

Assignment 1 Java Programming Basics

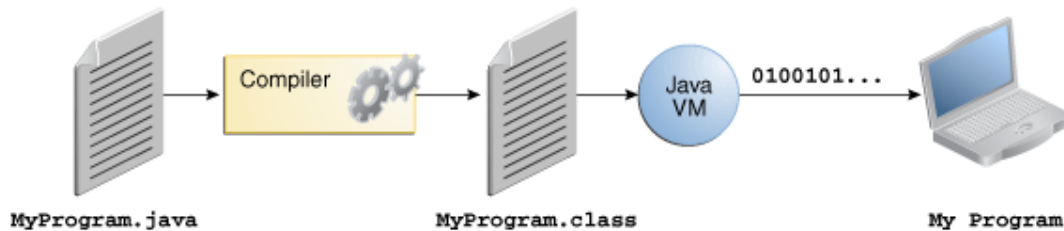
Name: Mansi Patil_KH

Part 1: Introduction to Java

1. What is Java? Explain its significance in modern software development.



- Java is a high-level, object oriented, and platform independent programming language.
- Developed by James Gosling at Sun Microsystems and released in 1995 (later acquired by Oracle).
- Java follows the principle of "Write Once, Run Anywhere" (WORA), meaning programs can run on any platform with a Java Virtual Machine (JVM).
- Designed for portability, security, and robustness.
- Java programs are compiled into bytecode, which runs on the Java Virtual Machine (JVM).
- In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine¹.
- The java launcher tool then runs the application with an instance of the Java Virtual Machine.



An overview of the software development process.

- Because the JVM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS.
- Used in desktop applications, mobile apps (Android), web applications, enterprise solutions, and cloud computing.

Its significance in modern software development.

Platform Independence (Write Once, Run Anywhere)

Java applications are compiled into bytecode (.class file). which can be executed on any machine that has a Java Virtual Machine (JVM), regardless of the underlying operating system.

Object-Oriented Programming (OOP)

- Java is based on OOP principles:
- Encapsulation (Data hiding through access modifiers).
- Abstraction (Hiding implementation details).
- Inheritance (Reusing code via class hierarchy).
- Polymorphism (Method overloading & overriding).

Secure

- No direct access to pointers, reducing memory leaks and security vulnerabilities.
- Java has a bytecode verifier that checks for illegal operations.

- Supports encryption and secure communication via APIs.

Robust & Reliable

- Strong memory management with automatic garbage collection.
- Exception handling mechanism (try-catch-finally) to manage runtime errors.
- No direct memory manipulation (e.g., pointer arithmetic is not allowed).

Community Support:

A large and active Java developer community provides extensive support, documentation, and learning resources, making it easier for developers to find solutions to challenges.

2. List and explain the key features of Java.



1.Simple & Familiar

- Java is easy to learn for programmers familiar with C and C++.
- No need to manage pointers or memory allocation manually.
- Provides automatic garbage collection.
- Java avoids the complexities of features like pointers, go-to statements, and multiple inheritance, promoting clean and understandable code.

2. Platform Independence:

- One of Java's groundbreaking features is its platform independence.
- The Java compiler converts source code into bytecode, which can be executed on any platform with a Java Virtual Machine (JVM).
- This cross-platform compatibility allows developers to write code on one machine and run it seamlessly on various operating systems, including Windows, Linux, and macOS.

3. Security:

- Security is a top priority in Java, making it a preferred choice for developing secure applications, such as those in the banking sector.
- Java's design eliminates the use of pointers, preventing unauthorized access to variables.
- Additionally, Java includes features like garbage collection, exception handling, and memory allocation, minimizing security vulnerabilities like stack corruption and buffer overflow.

4. Object-Oriented Programming (OOP):

- Java follows the principles of Object-Oriented Programming, organizing programs into objects and classes.
- The four main concepts of OOP — abstraction, encapsulation, inheritance, and polymorphism — are integral to Java's design, reusability, and maintainability in code.

5. Robustness & Reliable

- Strong memory management with automatic garbage collection.
- Exception handling mechanism (try-catch-finally) to manage runtime errors.
- No direct memory manipulation (e.g., pointer arithmetic is not allowed)

6. Distributed Computing & Networking Support

- Java supports network programming via built-in APIs.
- Can develop socket programming, web applications, and RMI (Remote Method Invocation) applications.
- Java applications can interact with databases, servers, and cloud services

7.Multithreading Support

- Java supports multithreading, allowing multiple tasks to run concurrently.
- Threads can be created using Thread class or Runnable interface.
- Synchronization prevents data inconsistency in multi-threaded programs.

8.High Performance

- Uses Just-In-Time (JIT) Compiler to convert bytecode into native machine code at runtime.
- Optimization techniques like HotSpot Compiler improve performance.

- Slower than C/C++, but performance is optimized for real-world applications.

9. Dynamic & Extensible

- Java supports dynamic memory allocation and loading of classes at runtime.
- Allows developers to extend existing applications using APIs, libraries, and frameworks.
- Uses Reflection API to inspect and modify classes at runtime.

10. Backward Compatibility

- Java ensures that older Java programs still work on newer versions of the language.
- New versions of Java introduce enhancements without breaking old code

3. What is the difference between compiled and interpreted languages? Where does Java fit in?



- **Compiled languages** are programming languages that are converted into executable machine code by a compiler before they can run.
 - This means that you need to compile your source code every time you make a change, and that you need to create different versions of your executable for different operating systems and architectures.
 - Some examples of compiled languages are C, C++, Java, and Rust.
 - The main advantages of compiled languages are that they tend to be faster, more efficient, and more secure than interpreted languages, as they have direct access to the hardware and can optimize the code during compilation.
- The main disadvantages are that they require more time and resources to compile, debug, and distribute, and that they are less flexible and portable than interpreted languages.

- **Interpreted languages** are programming languages that are executed by an interpreter at runtime.
- This means that you do not need to compile your source code before running it, and that you can run the same code on different platforms and architectures, as long as you have the interpreter installed.
- Some examples of interpreted languages are Python, Ruby, JavaScript, and PHP.
- The main advantages of interpreted languages are that they are easier to write, test, and modify, and that they offer more dynamism and portability than compiled languages.
- The main disadvantages are that they tend to be slower, less efficient, and less secure than compiled languages, as they have to go through an extra layer of abstraction and cannot optimize the code as much as compilers.

Java is a **hybrid** language. It is neither purely compiled nor purely interpreted.

Compilation: Java source code is first compiled into an intermediate form called **bytecode** by the Java compiler (javac). This bytecode is platform-independent and does not contain machine-specific code.

Interpretation/Execution: The bytecode is then executed by the **Java Virtual Machine (JVM)**, which interprets or just-in-time (JIT) compiles the bytecode into native machine code for the specific platform on which the Java program is running.

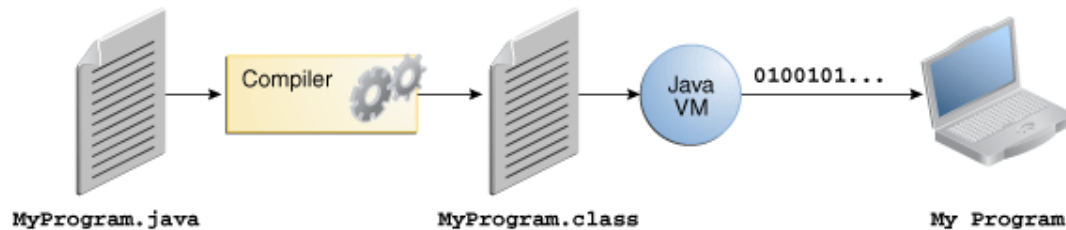
4. Explain the concept of platform independence in Java.

Platform Independence (Write Once, Run Anywhere - WORA)

- Java code is compiled into bytecode (.class file).
- The JVM interprets bytecode, making Java platform-independent.
- Can run on Windows, Linux, Mac, etc., without modification.

Java programs are compiled into bytecode, which runs on the Java Virtual Machine (JVM).

- In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine¹.
- The java launcher tool then runs the application with an instance of the Java Virtual Machine.



An overview of the software development process.

- Because the JVM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS.
- Used in desktop applications, mobile apps (Android), web applications, enterprise solutions, and cloud computing.

