Java : Getting Started

LUMINAR
TECHNOLAB

JDK | JRE | JVM

JDK:-

Java Development Kit (in short JDK) is Kit which provides the environment to Develop and execute(run) the Java program.

JDK is a kit(or package) which includes two things

i) Development Tools to provide an environment to develop your java programs-
   i) an interpreter/loader (java),
   ii) a compiler (javac),
   iii) an archiver (jar),
   iv) a documentation generator (javadoc)
   v) and other tools needed in Java development.)

ii) JRE (to execute your java program).

JDK is only used by Java Developers.

For eg. You(as Java Developer) are developing an accounting application on your machine, so what do you going to need into your machine to develop and run this desktop app? You are going to need J-D-K for that purpose for this you just need to go to official website of oracle to download the latest version of JDK into your machine.

JRE:-

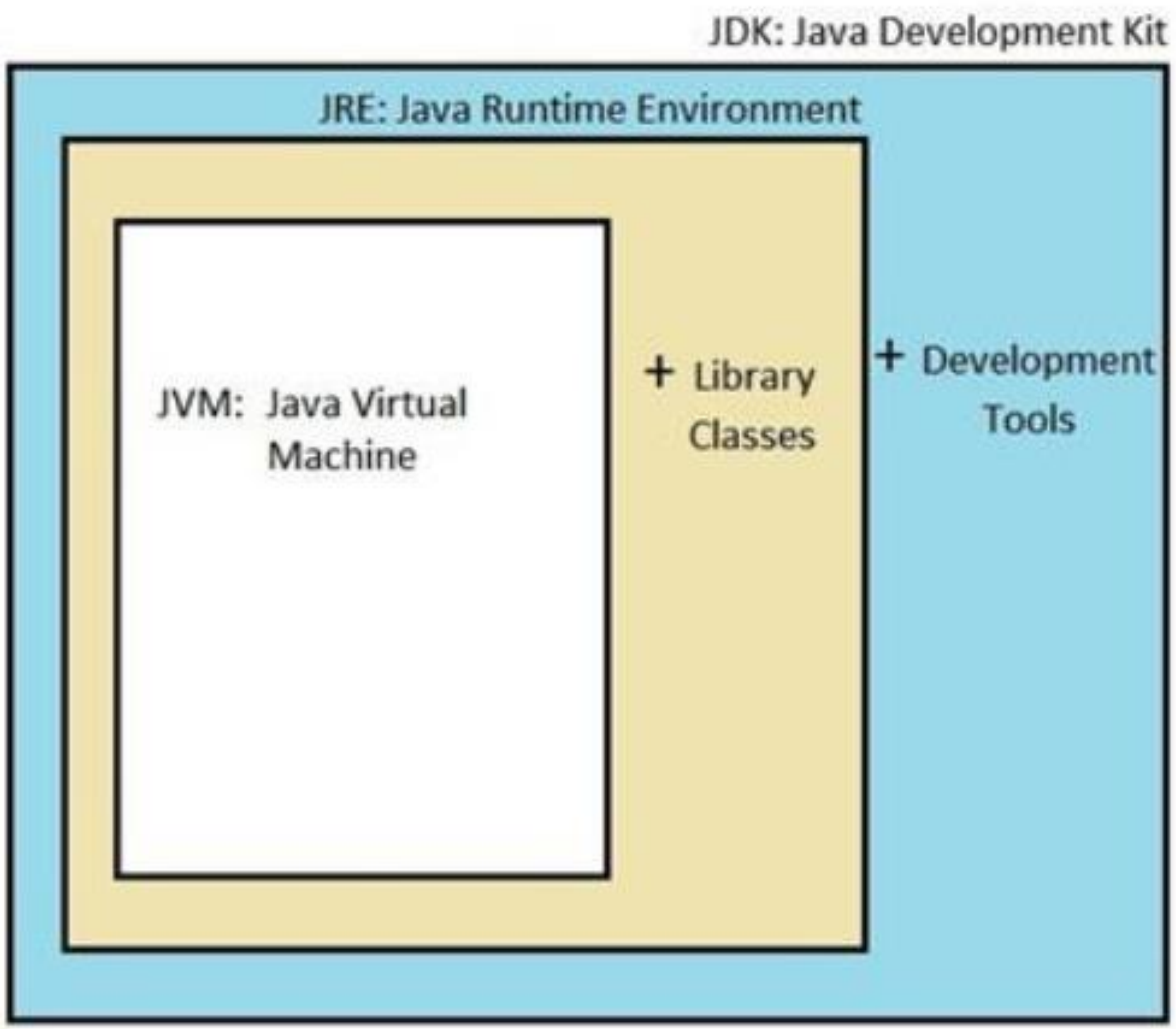
Java Runtime Environment (to say JRE) which provides environment to only run(not develop) the java program(or application)onto your machine.

For eg(continuing with the same example) after developing your accounting application , you want to run this application into your client's machine . Now in this case your client only need to run your application into his/her machine so your client should install JRE in-order to run your application into his machine. Hence, JRE is only used by them who only wants to run the Java Programs i.e. end users of your system.

JVM:-

Java Virtual machine(JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both.

- Whatever java program you run using JRE or JDK goes into JVM .
- JVM is responsible to execute the java program line by line hence it is also known as interpreter.
- Hence you don’t need to install JVM separately into your machine because it is inbuilt into your JDK or JRE installation package.


JDK: Java Development Kit
JRE: Java Runtime Environment
JVM: Java Virtual Machine
+ Library Classes
+ Development Tools

