Core Java Notes

Why you must learn Java

1. Wide Usage (Web-apps, backend,Mobile apps, enterprise software).
2. Object Oriented
3. Rich APIs and Community Support

What is a Programming Language

Giving instructions to a computer

What is an Algorithm

An algorithm is a step-by-step procedure for solving a problem or performing a task

History of Java : Developed by James Gosling at Sun Microsystems (Early 1990s): Originally named 'Oak', later renamed Java in 1995

Java Features :

Robust: Java is robust due to its strong memory management,exception handling, and type-checking mechanisms, which help in preventing system crashes and ensuring reliable performance.

Multithreaded : Multithreading in programming is the ability of a CPU to execute multiple threads concurrently, allowing for more efficient processing and task management.

Architecture Neutra : Java is architecturally neutral because its compiled code (bytecode) can run on any device with a Java Virtual Machine (JVM), regardless of the underlying hardware
Architecture.

Interpreted and High Performance:Java combines high performance with interpretability, as its bytecode is interpreted by the Java Virtual Machine (JVM), which employs Just-In-Time JIT) compilation for efficient and fast execution. Distributed :Java is inherently distributed,designed to facilitate network-based application development and interaction, seamlessly integrating with Internet protocols and remote method invocation.

Object Oriented Programming :

.JDK

• It's a software development kit required to develop Java applications.
• Includes the JRE, an interpreter/loader (Java), a compiler (javac), a doc generator (Javadoc), and other tools needed for Java development.
• Essentially, JDK is a superset of JRE.

JRE

• It's a part of the JDK but can be downloaded separately.
• Provides the libraries, the JVM, and other components to run applications
• Does not have tools and utilities for developers like compilers or debuggers

.JVM

• It's a part of JRE and responsible for executing the bytecode.

• Ensures Java's write-once-run-anywhere capability.
• Not platform-independent: a different JVM is needed for each type of OS.

Variables :Variables are like containers used for storing data values.

Variable Declaration:eg(int a=10) here a is variable

Data Types:
Data types in Java specify the size and type of values that a variable can store.

Primitive Data Types:
These are the basic building blocks for data manipulation and are predefined by the language. They store values directly in memory. Java has eight primitive data types:

- **byte:** 8-bit signed integer.
- **short:** 16-bit signed integer.
- **int:** 32-bit signed integer.
- **long:** 64-bit signed integer.
- **float:** 32-bit single-precision floating-point number.
- **double:** 64-bit double-precision floating-point number.
- **boolean:** Represents true or false values.
- **char:** 16-bit Unicode character.

2. Non-Primitive (Reference) Data Types:
These are more complex types that store references to memory locations where data is stored. They include:

- **Classes:** User-defined types that encapsulate data and methods.
- **Interfaces:** Define contracts for classes to implement.
- **Arrays:** Ordered collections of elements of the same type.
- **Strings:** Sequences of characters.

Java Identifier Rules: The only allowed characters for identifiers are all alphanumeric characters([A-Z],[a-z],[0-9]), '$' (dollar sign) and '_' (underscore).

2. Can't use keywords or reserved words

3. Identifiers should not start with digits([0-9]).

4. Java identifiers are case-sensitive.

5. There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.

Keywords: Keywords are the reserved words in java, which as special meaning or purpose of those words.(eg:for,if,while etc).

Operators : operators are special symbols that perform operations on variables or values.

Types of Operators in Java

1. **Arithmetic Operators:** are used to perform simple arithmetic operations on primitive and non-primitive data types.

- * : Multiplication
- / : Division
- % : Modulo
- + : Addition
- - : Subtraction

2. **Relational Operators**:Relational operators compare values and return Boolean results:

- `==` , Equal to.
- `!=` , Not equal to.
- `<` , Less than.
- `<=` , Less than or equal to.
- `>` , Greater than.
- `>=` , Greater than or equal to.

3. **Logical Operators:**Conditional operators are:

- **&&, Logical AND:** returns true when both conditions are true.
- **||, Logical OR:** returns true if at least one condition is true.
- **!, Logical NOT:** returns true when a condition is false and vice-versa

4. **Ternary Operator**:condition **?** if true **:** if false

Looping :in programming languages is a feature that facilitates the execution of a set of instructions repeatedly while some condition evaluates to true

for loop

The for loop is used when we know the number of iterations (we know how many times we want to repeat a task). The for statement includes the initialization, condition, and increment/decrement in one line.

Syntax:

```
for (initialization; condition; increment/decrement) {

// code to be executed

}
```

eg:import java.io.*;

```
class Geeks {

   public static void main(String[] args){

      for (int i = 0; i <= 10; i++) {  System.out.print(i + " ");  }  }  }
```

while Loop

A while loop is used when we want to check the condition before executing the loop body.

Syntax:

```
while (condition) {
// code to be executed
}
```

eg:import java.io.*;

```
class Geeks {
    public static void main(String[] args)
    {
        int i = 0;
        while (i <= 10) {
            System.out.print(i + " ");
            I++;
            }
    }
}
```

do-while Loop

The do-while loop ensures that the code block executes **at least once** before checking the condition.

Syntax:

```
do {
// code to be executed
} while (condition);
```

eg:class Geeks {

```
    public static void main(String[] args)
    {
        int i = 0;
        do {
            System.out.print(i + " ");
            i++;
        } while (i <= 10);
    }
}
```

Conditional Loops:

`if` Statement: The `if` statement executes a block of code only if a specified condition is true.

syntax:

```
if (condition) {
   // Code to be executed if the condition is true
}
```

eg:if (age >= 18){

```
   System.out.print(“you can vote”)
}
```

if-else Statement

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't.

Syntax:

```
        if(condition){
        // Executes this block if
        // condition is true
```

```
        }else{
        // Executes this block if
        // condition is false
        }
```

```
eg:import java.util.*;
class Geeks {
    public static void main(String args[])
    {
        int i = 10;


        if (i < 15)
            System.out.println("i is smaller than 15");
        else
            System.out.println("i is greater than 15");
    }
}
```

Switch Case

The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

Syntax:

```
        switch (expression) {
        case value1:
        // code to be executed if expression == value1
        break;
        case value2:
        // code to be executed if expression == value2
```

```
        break;
        // more cases...
        default:
        // code to be executed if no cases match
        }
```

eg:

```
import java.io.*;
class Geeks {
    public static void main(String[] args)
    {       int num = 20;
        switch (num) {
        case 5:
            System.out.println("It is 5");
            break;
        case 10:
            System.out.println("It is 10");
            break;
        case 15:
            System.out.println("It is 15");
            break;
        case 20:
            System.out.println("It is 20");
            break;
        default:
            System.out.println("Not present"); } } }
```