5. What are the various applications of Java in the real world?



1. Web Applications

Because Java is capable of interacting with various systems simultaneously, it is ideal for web app development. Hence, it allows building dynamic web apps that can easily interact with multiple interfaces.

The web development process is made more feasible with the use of web servers, Hibernate, JSP, and Spring. Some of its benefits include improved security, code reusability, availability of a large number of APIs, amazing IDEs, tools, and more.

2. Mobile Applications

Utilizing Java, developers can build cross-platform mobile applications. This means such apps are functional on all mobile operating systems and all sizes of screen devices. A survey states that Java is the second most used programming language for mobile app development. Some of the popular mobile apps empowered by Java include Spotify, Twitter, and Netflix.

Reasons that appeal the most to the developers to use Java for mobile app development are:

- Strengthened Security
- Simple code
- Highly Compatible
- Cross-platform Functioning

3. Gaming Applications

Java is one of the most desirable languages for programming gaming applications. Because it comes with a large range of open-source development frameworks.

Many popular games including Asphalt 6, Minecraft, and Mission Impossible 3 were created using Java. Developing 2D and 3D Android games is easy with Java as it offers jMonkeyEngine and Dalvik Virtual Machine.

4. Business Applications

Companies sometimes need to create robust applications to fulfill their business needs. This programming language is evolving continuously. So, it is capable of fulfilling the latest business requirements. For business app development, Java offers:

- Cross-platform functionality
- Robust performance in managing the workload of large enterprises
- Flexible integration

5. Cloud Applications

Cloud applications are the ones that store the user's data in a cloud. The user needs an internet connection to use cloud apps and all of their work and information are updated in real-time on the cloud.

And yes, Java helps build cloud apps too. In addition to applications, Java also allows developers to build cloud servers. They act as an affordable alternative for expensive IT infrastructures.

6. Scientific Applications

Java comes with advanced security features which makes it suitable for creating scientific apps. This language can serve well as a robust tool for programming complex mathematical operations.

Such programs must be written with more efficiency and security. Java serves as the core component in many top scientific applications like MATLAB.

7. Desktop GUI Applications

Java allows developers to design desktop applications with perfection. A modern approach to developing GUI apps is to utilize APIs like JavaFX, AWT, and Swing. You can enjoy various advantages such as simultaneous display of multiple instances, visual feedback, ease of learning, and more while using Java for desktop app development.

8. Embedded Systems

In an embedded system, various small units combine to perform a collective function for a large system. And when it comes to simplifying complex software solutions, Java has a proven track record.

Many developers use Java for Embedded Systems. The language provides a wide range of libraries which makes the development tasks easy. Embedded applications can just reuse them which greatly improves productivity.

Java follows the object-oriented programming approach to development. This makes it easy even for less experienced developers or beginners to create embedded systems. Java also offers a variety of features that can help developers manage complex systems.

9. Big Data Technology

Being a software utility, analyzing and extracting information from complex data structures is what Java was built for. It has a wide range of applications in the latest technological trends like AI, ML, and deep learning. Java is also considered as a viewpoint for Big data. Many popular ETL applications like Apache Kafka and Apatar are built upon Java programming language. Moreover, using Java provides them with unique features such as a stack provision system and automatic garbage selection. These can come in handy as it gives you an edge in the market.

10. Artificial Intelligence

The infrastructure of Java is embedded with intelligent software that can improve artificial intelligence programming. This makes Java one of the ideal programming languages for AI projects.

Standard widget tools, easy coding features, easy debugging, better user interactions, and many more great features come with this language. The process of AI programming can be perfected with the help of Java.

Part 2: History of Java

1. Who developed Java and when was it introduced?



Developed by James Gosling at Sun Microsystems and released in 1995.

2. What was Java initially called? Why was its name changed?



Initially, Gosling named the language “Oak” after an oak tree outside his office. However, due to trademark issues, the team later renamed it “Java,” inspired by their love for coffee. The first public implementation of Java, Java 1.0a, was released in 1995.

3. Describe the evolution of Java versions from its inception to the present.



Major Milestones in Java's Evolution

Year	Milestone
1991	James Gosling and team started working on “Oak” (later renamed Java).
1995	Java 1.0 officially released by Sun Microsystems.
1996	First Java Development Kit (JDK 1.0) launched.
1997	Java became the official language for web development.
1999	Java 2 (J2SE, J2EE, J2ME) introduced, bringing significant improvements.
2006	Sun Microsystems made Java open-source under GPL.
2010	Oracle acquired Sun Microsystems, taking over Java development.
2014	Java 8 released, introducing Lambda Expressions & Stream API.
2017	Oracle switched to a faster Java release cycle (every 6 months).
2018	Java 11 became a long-term support (LTS) version.
2021	Java 17 released as the next LTS version with modern features.
2024	Java 21 (latest LTS version) released, bringing virtual threads and pattern matching.

4. What are some of the major improvements introduced in recent Java versions?



Recent java focused on performance, security, usability.

Java 11

- Long-Term Support (LTS): Java 11 is an LTS release, which means it will receive extended support, making it a preferred version for enterprises.
- String Methods: New methods like `isBlank()`, `lines()`, `strip()`, and `repeat()` were added to the `String` class for improved text manipulation.
- HTTP Client (Standardized): The HTTP Client API, which was incubated in Java 9, became a standard part of the JDK in Java 11. It simplifies HTTP communication and supports both synchronous and asynchronous operations.

Java 17

- Sealed Classes (Finalized): Sealed classes were finalized and became part of the language, allowing developers to define restricted class hierarchies.
- Strong Encapsulation of JDK Internals: Continuing efforts from Java 16, Java 17 further strengthens the encapsulation of JDK internals, enforcing stricter boundaries.
- New macOS Rendering Pipeline: A new rendering pipeline for macOS using Apple's Metal API, improving performance on macOS devices.

Java 21

- Virtual Threads: Fully integrating virtual threads into the platform to handle concurrency and parallelism in a more efficient manner.

5. How does Java compare with other programming languages like C++ and Python in terms of evolution and usability?

<u>Feature</u>	<u>Java</u>	<u>C++</u>	<u>Python</u>
<u>Performance</u>	<u>High (JVM optimization, garbage collection)</u>	<u>Extremely High (manual memory management, low-level access)</u>	<u>Moderate (slower than Java and C++)</u>
<u>Ease of Use</u>	<u>Easy to learn but can be verbose</u>	<u>Steep learning curve, complex syntax</u>	<u>Very easy to learn, concise syntax</u>
<u>Cross-Platform Support</u>	<u>Excellent (JVM-based)</u>	<u>Excellent (compiled to platform-specific binaries)</u>	<u>Excellent (interpreted, works on most platforms)</u>
<u>Memory Management</u>	<u>Automatic (via garbage collection)</u>	<u>Manual (via pointers)</u>	<u>Automatic (via garbage collection)</u>
<u>Libraries/Frameworks</u>	<u>Vast (Spring, Hibernate, etc.)</u>	<u>Large (but lower-level than Java)</u>	<u>Extensive (Data Science, Web, ML, etc.)</u>
<u>Enterprise Adoption</u>	<u>Very high (large enterprise applications)</u>	<u>High (performance-critical applications)</u>	<u>High (especially in web and data science)</u>
<u>Type System</u>	<u>Statically Typed</u>	<u>Statically Typed (complex)</u>	<u>Dynamically Typed</u>
<u>Garbage Collection</u>	<u>Yes (Automatic)</u>	<u>No (manual management)</u>	<u>Yes (Automatic)</u>

<u>Modern Features</u>	<u>Introduced functional programming and other modern features</u>	<u>Modern features added slowly (e.g., lambda expressions, coroutines)</u>	<u>Rapid adoption of new features (asyncio, type hints, etc.)</u>
------------------------	--	--	---

Part 3: Data Types in Java

1. Explain the importance of data types in Java.



Data types define the kind of data a variable can hold. Set the rules — whether it's a number, a string of text, or even a true/false condition — that help programs know how to interpret and process the information.

a. Memory Efficiency

To optimize your system is through memory management, and data types play a huge role . When you select the appropriate data type, you're telling the system exactly how much space it should allocate for each value. For example, you might use float32 instead of float64 when you don't need high precision. Because float32 takes up half the memory, and that savings can really add up in large datasets.

b. Performance Optimization

Data processing speed is the name of the game, especially in big data applications. That performance hinges on the algorithms or hardware, but the right choice of data types is often an overlooked lever for optimization.

