Basics of java

1. what is java.this pdf
2. What is the flow of the program, what is .java and .class file this pdf
3. What is jdk,jvm,jre this pdf
4. What is bytecode,
5. Why java is platform independent and what is wora(write once and run anywhere) javanotespdf
6. What is variables javanotespdf pg 2
7. Types of variables define jnp pg 4; example code this pdf pg 3
8. What are datatypes in java.
9. What are the different types of the datatpes in java jnp pg 4
10. What is type casting types widening casting (small to large int to float) and narrowing cating (large to small from float to int) jnp pg no 6

Java control statements / control flow of java. Jnp pg n 9

1. The braching statements
   1. If statement
      1. simple if condition
      2. If - else conditiom
      3. If-elif-else condition
      4. Nested if condition
   2. switch statement
2. loop statement.
   1. do while loop
   2. While loop
   3. For loop
3. Jump statement
   1. Break statement and the continue and statement oops n -56

Oops concepts

1. class
2. Object oops notses pg 6 and 59 for details
3. Characteristic of object
   1. state
   2. Bsehaviour
   3. Identity
4. Encapsulation
5. Abstraction
6. Inheritance
7. Polymorphism

Java is a high-level, object-oriented, platform-independent programming language designed for the programmer to enable (WORA) “write once, run anywhere” . This means that compiled Java code can run on any platform with a Java Virtual Machine (JVM) without needing to be recompiled. Java code is compiled into bytecode, which is platform-independent and executed by the JVM. Java is widely used for developing web and mobile applications.

**Flow of a Java Program: From .java to Execution**

1. **Write the Java source code**
   * You write your program in a text file with a .java extension.
   * This file contains human-readable Java code (e.g., HelloWorld.java).
2. **Compile the .java file**
   * You use the **Java compiler (javac)** from the JDK to compile the .java file.
   * The compiler checks the syntax and converts your source code into **bytecode**.
   * Bytecode is saved in a .class file (e.g., HelloWorld.class).
   * This .class file is **platform-independent** — it’s not machine code but instructions for the JVM.
3. **Run the .class file using the JVM**
   * You use the **Java Virtual Machine (JVM)** to execute the .class file.
   * The JVM reads the bytecode and translates it into machine code specific to your operating system and hardware.
   * This allows the same .class file to run on any platform with a compatible JVM.
4. **Program executes**
   * The JVM executes the instructions and your program runs, producing output or performing actions as coded.

**In summary:**

.java (source code) --javac compiler--> .class (bytecode) --JVM--> Execution on machine

. BYTECODE: Byte code is an intermediate code generated from the source code by java compiler and it is platform independent.

JAVA DEVELOPMENT KIT (JDK): ¬

* The Java Development Kit (JDK) is a software development environment used for developing Java applications
* It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development.
* *JDK is only for development (it is not needed for running Java programs)*
* *JDK is platform-dependent (different version for windows, Linux, macOS)*

3. JAVA RUNTIME ENVIRONMENT (JRE):

* JRE is used to provide runtime environment for JVM.
* It contains set of libraries +other files that JVM uses at runtime.
* *JRE is only for end-users (not for developers).*
* *JRE is platform-dependent (different versions for different OS)*
* 4. JAVA VIRTUAL MACHINE (JVM): ¬
* JVM is an interpreter that converts a program in Java bytecode (intermediate language) into native machine code and executes it.
* JVM needs to be implemented for each platform because it will differ from platform to platform.

Example code for the types of variable;

class Demo {

int instanceVar = 100; // Instance variable

static int staticVar = 200; // Static variable

void method() {

int localVar = 300; // Local variable

System.out.println("Local: " + localVar);

System.out.println("Instance: " + instanceVar);

System.out.println("Static: " + staticVar);

}

public static void main(String[] args) {

Demo d1 = new Demo();

Demo d2 = new Demo();

d1.method();

d2.instanceVar = 500;

Demo.staticVar = 600;

d1.method();

d2.method();

}

}

Output (to understand difference)

Local: 300

Instance: 100

Static: 200

Local: 300

Instance: 100

Static: 600 <-- changed for all objects

Local: 300

Instance: 500 <-- changed only for d2

Static: 600 <-- same for all objects

**Answer format:**

"Local variables are declared inside methods and exist only during method execution. They don’t get default values, so we must initialize them before use."

"Instance variables are declared inside a class but outside methods. Each object has its own copy, and they get default values if not initialized."

"Static variables are declared with the static keyword inside a class. They belong to the class, not the object, and are shared among all objects."

* 