JDK = JRE + Development Tools

JRE = JVM + Library Classes

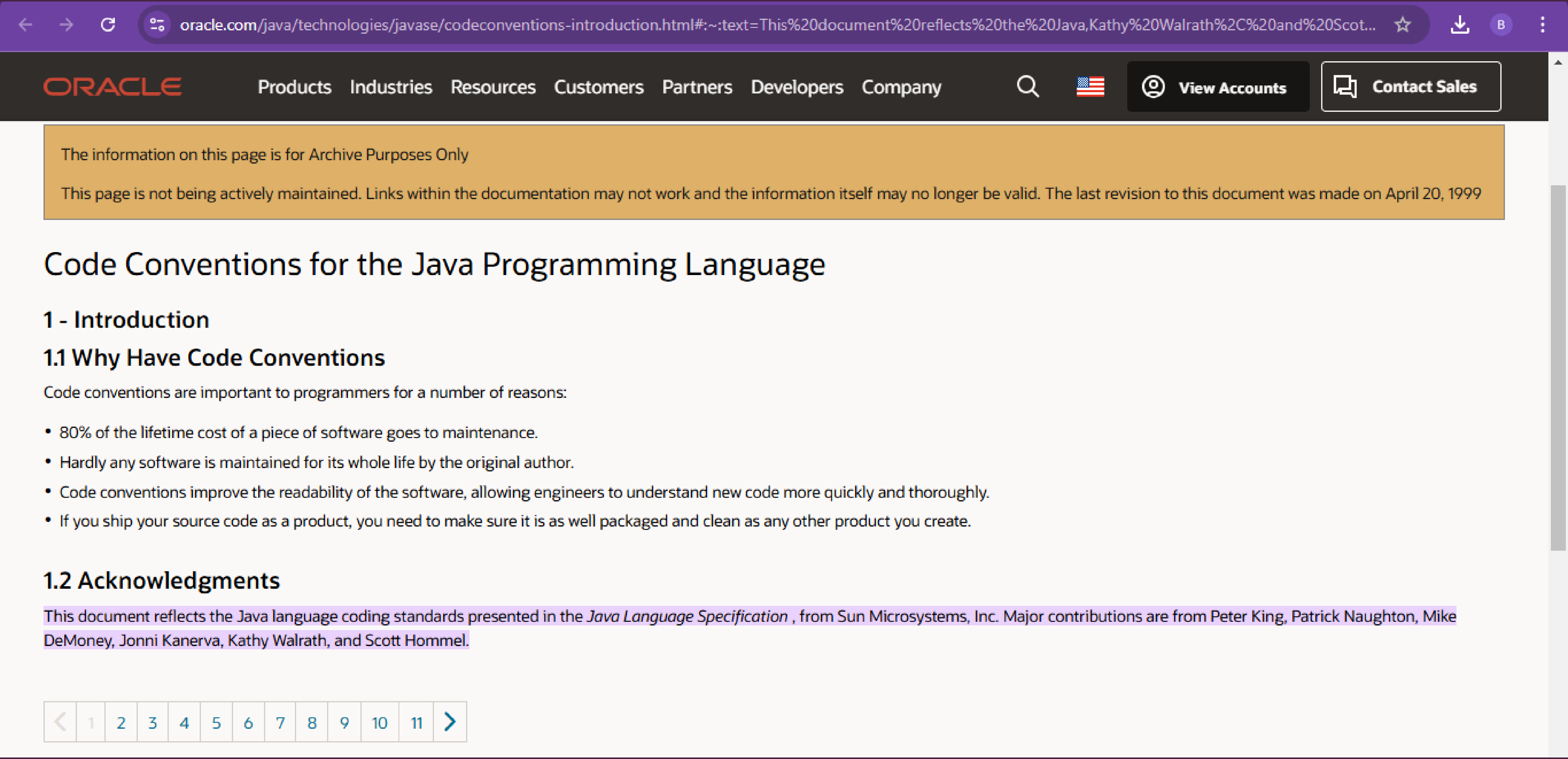
How to write a java program

Note :

Read your coding standards for a better understanding of writing Java code

Java Naming Conventions/ Java Coding Standards/Java coding best practices/Java clean code/ Java Code Conventions…

First and foremost, before writing any code you should specify a set of naming conventions for your Java project, such as how to name classes and interfaces, how to name methods, how to name variables, how to name constants, etc. These conventions must be obeyed by all programmers in your team.


oracle.com/java/technologies/javase/codeconventions-introduction.html#:~:text=This%20document%20reflects%20the%20Java,Kathy%20Walrath%2C%20and%20Scot...
ORACLE
Products
Industries
Resources
Customers
Partners
Developers
Company
View Accounts
Contact Sales
1 2 3 4 5 6 7 8 9 10 11

The information on this page is for Archive Purposes Only

This page is not being actively maintained. Links within the documentation may not work and the information itself may no longer be valid. The last revision to this document was made on April 20, 1999

Code Conventions for the Java Programming Language

1 - Introduction

1.1 Why Have Code Conventions

Code conventions are important to programmers for a number of reasons:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

1.2 Acknowledgments

This document reflects the Java language coding standards presented in the *Java Language Specification* , from Sun Microsystems, Inc. Major contributions are from Peter King, Patrick Naughton, Mike DeMoney, Jonni Kanerva, Kathy Walrath, and Scott Hommel.

Step 1: Open any IDE (Integrated Development Environment)
(e.g., Notepad, TextEdit, Notepad++, Eclipse, IntelliJ IDEA, etc.).

Step 2: Write your Java code
(following Java coding conventions).

Step 3: Save the file with the extension “.java” (e.g., MyFirstProgram.java).
(It’s better to create a folder first and save the file in that folder.)

Step 4: Open the Command Prompt/Terminal.

Step 5: Compile the program (e.g., javac MyFirstProgram.java).

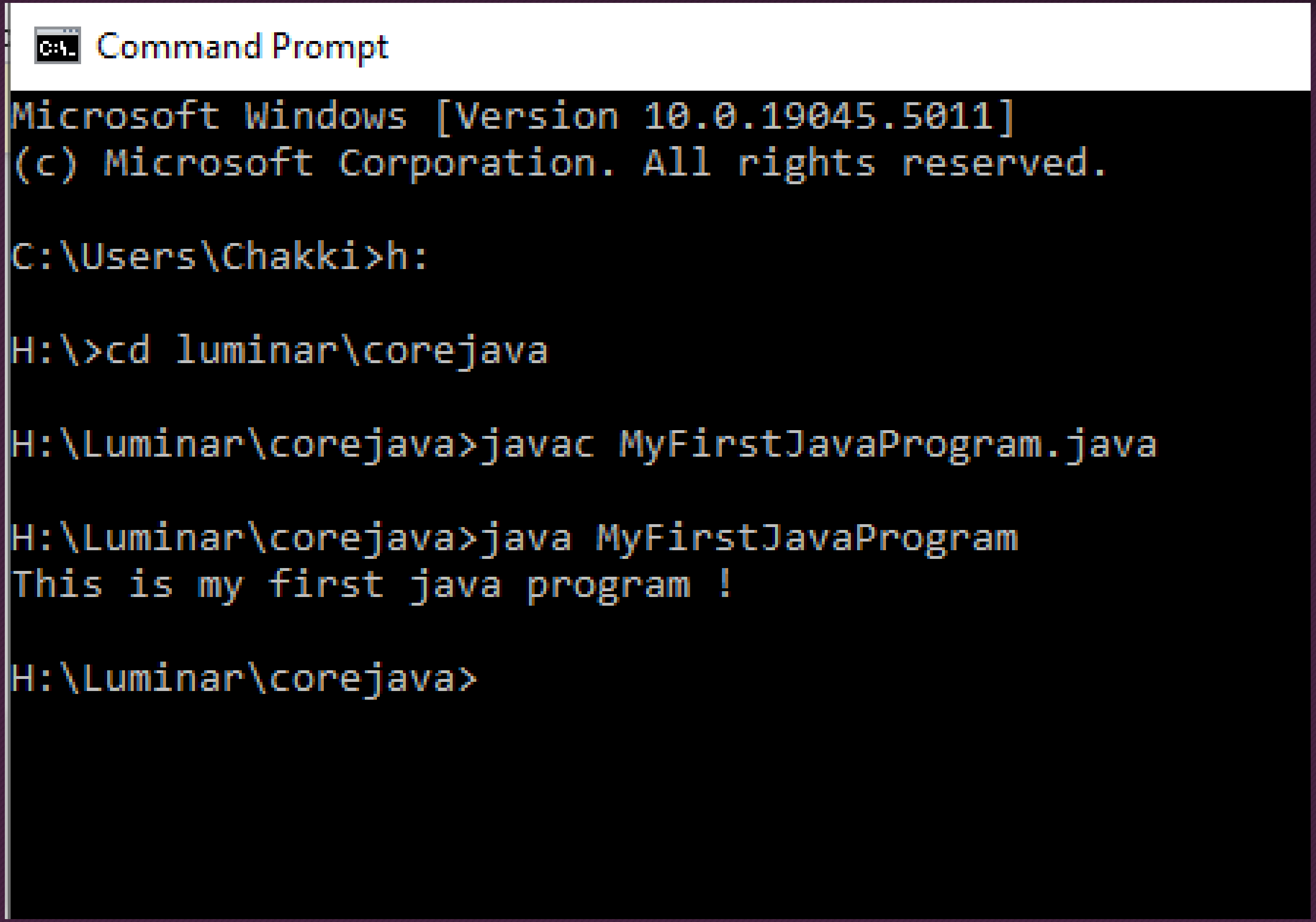
Step 6: Execute/run the program (e.g., java MyFirstProgram).