When you use smaller, more efficient data types, you reduce the workload on your system's processor. For instance, using integers instead of floats where precision isn't critical can give your code a speed boost. On a large scale, this can mean the difference between a process running in seconds versus minutes — or even hours.

c. Accuracy and Data Integrity

Improper data types can mess with your data integrity.

For example, if performing scientific calculations or financial modeling where precision is everything, the smallest rounding error could cascade into a huge problem. Some systems might even throw overflow errors if integers exceed their storage capacity.

d. Data Validation

When assign a type to each variable, not just specifying how it should be stored, also telling program what kind of data it should expect.

For instance, setting a variable to int ensures that you can only store whole numbers, preventing unexpected data from creeping in. If function expects a boolean and try to feed it a string, system will catch that error

e. Error Prevention

proper data typing helps with error handling.

2. Differentiate between primitive and non-primitive data types.



Primitive Data Types	Non-Primitive Data Types
These are the most basic data types in programming languages.	This is complex and used for one or more primitive data types.
It is used to represent simple values such as integers, booleans, and characters.	It is used to represent more complex data objects such as arrays, queues, trees, and stacks.
They have a fixed size and range of values.	Non-primitive can be resized or modified during runtime.
These data types are predefined in the programming language.	These are usually defined or created by the programmer based on needs.
Primitive data types are generally immutable, meaning their value cannot be changed once created.	Non-primitive data types are mutable, meaning their contents can be modified.
Primitive data types are usually stored in the stack memory.	Non-primitive data types are typically stored in heap memory.
It uses basic operations such as addition, subtraction, etc.	It uses complex operations such as traversal, sorting, and searching.
It is useful for small and simple programs.	It is used for handling large sets of data, and efficient algorithms.

3. List and briefly describe the eight primitive data types in Java.



The eight primitive data types supported by the Java programming language are:

- **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127. The byte data type can be useful for saving memory in large arrays, where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.
- **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767. can use a short to save memory in large arrays, in situations where the memory savings actually matters.
- **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -2^{31} and a maximum value of $2^{31}-1$. In Java SE 8 and later,

you can use the int data type to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of $2^{32}-1$.

- **long:** The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$. In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of $2^{64}-1$. Use this data type when you need a range of values wider than those provided by int.
- **float:** The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the Floating-Point Types, Formats, and Values section of the Java Language Specification, use a float (instead of double) if need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency.
- **double:** The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.
- **boolean:** The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
- **char:** The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

4. Provide examples of how to declare and initialize different data types.



byte:

```
byte b = 100;
```

short

```
short s = 5000;
```

int

```
int age = 25;
```

long

```
long distance = 150000L;
```

float

```
float price = 19.99f;
```

double

double weight = 75.5;

char

char letter = 'A';

boolean

boolean java = true;

5. What is type casting in Java? Explain with an example.

Typecasting in Java is the process of converting one data type to another data type using the casting operator. When you assign a value from one primitive data type to another type, this is known as type casting.

There are two types of Type Casting in java:

- **Widening Type Casting**

byte -> short -> int -> long -> float -> double

A lower data type is transformed into a higher one.

Implicit casting, also known as automatic conversion, is the process where the Java compiler automatically converts a smaller data type into a larger data type.

This type of casting is safe and does not require any explicit operator or additional code from the programmer. The reason it's considered safe is that when converting from a smaller to a larger data type, there's no risk of losing information or precision.

- **Narrow Type Casting**

double -> float -> long -> int -> short -> byte

The process of downsizing a bigger data type into a smaller one.

This process involves converting a larger data type into a smaller one, which could potentially lead to data loss.

Because of this risk, explicit casting must be performed manually by the programmer, using a casting operator to specify the desired conversion.

Explicit casting is used when the conversion will not lead to unacceptable data loss, or when such loss is a calculated and acceptable part of your program's logic.

6. Discuss the concept of wrapper classes and their usage in Java.



Each of Java's eight primitive data types has a class dedicated to it. These are known as wrapper classes because they "wrap" the primitive data type into an object of that class. The wrapper classes are part of the java.lang package, which is imported by default into all Java programs.

The wrapper classes in java to provide utility functions for primitives like converting primitive types to and from string objects, converting to various bases like binary, octal or hexadecimal, or comparing various objects.

The following two statements illustrate the difference between a primitive data type and an object of a wrapper class:

```
int x = 25;
```

```
Integer y = new Integer(33);
```

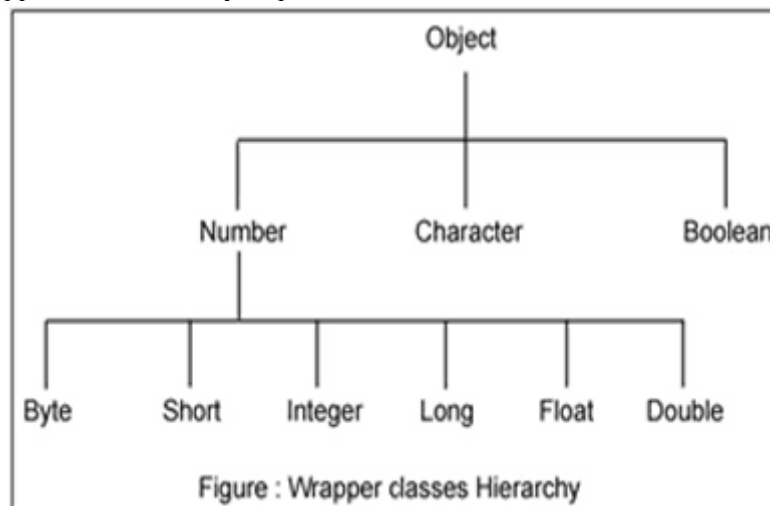
The first statement declares an int variable named x and initializes it with the value 25. The second statement instantiates an Integer object. The object is initialized with the value 33 and a reference to the object is assigned to the object variable y.

Below table lists wrapper classes in Java API with constructor details.

Primitive	Wrapper Class	Constructor Argument
-----------	---------------	----------------------

boolean	Boolean	boolean or String
byte	Byte	byte or String
char	Character	char
int	Integer	int or String
float	Float	float, double or String
double	Double	double or String
long	Long	long or String
short	Short	short or String

Below is wrapper class hierarchy as per Java API



1.Object-Oriented Features working with collections, require objects rather than primitive data types. For instance, Java's ArrayList can store objects but not primitive types. By using wrapper classes, we can store primitive values in collections like ArrayList or HashMap.

```
ArrayList<Integer> numbers = new ArrayList<>();
```

```
numbers.add(5); // Here, the int 5 is "wrapped" into an Integer object.
```

2. Utility Methods: Wrapper classes provide several useful methods. For example, the Integer class has methods like parseInt() to convert a String into an int.

```
String number = "100";
```

```
int result = Integer.parseInt(number);
```

3. Type Conversions: Wrapper classes allow easy conversion between types. For instance, Integer provides methods to convert integers to binary, hexadecimal, etc.

```
int num = 15;
```

```
String binaryString = Integer.toBinaryString(num); // "1111"
```

7. What is the difference between static and dynamic typing? Where does Java stand?

Static Typing

- **Type Checking Time:** Type checking is done at compile time.
- **Type Declarations:** You must specify the types of variables and objects at the time of writing the code, and the compiler ensures that values assigned to these variables match the declared types.
- **Examples:** Java, C, C++, Swift, Kotlin.
- **Advantages:**
 - **Early Detection of Errors:** Type errors are caught early during the compilation phase.
 - **Performance:** Since the types are known at compile time, certain optimizations can be made by the compiler, potentially leading to better performance.

Dynamic Typing

- **Type Checking Time:** Type checking is done at runtime, when the program is executed.
- **Type Declarations:** You do not need to explicitly declare types for variables or objects; the type is inferred by the interpreter or runtime system based on the value assigned.
- **Examples:** Python, JavaScript, Ruby, PHP.
- **Advantages:**
 - **Flexibility:** You can write more flexible and concise code since you don't have to specify types explicitly.
 - **Faster Prototyping:** Developers can quickly write and modify code without worrying about type definitions, making it faster for prototyping and scripting.

Java is a **statically typed** language.

