemented at the hardware level, making them faster than equivalent Java code.

Disadvantages of Operators in Java:
The disadvantages of Operators in Java are mentioned below:

Operator Precedence:
Operators in Java have a defined precedence, which can lead to unexpected results if not used properly.

Type conversions:
Java performs implicit type conversions when using operators, which can lead to unexpected results or errors if not used properly.

Overloading:
Java allows for operator overloading, which can lead to confusion and errors if different classes define the same operator with different behaviours.

Simple Input/Output:

simple input/output (I/O) statements are used to interact with users by reading input from the keyboard (or other input sources) and displaying output to the console (or other output destinations).
These operations are fundamental for building interactive and user-friendly applications.
Let's explore few common use cases for simple I/O in Java:

Use Case: Reading user name from user and great them.

```
import java.util.Scanner;
public class InputExample {
   public static void main (String [] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter your name: ");
      String name = scanner.nextLine();
      System.out.println("Hello, " + name + "! Welcome!");
      scanner.close();
   }
}
```

Use Case: Reading any two numbers from user and displaying their numeric sum after those inputs.

```
import java.util.Scanner;
public class NumericInputExample {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter first number: ");
      int number1 = scanner.nextInt();
      System.out.print("Enter second number: ");
      int number2 = scanner.nextInt();
                  int numberSum=number1+number2;
      System.out.println("Sum of "+number1+" and "+number2+" = " + numberSum);
      scanner.close();
   }
}
```

Use Case: Input Student firstName, rollNumber and avgMarks and displaying formatted output:

```
public class FormattedOutputExample {
   public static void main(String[] args) {
      String firstName = "Sarthak";
      int rollNumber = 27;
      double avgMarks = 5.8;
      System.out.printf("First Name: %s, Roll Number: %d, Avg Marks: %.1f %n", firstName, rollNumber, avgMarks);
   }
}
```

==========

Assignment based on topics Covered

==========

Multi choice Question and answers:

Q1.What is the size of the int data type in Java?

A) 1 byte
B) 2 bytes
C) 4 bytes
D) 8 bytes

Q2.Which of the following data types in Java is used to represent single-precision floating-point numbers?

A) float
B) double
C) float32
D) real

Q3.What is the default value of a boolean variable in Java if it is not explicitly initialized?

A) true
B) false
C) null
D) 0

Q4.Which data type in Java is used to store a single character?

A) char
B) string
C) character
D) byte

Q5.What is the maximum value that can be stored in an int data type in Java?
A) 2^15 - 1
B) 2^31 - 1
C) 2^63 - 1
D) It depends on the JVM implementation

Q6.Which of the following is a valid Java variable name?
A) 2count
B) _totalCount
C) break
D) my-variable

Q7.What is the convention for naming constants in Java?
A) camelCase
B) PascalCase
C) UPPER_CASE_WITH_UNDERSCORES
D) lowercase

Q8.What is the value of a constant variable in Java?
A) It can change during program execution.
B) It remains constant throughout the program execution.
C) It defaults to null if not explicitly initialized.
D) It depends on the data type of the variable.

Q9.Which of the following variable names is not allowed in Java?
A) totalAmount
B) _customerName
C) 3rdQuarterSales
D) myVariableName

Q10.What is the value of a variable if it is not explicitly initialized in Java?
A) It defaults to 0.
B) It defaults to null.
C) It contains garbage value.
D) It depends on the data type of the variable.

Q11.Which operator is used to perform addition in Java?
A) +
B) -
C) *
D) /

Q12.What is the result of the expression 5 % 2 in Java?
A) 2
B) 2.5
C) 1
D) 0.5

Q13.Which operator is used for logical AND in Java?
A) &&
B) ||
C) !
D) &

Q14.What is the value of x after the statement int x = 10; x++; in Java?
A) 10
B) 11
C) 9
D) 0

Q15.What is the result of the expression (5 > 3) && (7 < 4) in Java?
A) true
B) false
C) Compile-time error
D) Runtime error

Q16.Which of the following statements will result in a compilation error in Java?
A) int x = 10.5;
B) final int MAX_VALUE = 100;
C) boolean flag = "true";
D) int y = 10 / 0;

Q17.What will happen if you try to assign a value to a final variable after its declaration in Java?
A) compilation error.
B) overwrite the existing value.
C) throw a runtime exception.
D) prompt the user for a new value.