Java first program

I have created a folder named 'corejava' in the Luminar folder on the H: drive.

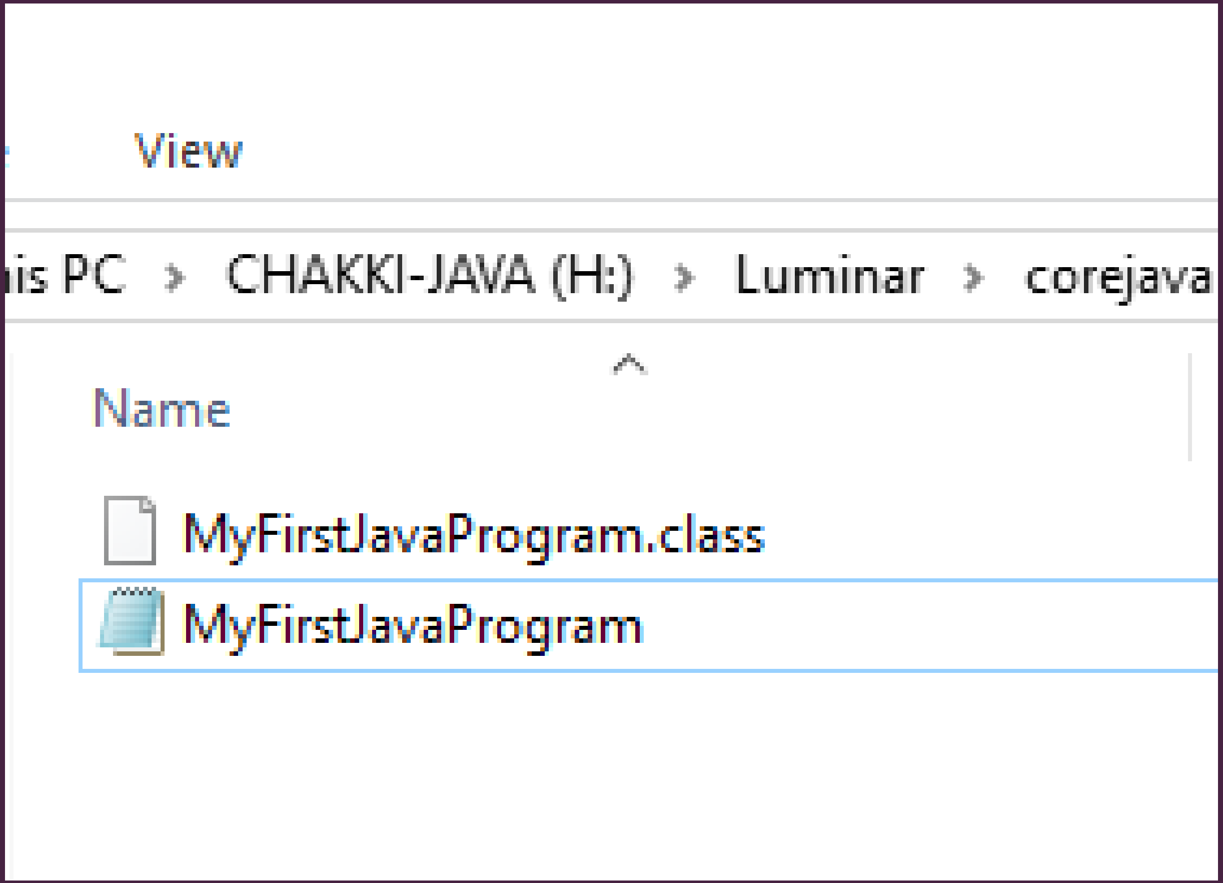
```
class MyFirstJavaProgram {
   public static void main(String args[]){
      System.out.println("This is my first java program”);
   }
}
```

Compile and run a java program



```
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Chakki>h:

H:\>cd luminar\corejava

H:\Luminar\corejava>javac MyFirstJavaProgram.java

H:\Luminar\corejava>java MyFirstJavaProgram
This is my first java program !

H:\Luminar\corejava>
```

After successful compilation, the compiler creates a .class file with the same name as your program



MyFirstJavaProgram.class
MyFirstJavaProgram

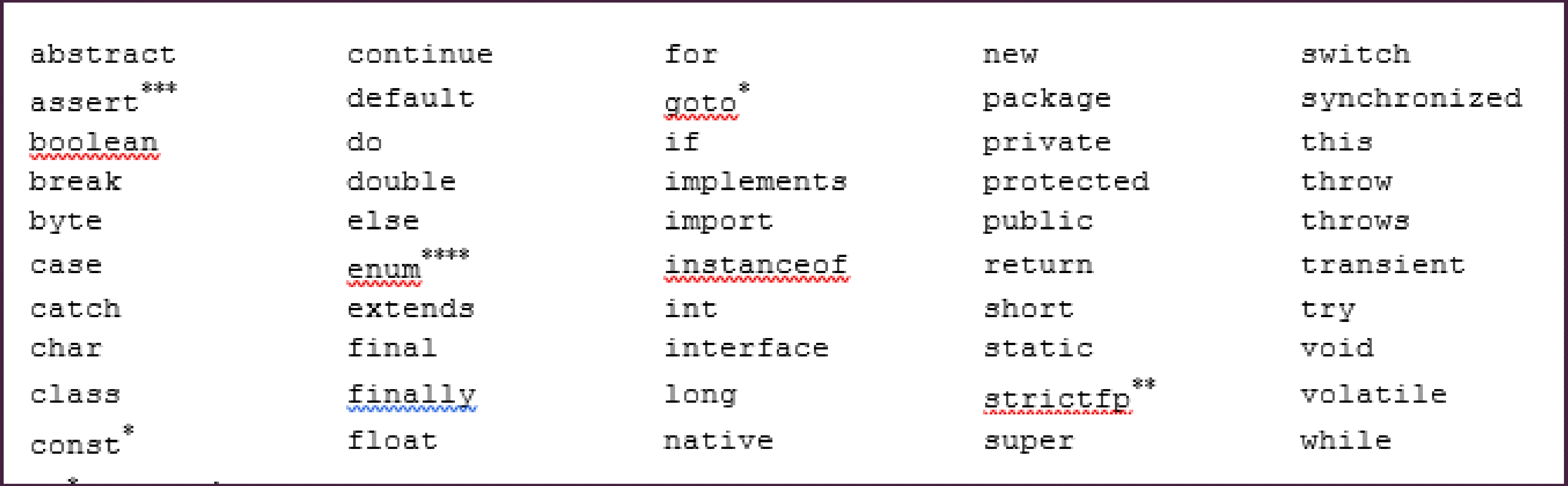
Java Comments

- Comments can be used to explain Java code.
- Make it more readable.
- It can also be used to prevent execution when testing alternative code.