- **Static Typing in Java:** Declare the type of a variable or object when write the code. The Java compiler ensures that types are correct during compile time, and type-related errors are flagged at that point.
- For example:
`String name = "Alice";`
`int age = 30;`

Coding Questions on Data Types:

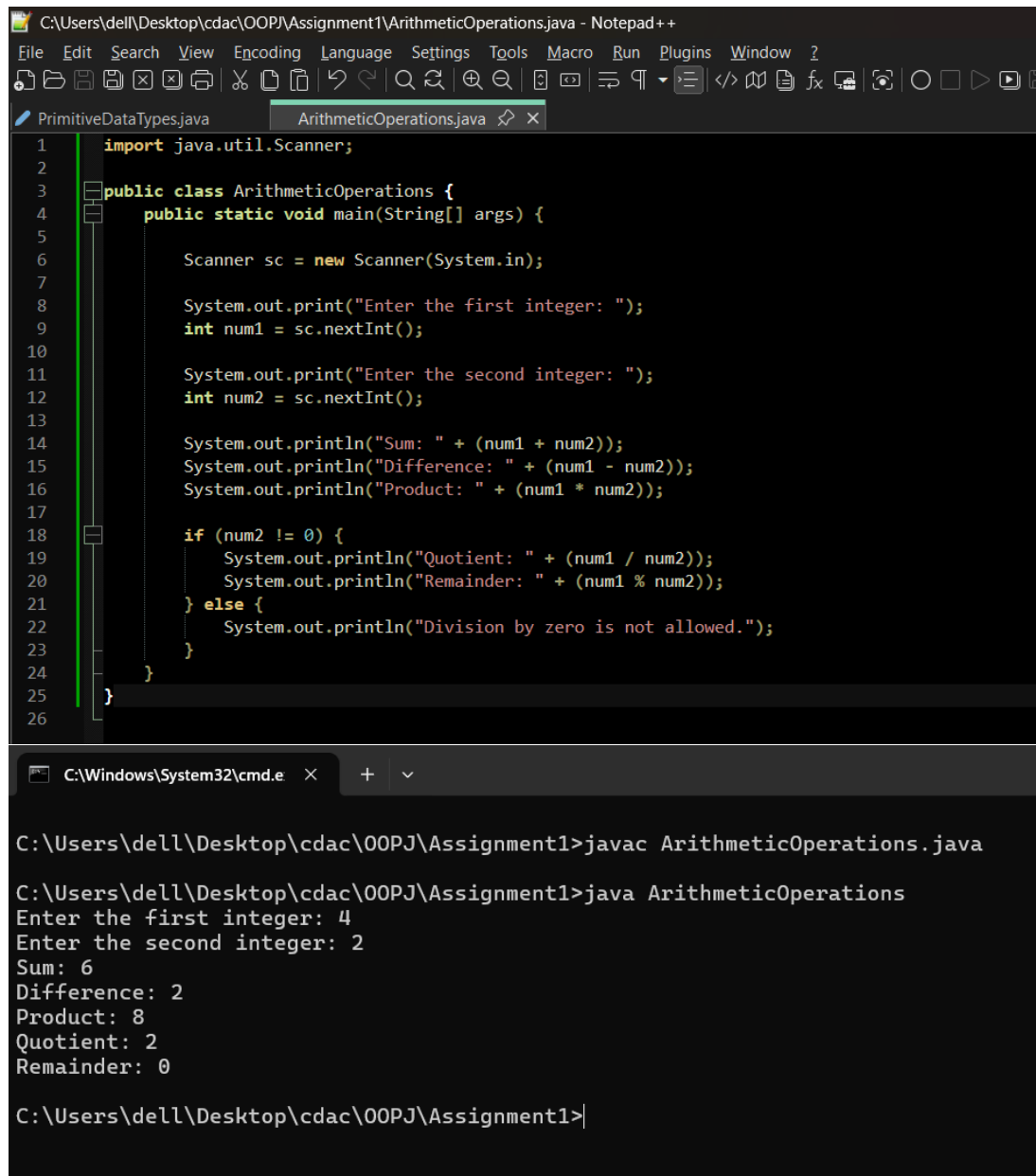
1. Write a Java program to declare and initialize all eight primitive data types and print their values.

Program:

```
*C:\Users\dell\Desktop\cdac\OOPJ\Assignment1\PrimitiveDataTypes.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
PrimitiveDataTypes.java x
1 public class PrimitiveDataTypes{
2     public static void main(String[] args) {
3
4         byte by = 100;
5         short s = 5000;
6         int i = 100000;
7         long l = 10000000000L;
8
9         float f = 3.14f;
10        double d = 3.14159;
11        char c = 'A';
12        boolean b = true;
13
14        System.out.println("byte value: " + by);
15        System.out.println("short value: " + s);
16        System.out.println("int value: " + i);
17        System.out.println("long value: " + l);
18        System.out.println("float value: " + f);
19        System.out.println("double value: " + d);
20        System.out.println("char value: " + c);
21        System.out.println("boolean value: " + b);
22    }
23 }
24

C:\Windows\System32\cmd.e x + v
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>javac PrimitiveDataTypes.java
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>java PrimitiveDataTypes
byte value: 100
short value: 5000
int value: 100000
long value: 10000000000
float value: 3.14
double value: 3.14159
char value: A
boolean value: true
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>|
```