Q18.Which of the following expressions will result in a division by zero error in Java?
A) int result = 10 / 2;
B) int value = 5 % 0;
C) double ratio = 3.0 / 0;
D) int total = 10 + 2 * 3;

Q19.What is the result of the expression 10 > "5" in Java?
A) true
B) false
C) Compilation error
D) Runtime error

Q20.Which of the following declarations is incorrect in Java?
A) double price = 10.99;
B) int count = "5";
C) final String NAME = "John";
D) boolean flag = true;

Q21.What is the output of the following Java code snippet?

```
public class OutputQuestion {
    public static void main(String[] args) {
        int x = 5;
        int y = 2;
        double result = x / y;
        System.out.println(result);
    }
}
```

A) 2.5
B) 2.0
C) 2
D) Compilation error

Q22.What will be printed by the following Java code snippet?

```
public class StringConcatenationExample {
   public static void main(String[] args) {
      String webDomainName = "cyberinfomines";
      String webDomainExtensionName = "com";
      System.out.println(webDomainName + "." + webDomainExtensionName);
   }
}
```

A) cyberinfomines
B) com
C) cyberinfomines.com
D) Compilation error

Q23.What is the output of the following Java code snippet?

```
public class BooleanOutputExample {
   public static void main(String[] args) {
      boolean flag = true;
      System.out.println(!flag);
   }
}
```

A) true
B) false
C) Compilation error
D) Runtime error

Q24.What will be printed by the following Java code snippet?

```
public class ConditionalOperatorExample {
   public static void main(String[] args) {
      int x = 10;
      int y = 5;
      int result = (x > y) ? x : y;
      System.out.println(result);
   }
}
```

A) 10
B) 5
C) Compilation error
D) Runtime error

Q25.What is the output of the following Java code snippet?

```
public class ModulusExample {
   public static void main(String[] args) {
      int result = 10 % 3;
      System.out.println(result);
   }
}
```

A) 3
B) 1
C) 0
D) Compilation error

Coding Questions:
==========
Write an application which take any 3 numbers and swap them without using any 4th variable.
Write an application in java to get principle, rate of interest and time from user and find out total money that has to return back.
Ramesh has a floor of X feet length and Y feet breadth; he wants to make tiled floor. Floor tiles comes in packet of 4 tiles AXB per square feet.
Labour cost for tilling is RsP per square feet.
Write an application get all required information and calculate total floor area, number of tile packets required and labour cost and total cost for tilling floor.

Check your multi choice Question's answers:

Q1. Correct Answer: C) 4 bytes
Q2. Correct Answer: A) float
Q3. Correct Answer: B) false
Q4. Correct Answer: A) char
Q5. Correct Answer: B) 2^31 - 1
Q6. Correct Answer: B) _totalCount
Q7. Correct Answer: C) UPPER_CASE_WITH_UNDERSCORES
Q8. Correct Answer: B) It remains constant throughout the program execution.
Q9. Correct Answer: C) 3rdQuarterSales
Q10. Correct Answer: D) It depends on the data type of the variable.
Q11. Correct Answer: A) +
Q12. Correct Answer: C) 1
Q13. Correct Answer: A) &&
Q14. Correct Answer: B) 11
Q15. Correct Answer: B) false
Q16. Correct Answer: A) int x = 10.5;
Q17. Correct Answer: A) It will result in a compilation error.
Q18. Correct Answer: B) int value = 5 % 0;
Q19. Correct Answer: C) Compilation error
Q20. Correct Answer: B) int count = "5";
Q21. Correct Answer: B) 2.0
Q22. Correct Answer: C) cyberinfomines.com
Q23. Correct Answer: B) false
Q24. Correct Answer: A) 10
Q25. Correct Answer: B) 1