Ref: Java Code Conventions September 12, 1997

Page No:6

Java Key words



abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

*	not used
**	added in 1.2
***	added in 1.4
****	added in 5.0

Java Datatypes & Variables

Examples

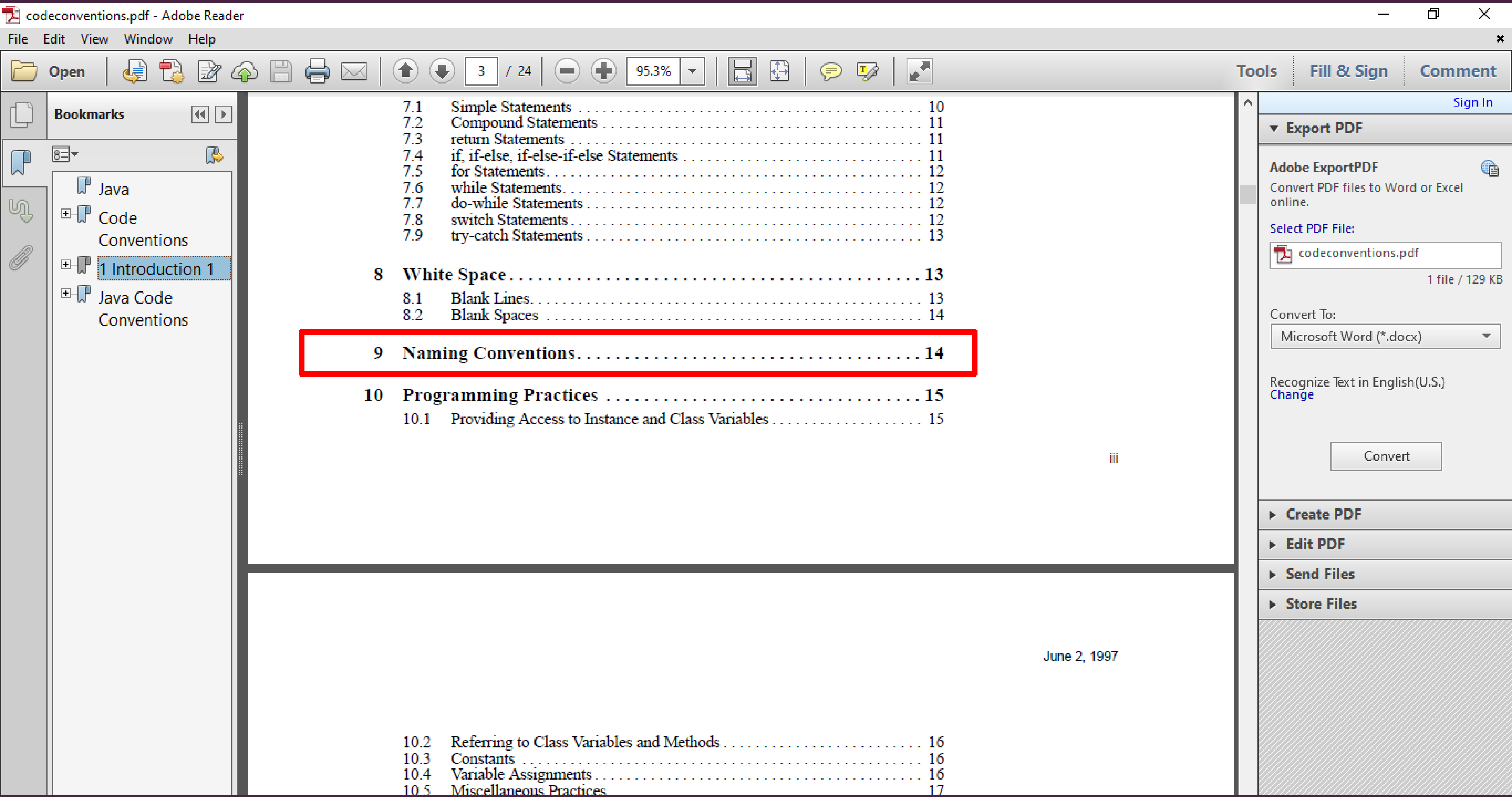
```
int counter;

float balanceAmount;

String studentName;
```

🟦 Datatype

🟨 Variable

Java Variables

- When we want to store any information, we store it in an address of the computer. Instead of remembering the complex address where we have stored our information, we name that address. The naming of an address is known as variable.
- Variable is the name of memory location.
- It means when we declare a variable some part of memory is reserved.



Convert PDF files to Word or Excel online.

Recognize Text in English(U.S.)

7.1 Simple Statements 10
7.2 Compound Statements 11
7.3 return Statements 11
7.4 if, if-else, if-else-if-else Statements 11
7.5 for Statements 12
7.6 while Statements 12
7.7 do-while Statements 12
7.8 switch Statements 12
7.9 try-catch Statements 13

8 White Space 13
8.1 Blank Lines 13
8.2 Blank Spaces 14

9 Naming Conventions 14

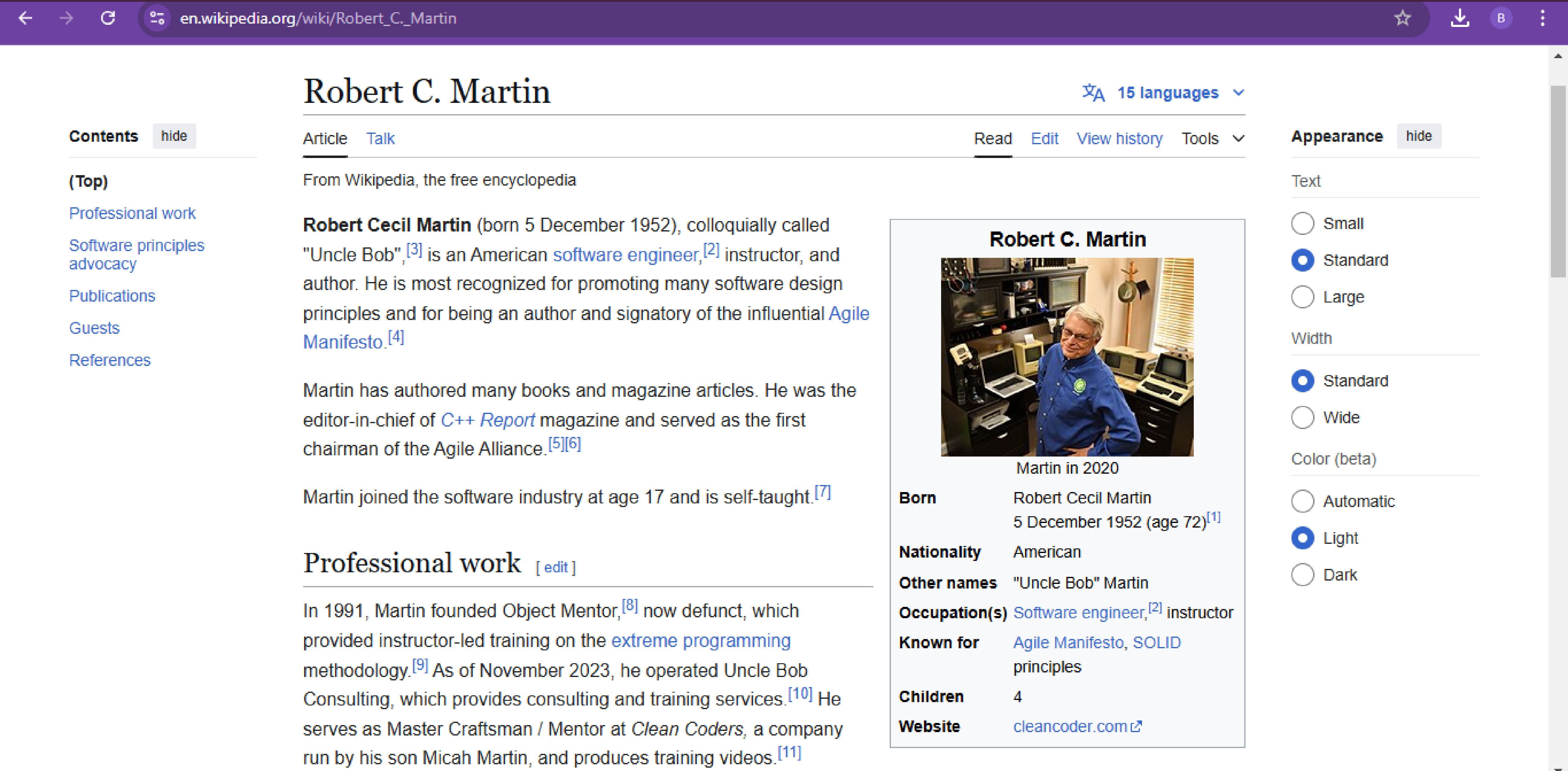
10 Programming Practices 15
10.1 Providing Access to Instance and Class Variables 15

iii

June 2, 1997

10.2 Referring to Class Variables and Methods 16
10.3 Constants 16
10.4 Variable Assignments 16
10.5 Miscellaneous Practices 17