2. Write a Java program that takes two integers as input and performs all arithmetic operations on them.

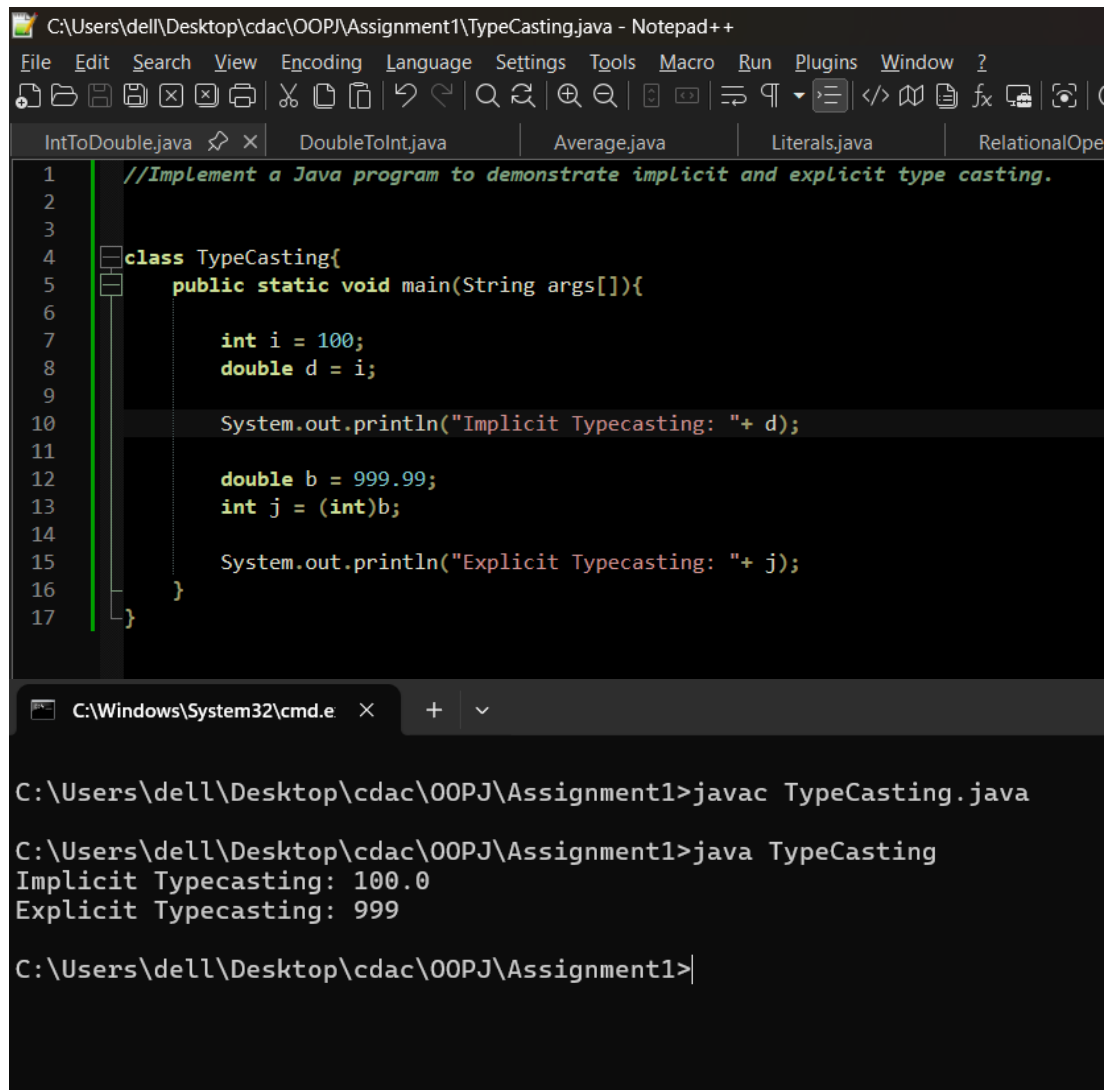


The image shows a Notepad++ editor window with the file `ArithmeticOperations.java` open. The code defines a `public class ArithmeticOperations` with a `main` method that uses `Scanner` to take two integers as input and performs addition, subtraction, multiplication, and division (with a zero-check). Below the editor, a Windows command prompt shows the compilation and execution of the program, with input values 4 and 2, and the resulting arithmetic outputs.

```
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1\ArithmeticOperations.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
PrimitiveDataTypes.java ArithmeticOperations.java x
1 import java.util.Scanner;
2
3 public class ArithmeticOperations {
4     public static void main(String[] args) {
5
6         Scanner sc = new Scanner(System.in);
7
8         System.out.print("Enter the first integer: ");
9         int num1 = sc.nextInt();
10
11        System.out.print("Enter the second integer: ");
12        int num2 = sc.nextInt();
13
14        System.out.println("Sum: " + (num1 + num2));
15        System.out.println("Difference: " + (num1 - num2));
16        System.out.println("Product: " + (num1 * num2));
17
18        if (num2 != 0) {
19            System.out.println("Quotient: " + (num1 / num2));
20            System.out.println("Remainder: " + (num1 % num2));
21        } else {
22            System.out.println("Division by zero is not allowed.");
23        }
24    }
25 }
26
```

```
C:\Windows\System32\cmd.e x + v
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>javac ArithmeticOperations.java
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>java ArithmeticOperations
Enter the first integer: 4
Enter the second integer: 2
Sum: 6
Difference: 2
Product: 8
Quotient: 2
Remainder: 0
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>
```

3. Implement a Java program to demonstrate implicit and explicit type casting.



The image shows a Notepad++ editor window with the file `TypeCasting.java` open. The code implements a Java program to demonstrate implicit and explicit type casting. Below the editor, a Windows command prompt window shows the compilation and execution of the program.

```
//Implement a Java program to demonstrate implicit and explicit type casting.

class TypeCasting{
    public static void main(String args[]){

        int i = 100;
        double d = i;

        System.out.println("Implicit Typecasting: "+ d);

        double b = 999.99;
        int j = (int)b;

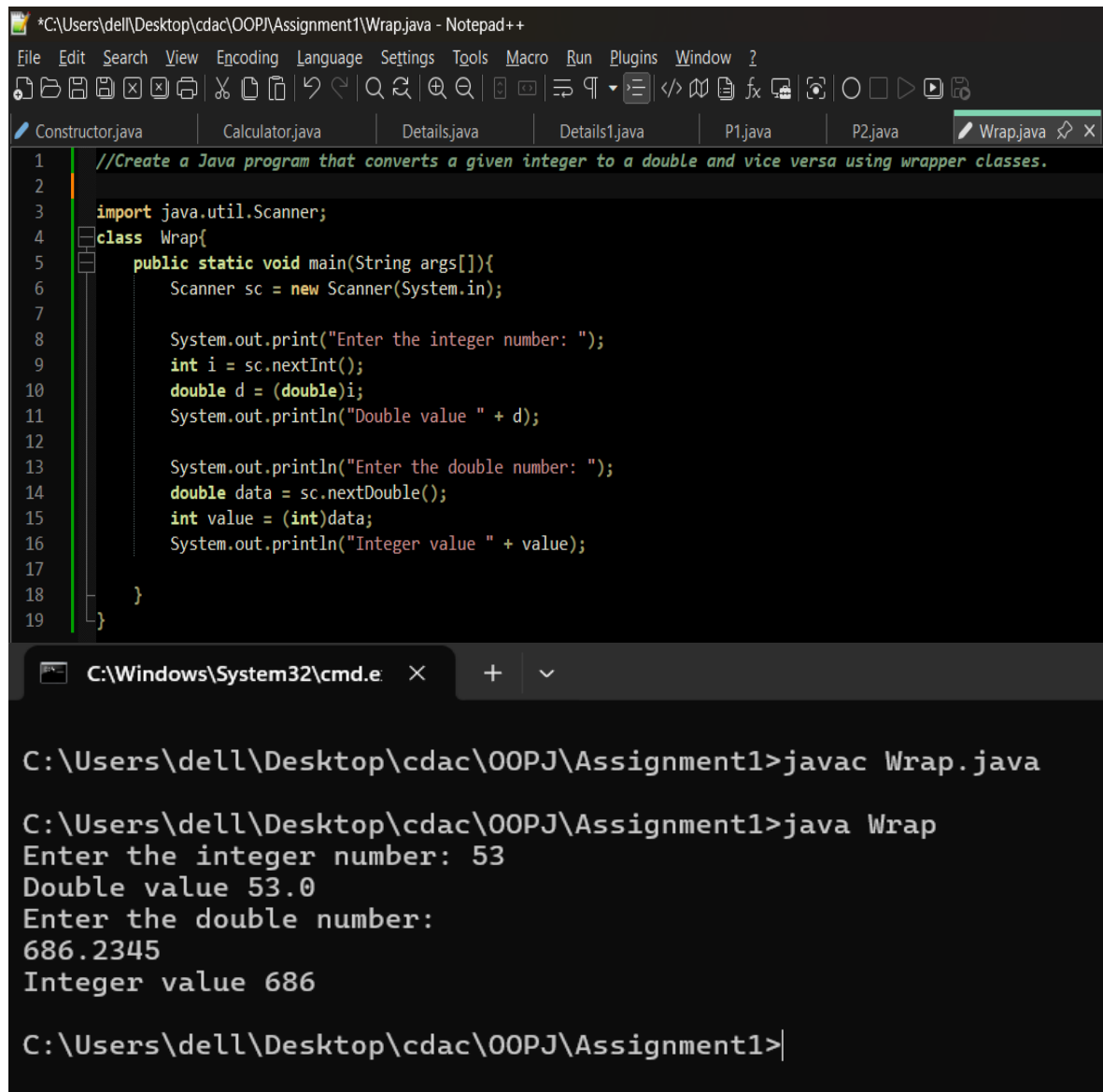
        System.out.println("Explicit Typecasting: "+ j);
    }
}
```

```
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>javac TypeCasting.java

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>java TypeCasting
Implicit Typecasting: 100.0
Explicit Typecasting: 999

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>|
```


4. Create a Java program that converts a given integer to a double and vice versa using wrapper classes.



The screenshot displays a Notepad++ window titled '*C:\Users\dell\Desktop\cdac\OOPJ\Assignment1\Wrap.java - Notepad++'. The editor contains the following Java code:

```
1 //Create a Java program that converts a given integer to a double and vice versa using wrapper classes.
2
3 import java.util.Scanner;
4 class Wrap{
5     public static void main(String args[]){
6         Scanner sc = new Scanner(System.in);
7
8         System.out.print("Enter the integer number: ");
9         int i = sc.nextInt();
10        double d = (double)i;
11        System.out.println("Double value " + d);
12
13        System.out.println("Enter the double number: ");
14        double data = sc.nextDouble();
15        int value = (int)data;
16        System.out.println("Integer value " + value);
17    }
18 }
19 }
```

Below the editor, a Windows command prompt window is open, showing the compilation and execution of the program:

```
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>javac Wrap.java

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>java Wrap
Enter the integer number: 53
Double value 53.0
Enter the double number:
686.2345
Integer value 686

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>|
```