According to Robert Martin (the author of *Clean Code*), an identifier (a class, a method, and a variable) should have the following characteristics:



en.wikipedia.org/wiki/Robert_C._Martin

Robert C. Martin

Contents hide
(Top)
Professional work
Software principles advocacy
Publications
Guests
References

From Wikipedia, the free encyclopedia

Robert Cecil Martin (born 5 December 1952), colloquially called "Uncle Bob",[3] is an American software engineer,[2] instructor, and author. He is most recognized for promoting many software design principles and for being an author and signatory of the influential Agile Manifesto.[4]

Martin has authored many books and magazine articles. He was the editor-in-chief of *C++ Report* magazine and served as the first chairman of the Agile Alliance.[5][6]

Martin joined the software industry at age 17 and is self-taught.[7]

Professional work [edit]

In 1991, Martin founded Object Mentor,[8] now defunct, which provided instructor-led training on the extreme programming methodology.[9] As of November 2023, he operated Uncle Bob Consulting, which provides consulting and training services.[10] He serves as Master Craftsman / Mentor at *Clean Coders*, a company run by his son Micah Martin, and produces training videos.[11]

Robert C. Martin

Martin in 2020

Born	Robert Cecil Martin 5 December 1952 (age 72)[1]
Nationality	American
Other names	"Uncle Bob" Martin
Occupation(s)	Software engineer,[2] instructor
Known for	Agile Manifesto, SOLID principles
Children	4
Website	cleancoder.com

Appearance hide

Text: Small [✓ Standard] Large

Width: [✓ Standard] Wide

Color (beta): Automatic [✓ Light] Dark

Self-explanatory:

A name must reveal its intention so everyone can understand and change the code easily.

For example, the names *d* or *str* do not reveal anything; however the names *daysToExpire* or *inputText* do reveal their intention clearly.

Note that if a name requires a comment to describe itself, then the name is not self-explanatory.

Meaningful distinctions:

If names must be different, then they should also mean something different.

For example, the names *a1* and *a2* are meaningless distinction; and the names *source* and *destination* are meaningful distinction.

Pronounceable:

Names should be pronounceable as naturally as spoken language because we are humans - very good at words.

For example, which name can you pronounce and remember easily: *genStamp* or *generationTimestamp*?

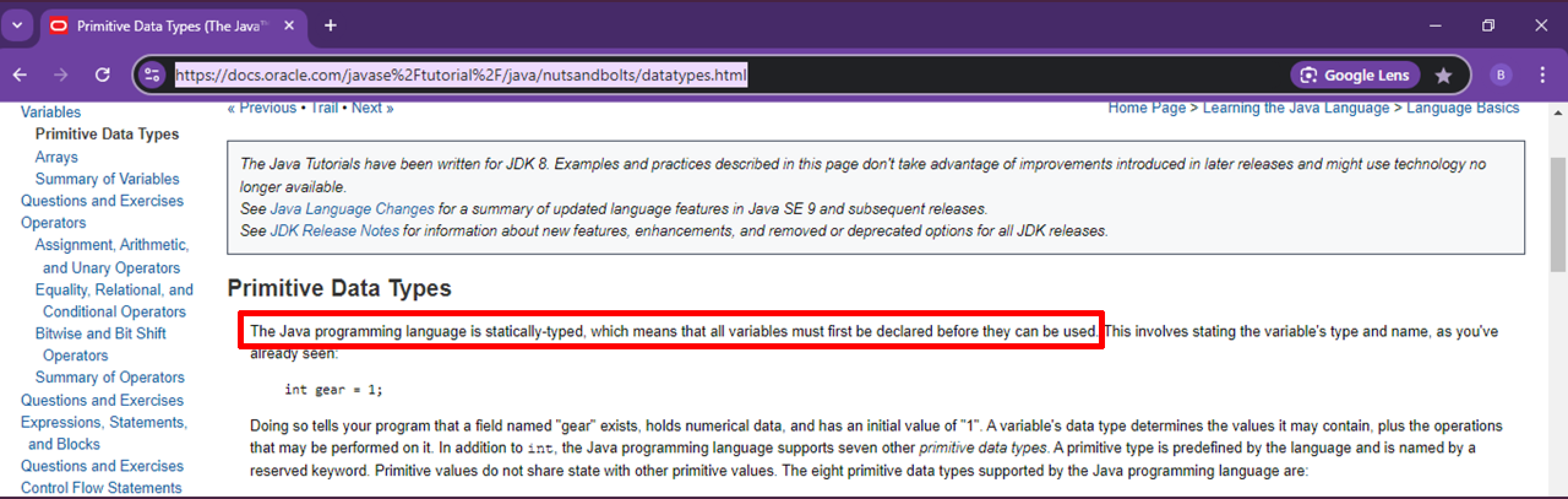
Java Variable naming - coding best practices

1. Variable names should be nouns, starting with a lowercase letter.
2. Using camelCase notation for names (internal words start with capital letters)

For example: *number, counter, birthday, gender*, etc.

1. Common names for :
   - temporary variables are
     - *Integer values : i , j* and *k , m* and *n*
     - *characters c, d* and *e*

https://docs.oracle.com/javase%2Ftutorial%2F/java/nutsandbolts/datatypes.html


Primitive Data Types (The Java™
https://docs.oracle.com/javase%2Ftutorial%2F/java/nutsandbolts/datatypes.html
Google Lens

Variables
Primitive Data Types
Arrays
Summary of Variables
Questions and Exercises
Operators
Assignment, Arithmetic, and Unary Operators
Equality, Relational, and Conditional Operators
Bitwise and Bit Shift Operators
Summary of Operators
Questions and Exercises
Expressions, Statements, and Blocks
Questions and Exercises
Control Flow Statements

« Previous • Trail • Next »

Home Page > Learning the Java Language > Language Basics

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.
See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Primitive Data Types

The Java programming language is statically-typed, which means that all variables must first be declared before they can be used. This involves stating the variable's type and name, as you've already seen:

```
int gear = 1;
```

Doing so tells your program that a field named "gear" exists, holds numerical data, and has an initial value of "1". A variable's data type determines the values it may contain, plus the operations that may be performed on it. In addition to `int`, the Java programming language supports seven other *primitive data types*. A primitive type is predefined by the language and is named by a reserved keyword. Primitive values do not share state with other primitive values. The eight primitive data types supported by the Java programming language are:

Java Datatypes

- As we discussed when a variable is declared some part of memory is reserved.
- But how much memory will be reserved?
- It depends upon the data type of the variable. i.e. how much memory will be reserved and which type of data can be stored in reserved memory is depends upon data type of the variable.