5. Write a Java program to swap two numbers using a temporary variable and without using a temporary variable.

```
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1\SwapNumbers.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
PrimitiveDataTypes.java ArithmeticOperations.java SwapNumbers.java x
1 import java.util.Scanner;
2 class SwapNumbers{
3     public static void main(String args[]){
4         Scanner sc = new Scanner(System.in);
5
6         System.out.println("Swapping using a temporary variable:");
7         System.out.print("Enter the first number: ");
8         int a = sc.nextInt();
9         System.out.print("Enter the first number: ");
10        int b = sc.nextInt();
11
12        int c = a;
13        a = b;
14        b = c;
15
16        System.out.println("After swapping a : " + a);
17        System.out.println("After swapping b : " + b);
18
19        System.out.println("Swapping without using a temporary variable:");
20        System.out.print("Enter the first number again: ");
21        a = sc.nextInt();
22        System.out.print("Enter the second number again: ");
23        b = sc.nextInt();
24
25        a = a + b;
26        b = a - b;
27        a = a - b;
28        System.out.println("After Swapping a : " + a);
29        System.out.println("After Swapping b : " + b);
30    }
31 }
32

C:\Windows\System32\cmd.e x + v
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>javac SwapNumbers.java
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>java SwapNumbers
Swapping using a temporary variable:
Enter the first number: 4
Enter the first number: 2
After swapping a : 2
After swapping b : 4
Swapping without using a temporary variable:
Enter the first number again: 4
Enter the second number again: 2
After Swapping a : 2
After Swapping b : 4
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>
```

6. Develop a program that takes user input for a character and prints whether it is a vowel or consonant.

```
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1\Character.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
ArithmeticOperations.java SwapNumbers.java IntToDouble.java DoubleToInt.java

1  import java.util.Scanner;
2
3  class Character{
4      public static void main(String args[]){
5          Scanner sc = new Scanner(System.in);
6          char ch = sc.next().charAt(0);
7          if(ch>='A' && ch<='Z'){
8              if(ch=='A' || ch=='E' || ch=='I' || ch=='O' || ch=='U'){
9                  System.out.println(ch+" is a vowel.");
10             }else{
11                 System.out.println(ch+" is a consonant.");
12             }
13         }else if(ch>='a' && ch<='z'){
14             if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u'){
15                 System.out.println(ch+" is a vowel.");
16             }else{
17                 System.out.println(ch+" is a consonant.");
18             }
19         }else if(ch>='0' && ch<='9'){
20             System.out.println(ch+" is a number.");
21         }else{
22             System.out.println(ch+" is a special character.");
23         }
24     }
25 }
```

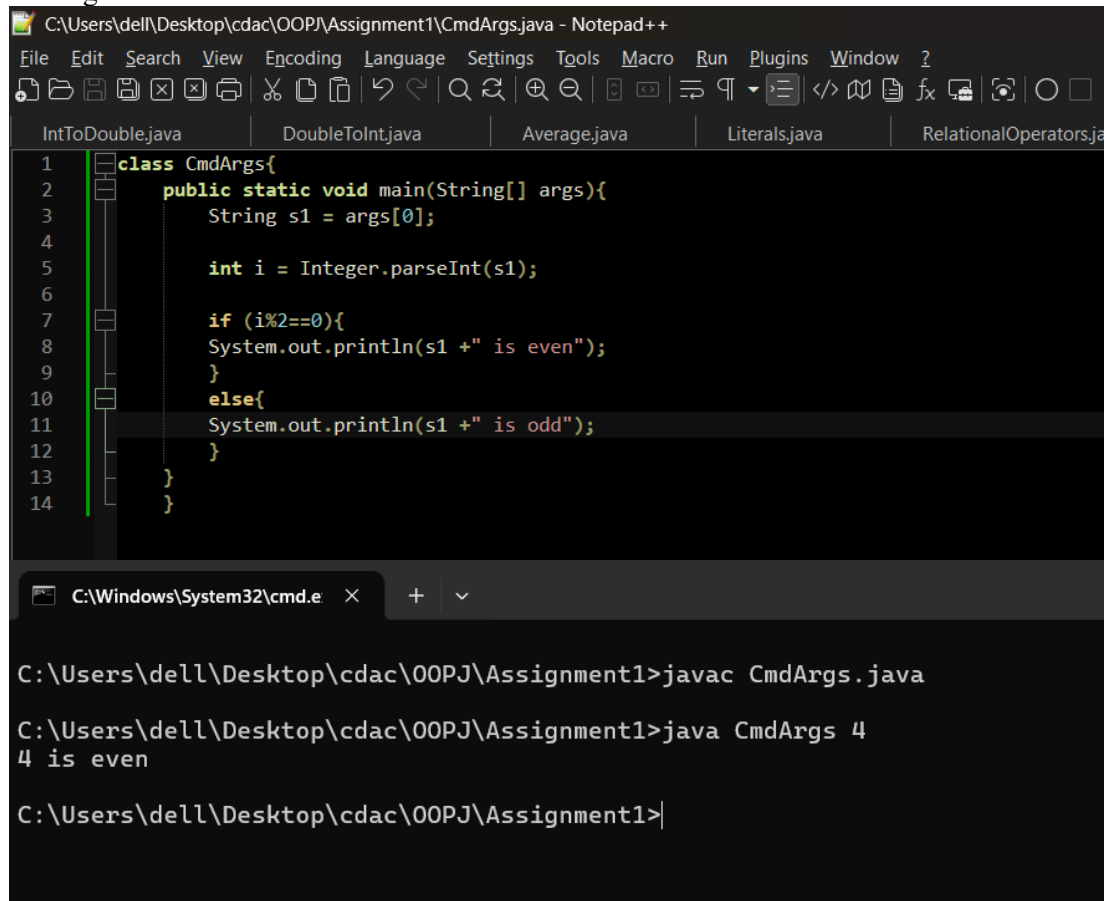
```
C:\Windows\System32\cmd.e  x  +  v

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>javac Character.java

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>java Character
e
e is a vowel.

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>
```

7. Create a Java program to check whether a given number is even or odd using command-line arguments.



The image shows a Notepad++ window titled "C:\Users\dell\Desktop\cdac\OOPJ\Assignment1\CmdArgs.java - Notepad++". The editor contains the following Java code:

```
1 class CmdArgs{
2     public static void main(String[] args){
3         String s1 = args[0];
4
5         int i = Integer.parseInt(s1);
6
7         if (i%2==0){
8             System.out.println(s1 + " is even");
9         }
10        else{
11            System.out.println(s1 + " is odd");
12        }
13    }
14 }
```

Below the editor, a Windows command prompt window is open, showing the following commands and output:

```
C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>javac CmdArgs.java

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>java CmdArgs 4
4 is even

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>|
```

Part 4: Java Development Kit (JDK)

1. What is JDK? How does it differ from JRE and JVM?



JDK: It is a software development environment that is used to develop java applications. The JDK is a complete package that allows developers to write, compile, and debug Java applications. It includes the JRE, along with development tools.

JRE vs JDK vs JVM

JDK = JRE + Development Tools (compiler, debugger, Interpreter, etc.).

JRE = JVM + Core Libraries (Only for running Java apps), runtime files (no compiler)

JVM = runs java applications by converting bytecode to machine code.

2. Explain the main components of JDK.



Component	Description
JRE (Java Runtime Environment)	Required to run Java applications.
Compiler (javac)	Converts Java source code (.java) into bytecode (.class).
Java Virtual Machine (JVM)	It executes the compiled bytecode.
Java Runtime Environment (JRE)	It includes libraries and JVM which are necessary to run a java applications.
Debugger (jdb)	Helps debug Java programs.
JavaDoc (javadoc)	Generates documentation from Java comments.
Java Archive (jar)	Packages multiple .class files into a single .jar file.
API libraries	Predefined classes and methods for java development
Other Development Tools	Profilers, monitoring tools, jheap, jconsole, etc.

3. Describe the steps to install JDK and configure Java on your system.

Step 1: Download the JDK

1. Go to the official [Java Downloads page](#) (or the specific version you want).
2. Select the appropriate version of the JDK for your system (e.g., **Windows x64**).
3. Click on the **Download** button, accept the license agreement, and download the installer .exe file.

Step 2: Install the JDK

1. Once the .exe file is downloaded, double-click it to start the installation.
2. Follow the installation wizard steps:
 - Choose the installation directory (default is typically C:\Program Files\Java\jdk-<version>).
 - Select the installation features .
 - Click **Next** and then **Install** to proceed.

Step 3: Set the JAVA_HOME Environment Variable

1. Right-click on **This PC** or **Computer** and select **Properties**.
2. Click on **Advanced system settings**.
3. Under the **System Properties** window, click on the **Environment Variables** button.
4. In the **System Variables** section, click **New** to create a new variable:
 - a. **Variable Name:** JAVA_HOME
 - b. **Variable Value:** The path to the JDK installation directory (e.g., C:\Program Files\Java\jdk-<version>).
5. Click **OK**.

Step 4: Add JDK to the PATH Environment Variable

1. In the **System Variables** section, find the Path variable, select it, and click **Edit**.
2. Click **New** and add the path to the bin directory inside the JDK installation (e.g., C:\Program Files\Java\jdk-<version>\bin).
3. Click **OK** to save the changes.

Step 5: Verify the Installation

1. Open **Command Prompt** (cmd).
2. Type the following command to verify that Java is installed:
java -version

This should display the version of Java installed on your system.

4. Write a simple Java program to print “Hello, World!” and explain its structure.



The screenshot shows a Notepad++ window titled "C:\Users\dell\Desktop\cdac\OOPJ\Assignment1\HelloWorld.java - Notepad++". The code in the editor is as follows:

```
1 class HelloWorld{
2     public static void main(String args[]){
3         System.out.println("Hello,World!");
4     }
5 }
```

Below the editor is a Windows Command Prompt window titled "C:\Windows\System32\cmd.e". It shows the following commands and output:

```
Microsoft Windows [Version 10.0.26100.3194]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>javac HelloWorld.java

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>java HelloWorld
Hello,World!

C:\Users\dell\Desktop\cdac\OOPJ\Assignment1>|
```

Explanation of the Structure:

1. Class Declaration:

```
class HelloWorld {
```

- o **class:** The keyword used to define a class in Java.

- **HelloWorld:** This is the name of the class. In Java, class names should start with an uppercase letter, and by convention, they are written in **CamelCase**.
- **{:** The opening curly brace begins the body of the class.

2. Main Method:

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

- **public:** This means the main method is accessible from anywhere.
- **static:** This keyword means that the method belongs to the class itself, rather than instances of the class.
- **void:** This means the main method doesn't return any value.
- **main:** The name of the method that the Java runtime looks for when starting a program.
- **String[] args:** This parameter is an array of **Strings**. It's used to accept command-line arguments when the program is run.

3. System.out.println():

```
System.out.println("Hello, World!");
```

- **System:** This is a predefined class in Java that provides access to system-level resources.
- **out:** It's a static member of the System class and refers to the standard output stream (the console).
- **println():** This method prints the specified text followed by a new line. In this case, it prints the string **"Hello, World!"**.

4. Closing Braces:

- The closing curly braces **}** at the end of the program indicate the end of the main method and the HelloWorld class.

Running the Program:

1. Save the code in a file named HelloWorld.java.
2. Open a terminal or command prompt and navigate to the folder containing the file.
3. Compile the program using the javac compiler:

```
javac HelloWorld.java
```

4. After compiling, run the program using the java command:

```
java HelloWorld
```

Output:

```
Hello, World!
```

5.What is the significance of the PATH and CLASSPATH environment variables in Java?



The PATH environment variable is used by the operating system to locate executable files. When you run a command in the command prompt (like java or javac), the system looks for the executable in the directories listed in the PATH variable.

Significance in Java:

- **Allows Access to Java Commands:** The PATH variable tells the operating system where to find the Java tools such as the Java Virtual Machine (JVM) and Java compiler (javac). This ensures that when you run commands like java or javac from the command line, the operating system knows where to look for them.

The **CLASSPATH** environment variable is used by the Java runtime and compiler to locate Java classes (i.e., compiled .class files). When you run a Java application, the JVM uses the CLASSPATH to search for the required classes and libraries

Significance in Java:

- **Class Location:** It tells the Java runtime and the compiler where to look for user-defined classes and third-party libraries (JAR files). If you have external libraries or custom class files, you need to specify their locations in the CLASSPATH.

6. What are the differences between OpenJDK and Oracle JDK?

Feature	OpenJDK	Oracle JDK
License	Open-source (GPLv2+CE)	Proprietary (Oracle Technology License)
Commercial Support	Community support or third-party vendors	Paid commercial support from Oracle
Updates	Community-driven, less frequent updates	Regular security and bug fixes, long-term support
Performance	Comparable to Oracle JDK	Possible optimizations specific to Oracle JDK
Tools	Lacks Oracle proprietary tools (e.g., JDK Mission Control)	Includes Oracle's proprietary tools (JDK Mission Control, Java Flight Recorder)
Security	Community-driven security updates	Timely, commercial security updates
Distribution	Distributed by multiple vendors	Distributed directly by Oracle

7. Explain how Java programs are compiled and executed.



Compilation:

First, the source '.java' file is passed through the compiler, which then encodes the source code into a Bytecode.

The content of each class contained in the source file is stored in a separate '.class' file.

Execution:

To run, the main class file (the class that contains the method main) is passed to the JVM and then goes through three main stages before the final machine code is executed. These stages are:

These states do include:

1. ClassLoader
2. Bytecode Verifier
3. Just-In-Time Compiler

Stage 1: Class Loader

It is responsible for loading Java bytecode (.class files) into JVM memory when a program starts.

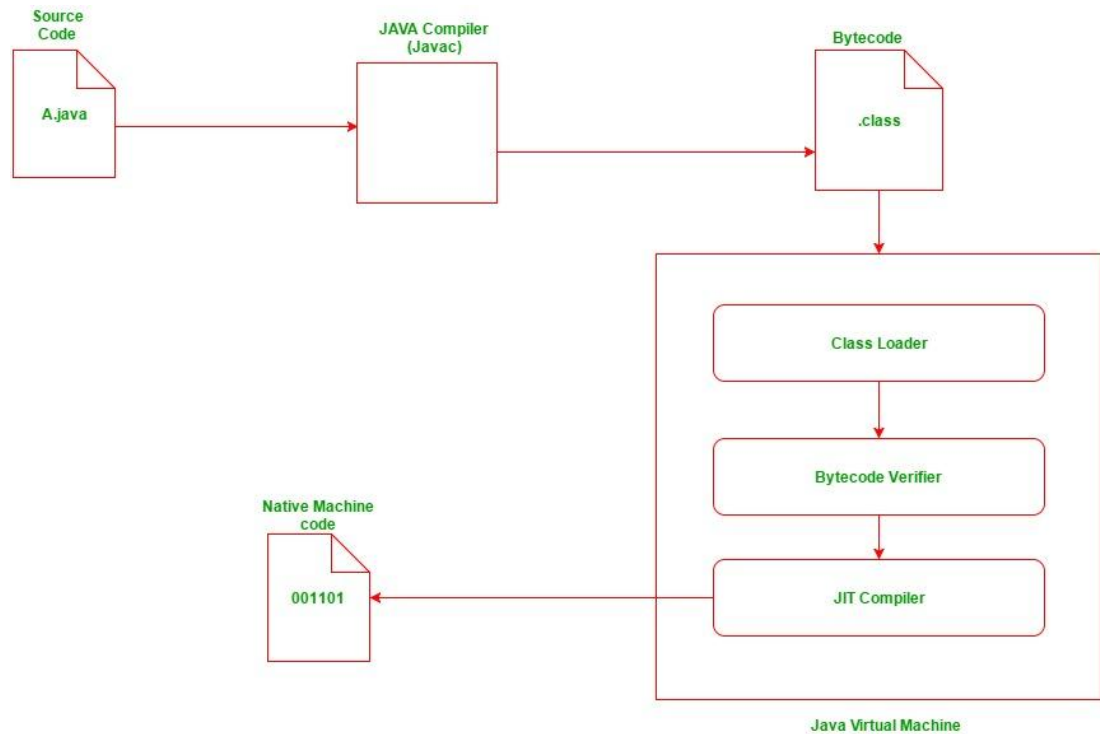
Stage 2: Bytecode Verifier

After the bytecode of a class is loaded by the class loader, it has to be inspected by the bytecode verifier, whose job is to check that the instructions don't perform damaging actions. The following are some of the checks carried out:

- Variables are initialized before they are used.
- Method calls match the types of object references.
- Rules for accessing private data and methods are not violated.
- Local variable accesses fall within the runtime stack.
- The run-time stack does not overflow.
- If any of the above checks fail, the verifier doesn't allow the class to be loaded.