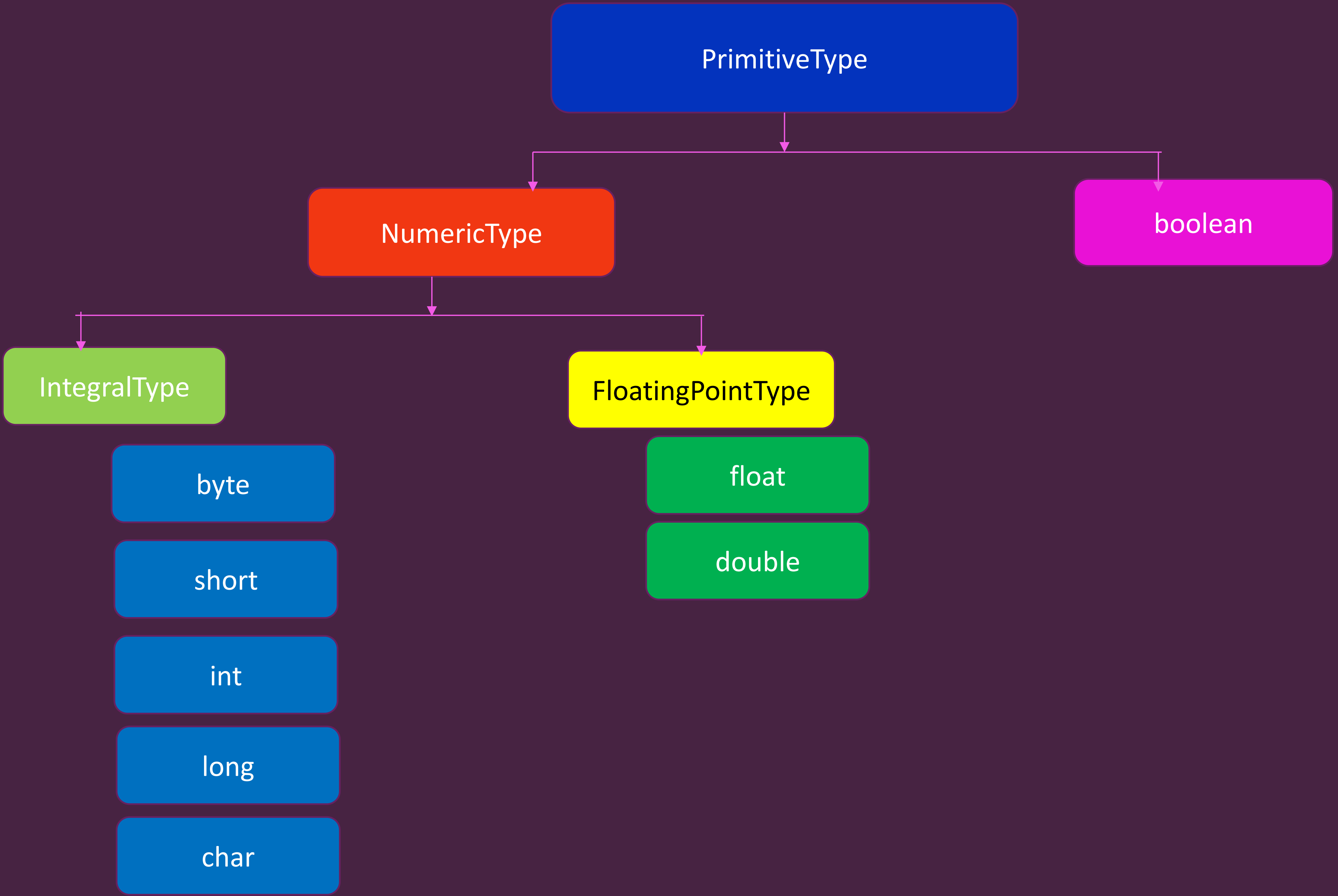
Java has two categories of data types

Primitive Data Types and Non-Primitive Type (Reference Data Types)**

**Non-Primitive Types" is a broader way of categorizing them. It is still accurate to call them "Reference Data Types" because they reference memory locations that hold the actual data.

Primitive Data Types

Primitive data types represent simple values like numbers, characters, or logical values. They are the most basic data types and are built into the Java language. There are 8 primitive data types in Java.


PrimitiveType
NumericType
boolean
IntegralType
FloatingPointType
byte
short
int
long
char
float
double

byte

- **Size**: 8 bits (1 byte)
- **Range**: -128 to 127 (inclusive)(Inclusive means that both the lower bound (-128) and the upper bound (127) are part of the range.)
- **Description**:
  - ✓ The byte data type is an 8-bit signed integer. (signed-> means that the number can be both positive and negative. One bit is used to indicate the sign of the number, while the remaining bits represent the magnitude (the absolute value).It is useful for saving memory, especially in large arrays or when memory is a concern.
  - ✓ It is smaller than the short, int, and long types, and it can be used when you know that the values you need to store will be within the range of -128 to 127.
  - ✓ Example usage:

```
byte b = 100;
```

short

- **Size**:16 bits (2 bytes)
- **Range**: -32,768 to 32,767 (inclusive)
- **Description**:
  - ✓ The short data type is a 16-bit signed integer. It is used to save memory when you are certain the values will be in the short range, but more memory-efficient than int if you're dealing with large arrays.
  - ✓ Example usage:

```
short s = 1500;
```

int

- **Size**:32 bits (4 bytes)
- **Range**: -2^31 to 2^31 - 1 (i.e., -2,147,483,648 to 2,147,483,647)
- **Description**:
  - ✓ The int data type is a 32-bit signed integer. It is one of the most commonly used data types in Java and is the *default type for integer values*.
  - ✓ Example usage:

```
int minimumBalance = 50000;
```

long

- **Size**:64 bits (8 bytes)

- **Range**: -2^63 to 2^63 - 1 (ie. -2^{63} to $+2^{63}$)
  Minimum value : -9,223,372,036,854,775,808
  Maximum value : + 9,223,372,036,854,775,807

- **Description**:

  - ✓ The long data type is a 64-bit signed integer. It is used when you need a larger range than int provides..

  - ✓ Example usage:

```
long cableLength = 100000L;
```

float

- **Size**:32 bits (4 bytes)

- **Range**: Varies, but approximately ±1.4 × 10^-45 to ±3.4 × 10^38
  (ie ± 1.4 x 10 $^{-45}$ to ±3.4 × 10 $^{-38}$
  Minimum value: ±0.0000000000000000000000000000000000000000000000000014
  (approximately 1.4×10−451.4×10 −45)
  Maximum value: ±340,000,000,000,000,000,000,000,000,000,000,000,000 (approximately 3.4×10383.4×10 38)
- **Description**:

  - ✓ The float data type is a 32-bit IEEE 754 floating point. It is used for single-precision floating-point values. The float data type is mainly used for saving memory in large arrays of floating-point numbers.

  - ✓ You need to append an f or F to the number to define it as a float (e.g., 3.14f).
  - ✓ Example usage:

```
float marks = 3.14f;
```

IEEE 754 Standard:

The representation of floating-point numbers in Java, as well as in most modern programming languages and computer systems, follows the IEEE 754 standard for floating-point arithmetic.

According to this standard, a floating-point number is divided into three parts:

Sign bit: 1 bit to indicate whether the number is positive or negative.

Exponent: 8 bits to store the exponent (how much the number should be multiplied or divided by a power of 2).

Mantissa (also called the fraction): 23 bits to store the significant digits of the number.

The term "single-precision" comes from the fact that this format provides less precision and range compared to double-precision (64-bit) floating-point values.

double

- **Size**:64 bits (8 bytes)
- **Range:** Varies, but approximately ±4.9 × 10^-324 to ±1.8 × 10^308
  (ie. $\pm 4.9 \times 10^{-324}$ to $\pm 1.8 \times 10^{308}$
- **Description**:
  - ✓ The double data type is a 64-bit IEEE 754 floating point. It is used for double-precision floating-point values, offering a larger precision than float.
  - ✓ *It is the default type for decimal values in Java.*
  - ✓ Example usage:

```
double amount = 3.14159265359;
```

float requires f or F because Java treats decimal numbers as double by default.

long requires L or l because Java treats integer literals as int by default.

short does not require a suffix because Java can automatically interpret smaller integer values and does not need a special indication like L or f. However, for values that exceed the range of short, you would need to cast them explicitly.

char

- **Size**:16 bits (2 bytes)
- **Range:** 0 to 65,535 (unsigned)
- **Description**:
  - ✓ The char data type represents a single 16-bit Unicode character. It is used to store individual characters like letters, digits, and symbols.
  - ✓ Example usage:

```
char c = 'A';
```

16 bits (2 bytes):A 16-bit value can represent 65536 different values This means that a 16-bit Unicode character can have one of 65,536 possible values.

Unicode: Unicode is an international character encoding standard designed to include characters from all the writing systems in the world.It assigns a unique number, called a code point, to each character.

16-bit Unicode Encoding (UTF-16): Unicode characters can be encoded in several different ways. One of the most common encodings is UTF-16, which uses 16 bits to represent characters.

boolean

- **Size**:Not precisely defined; typically 1 bit.
- **Range:** true or false
- **Description**:
  - ✓ The boolean data type can only have two possible values: true or false. It is used for simple flags that track true/false conditions.
  - ✓ Example usage:

```
boolean flag = true;
```

Summary Table of Primitive Data Types

Data Type	Size	Range	Default Value
byte	8 bits	-128 to 127	0
short	16 bits	-32,768 to 32,767	0
int	32 bits	-2,147,483,648 to 2,147,483,647	0
long	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
float	32 bits	$\pm 1.4 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	0.0f
double	64 bits	$\pm 4.9 \times 10^{-324}$ to $\pm 1.8 \times 10^{308}$	0.0
char	16 bits	0 to 65,535	'\u0000'
boolean	1 bit	true or false	false

Use Cases and Considerations:

- **Choosing the Right Type**: Select a data type based on the range of values you expect and memory constraints. For example, use byte if memory is critical and the range of values is small, and use long if the range of values is large.

- **Memory Efficiency**: Smaller data types like byte and short use less memory, which is useful in large data sets or embedded systems.

- **Precision**: For floating-point numbers, use double when precision is important, as it offers higher precision than float.

Each primitive type serves a specific purpose in terms of the size, range, and precision of values they can hold. Properly choosing the appropriate data type is essential for writing efficient and maintainable Java code.

String

- In addition to the eight primitive data types listed above, the Java programming language also provides special support for character strings via the java.lang.String class.

- Enclosing your character string within double quotes will automatically create a new String object;

for example, String s = "this is a string";.

- String objects are immutable, which means that once created, their values cannot be changed.

- The String class is not technically a primitive data type, but considering the special support given to it by the language, we'll probably tend to think of it as such.

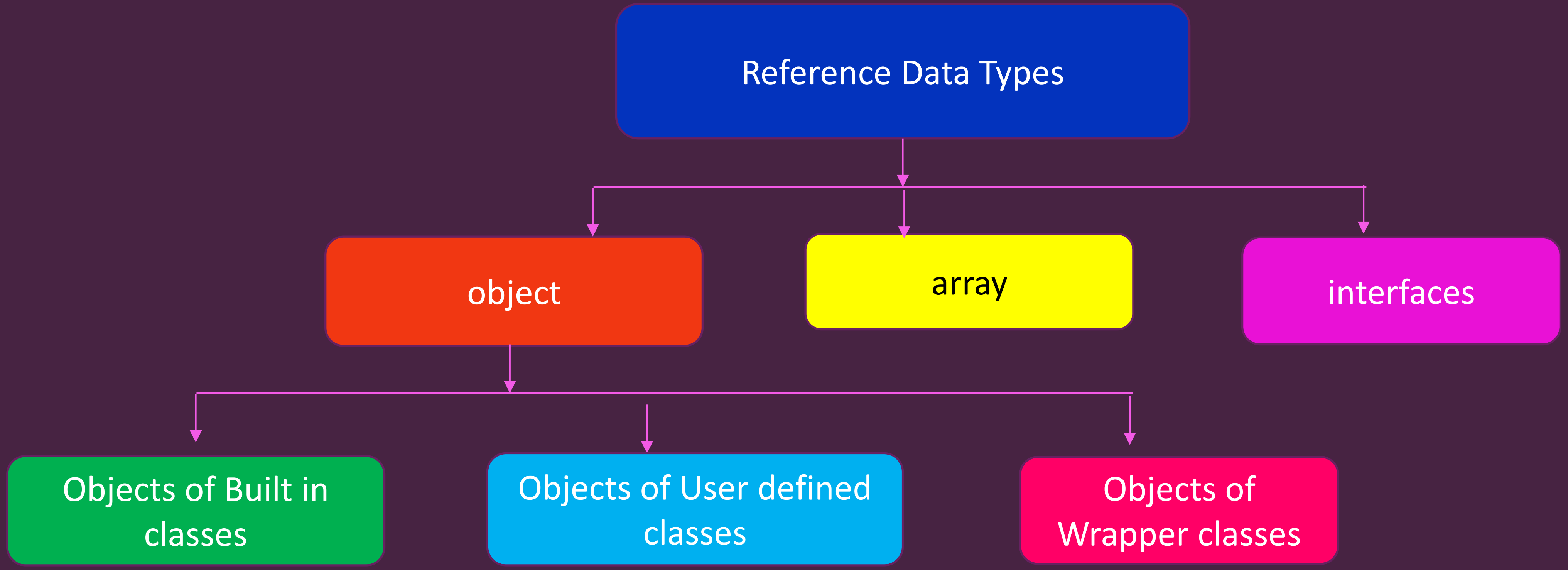
Default Values

- It's not always necessary to assign a value when a field(variable) is declared.
- Fields that are declared but not initialized will be set to a reasonable default by the compiler.
- Generally speaking, this default will be zero or null, depending on the data type.
- Relying on such default values, however, is generally considered bad programming style.

- Local variables are slightly different; the compiler never assigns a default value to an uninitialized local variable.
- If you cannot initialize your local variable where it is declared, make sure to assign it a value before you attempt to use it.
- Accessing an uninitialized local variable will result in a compile-time error.

Reference Data Types

store references (or addresses) to objects in memory, rather than actual values. They are not primitive types. A reference data type can be any object, array, or interface, and they can be classified into the following types:


Reference Data Types
object
array
interfaces
Objects of Built in classes
Objects of User defined classes
Objects of Wrapper classes

Differences Between Primitive and Reference Data Types

Feature	Primitive Data Types	Reference Data Types
Storage	Stores the actual value	Stores the reference/address to an object
Memory Usage	Uses fixed size (determined by the data type)	Size depends on the object or array size
Default Value	Has a default value (e.g., 0 for int, false for boolean)	Default is null for object references
Nullability	Cannot be null	Can be null
Example	int, char, boolean, double, etc.	String, Object, arrays, interfaces

Java Keyboard Input

Scanner class

To Read from ***Keyboard*** (Standard Input/console) You can use ***Scanner*** is a class in java.util package.