Stage 3: Just-In-Time Compiler

- This is the final stage encountered by the java program, and its job is to convert the loaded bytecode into machine code. When using a JIT compiler, the hardware can execute the native code, as opposed to having the JVM interpret the same sequence of bytecode repeatedly and incurring the penalty of a relatively lengthy translation process. This can lead to performance gains in the execution speed unless methods are executed less frequently.



-
- The process can be well-illustrated by the following diagram given above as follows from which we landed up to the conclusion.

8. What is Just-In-Time (JIT) compilation, and how does it improve Java performance?

JIT (Just in Time) Compiler converts bytecode into native machine code at runtime to improve performance.

Purpose: Speed up the execution process for frequently used code.

Working:

JVM first interprets the bytecode line by line.

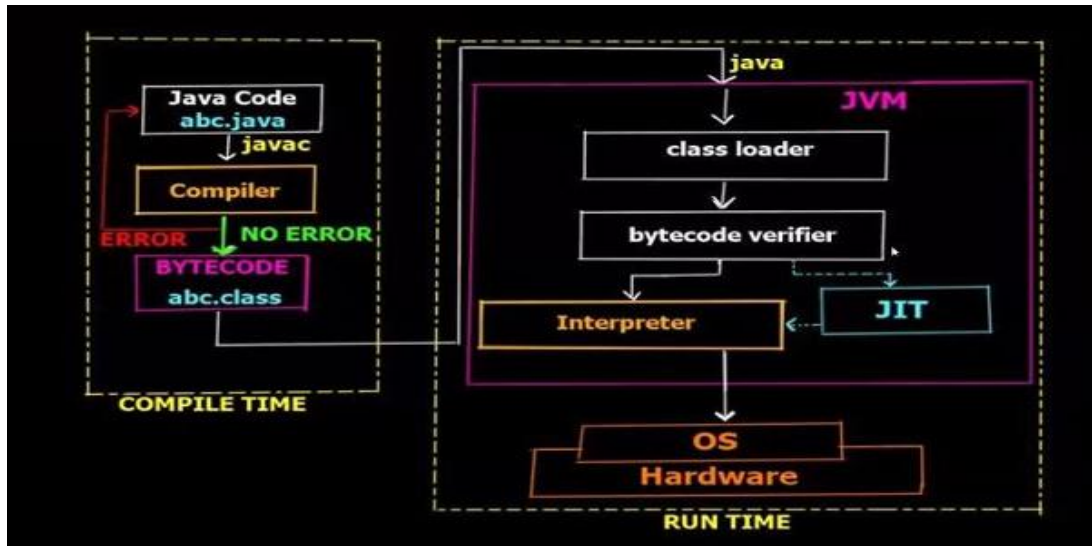
JIT detects hotspot methods (frequently used methods)

Convert those methods into native machine code.

Stores compiled code for future reuse

JIT compilation enhances Java performance by:

- **Reducing interpretation overhead:** Bytecode is compiled into native machine code at runtime, which improves execution speed.
- **Optimizing frequently used code:** Hot spots are compiled and optimized dynamically based on runtime data.
- **Platform-specific optimizations:** JIT allows the JVM to optimize code for the specific hardware and operating system.
- **Adaptive behavior:** JIT continuously optimizes the code as the program runs, based on actual execution patterns.



9. Discuss the role of the Java Virtual Machine (JVM) in program execution.



JVM:

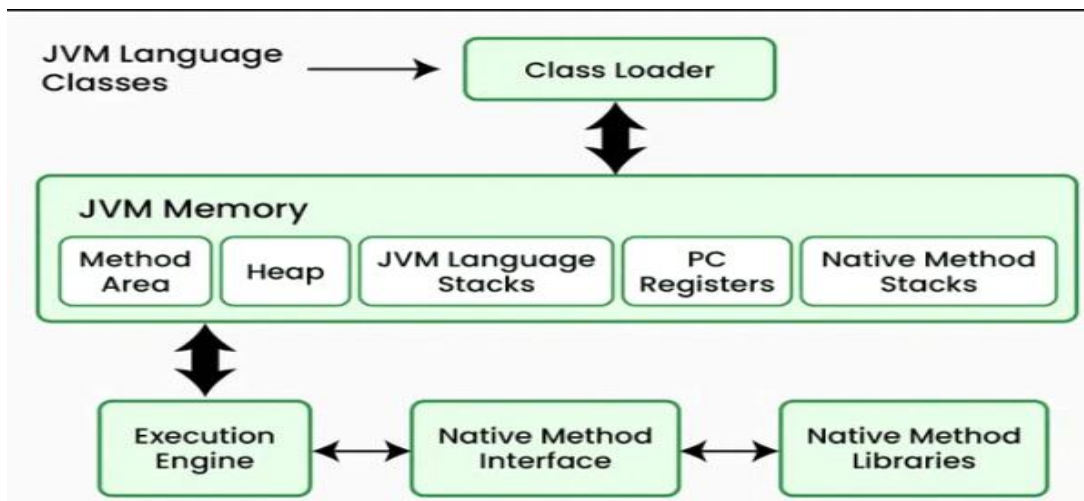
JVM is the core of Java's architecture. It is responsible for loading, verifying, and executing Java bytecode.

It serves as a bridge between Java programs and the underlying operating system.

It is an engine that provides a run-time environment to run the Java applications and it is part of JRE.

It runs java applications by converting bytecode to machine code.

Java uses the combination of both (compiler and interpreter). source code (.java file) is first compiled into byte code and generates a class file (.class file). Then JVM converts the compiled binary byte code into a specific machine language. In the end, JVM is a specification for a software program that executes code and provides the runtime environment for that code.



JVM consists of three main subsystems:

- Class Loader Subsystem
- JVM Memory Areas
- Native Method Interface
- Execution Engine
- Native Method Libraries

The Class Loader subsystem -It is responsible for loading Java bytecode (.class files) into JVM memory when a program starts.

Three Types of Class Loaders:

Bootstrap ClassLoader– Loads core Java classes from rt.jar (like java.lang.Object).

Extension ClassLoader– Loads classes from the ext directory (java.ext.dirs).

Application ClassLoader – Loads application-specific classes from the classpath.

Class Loading Process:

Loading: Reads .class files from disk or network.

Linking:- Verification: Ensures bytecode follows Java standards.

Preparation: Allocates memory for static variables.

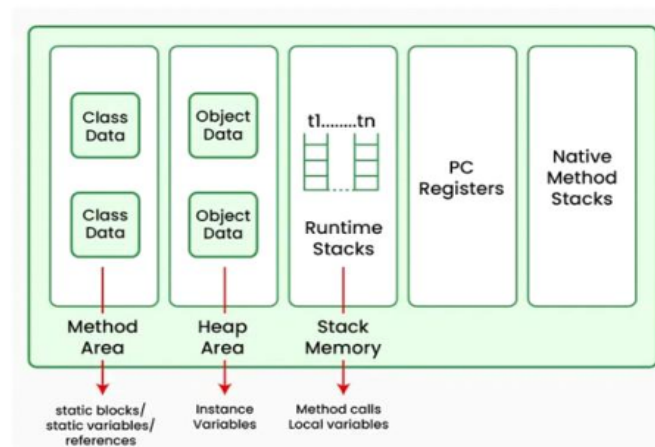
Resolution: Converts symbolic references to actual memory locations.

Initialization: Executes static initializers and assigns values

JVM Memory Areas:

- JVM divides memory into different areas:

Memory Area	Description
Method Area	Stores class metadata, static variables, references and method code.
Heap	Stores objects and instance variables (shared across threads).
Stack	Stores method call frames and local variables (separate for each thread).
PC Register	Holds the address of the currently executing instruction.
Native Method Stack	Manages native (non-Java) method calls.



Execution Engine:

Execution engine executes the “.class” (bytecode).

It reads the byte-code line by line, uses data and information present in various memory area and executes instructions.

It can be classified into three parts:

- **Interpreter:** It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required.
- **Just-In-Time Compiler(JIT):** It is used to increase the efficiency of an interpreter. It compiles the entire bytecode and changes it to native/machine code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re interpretation is not required, thus efficiency is improved
- **Garbage Collector:** It destroys un-referenced objects. Automatically manages memory by reclaiming unused objects.

Java Native Interface (JNI):

It is an interface that interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution. It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.

Java Method Libraries:

These are collections of native libraries required for executing native methods. They include libraries written in languages like C and C++.