Scanner is used for obtaining the input of the primitive types like int, double etc. and strings. It is the easiest way to read input in a Java program, though not very efficient.

Reading Input from the Console

To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream (Keyboard).

```
Scanner scanner = new Scanner(System.in);
```

Reading Input from a File

```
File file = new File("input.txt");
Scanner scanner = new Scanner(file);
```

Reading Input from a String

```
Scanner scanner = new Scanner("Hello World");
```

Tokenization Process:

- The Scanner will treat any sequence of non-whitespace characters as a token.
- When using the Scanner class in Java, whitespace characters like spaces, tabs, and newlines are typically used as delimiters to separate tokens.
- When you use methods like nextInt(), nextDouble() or next() or other methods they do not consume the newline character (\n) that is entered when the user presses the Enter key after their input.
- However, the nextLine() method works a bit differently compared to nextFoo() methods.
- nextLine() reads the entire line of input, including any spaces, and returns it as a String. It consumes the newline character (\n) at the end of the line and returns everything up to the next newline.

Public methods in Scanner class.

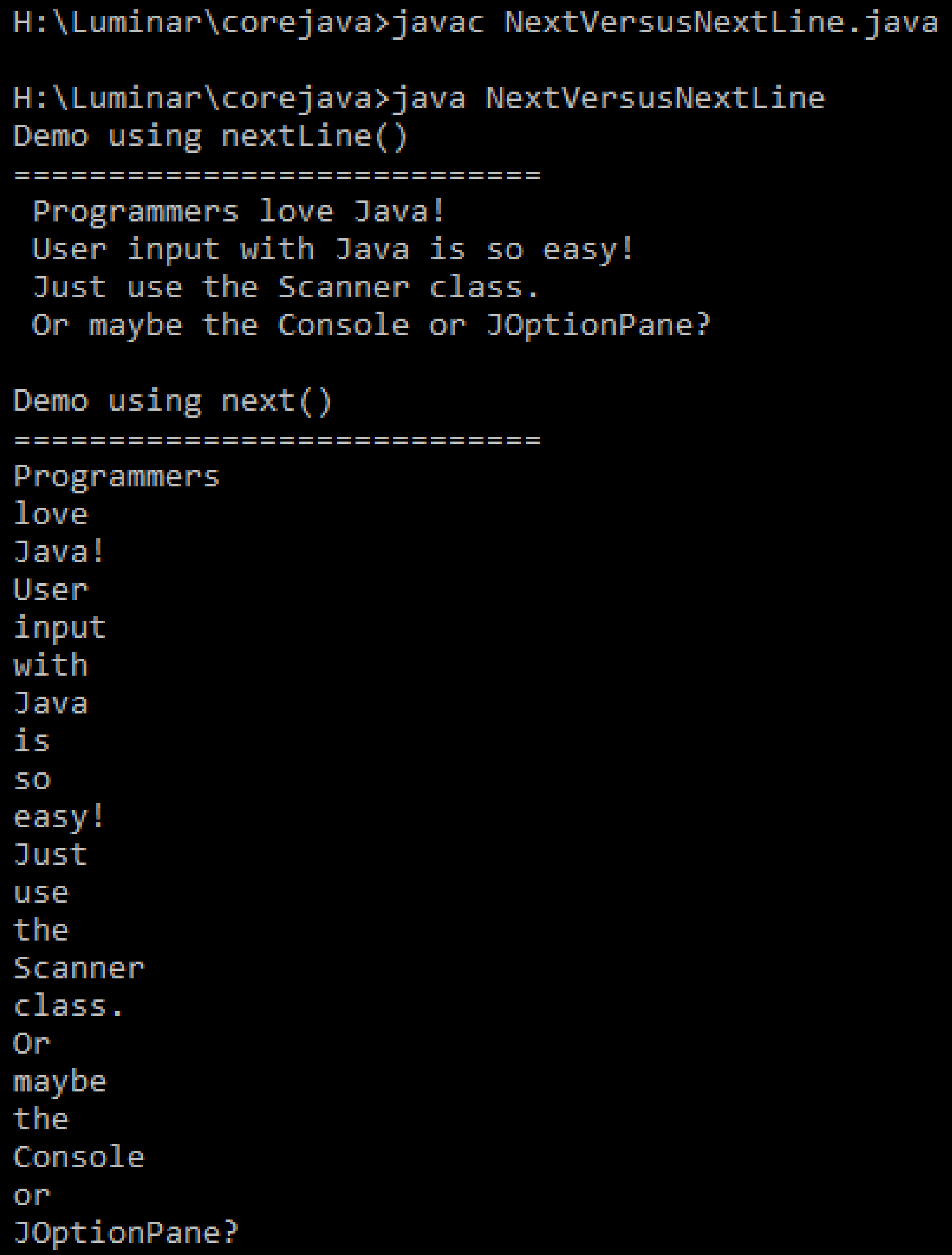
nextInt()	Reads an integer value.
nextFloat()	Reads a float value.
nextLine()	Reads an entire line of input (including spaces) and returns it as a String.
nextDouble()	Reads a double value.
next().charAt(0)	Reads a single character
next()	Reads a single word

Problem No 1

```
import java.util.Scanner;

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

            String sampleText =
                   " Programmers love Java!\n"                          //line no 1
                 + " User input with Java is so easy!\n"                //line no 2
                 + " Just use the Scanner class.\n"                     //line no 3
                 + " Or maybe the Console or JOptionPane?\n";           //line no 4

            Scanner scanner = new Scanner(sampleText);
            System.out.println("Demo using nextLine()\n============================");

            System.out.println(scanner.nextLine());
            System.out.println(scanner.nextLine());
            System.out.println(scanner.nextLine());
            System.out.println(scanner.nextLine());
```

```
Scanner scan = new Scanner(sampleText);

            System.out.println("\nDemo using next()\n===============================");
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());
            System.out.println(scan.next());

            scanner.close();
            scan.close();
      }
}
```

Output



```
H:\Luminar\corejava>javac NextVersusNextLine.java

H:\Luminar\corejava>java NextVersusNextLine
Demo using nextLine()
============================
 Programmers love Java!
 User input with Java is so easy!
 Just use the Scanner class.
 Or maybe the Console or JOptionPane?

Demo using next()
============================
Programmers
love
Java!
User
input
with
Java
is
so
easy!
Just
use
the
Scanner
class.
Or
maybe
the
Console
or
JOptionPane?
```

Explanation

Note:

next() can read the input only till the whitespace. It can't read two words separated by space. Also, next() places the cursor in the same line after reading the input.

nextLine() reads input including space between the words (that is, it reads till the end of line \n).

Problem No 2

```
Int rollNo;
String name;
String address;

Scanner sc=new Scanner(System.in);

System.out.println("Enter roll no");
rollNo=sc.nextInt(); // Read numerical value from input

System.out.println("Enter Name");
name=sc.nextLine(); // Read 1st string (this is skipped)

System.out.println("Enter Address");
address =sc.nextLine(); // Read 2nd string (this appears right after reading numerical value)
```

Output

```
Enter roll no
3               // This is the input
Enter Name      // The program is supposed to stop here and wait for the input, but is skipped
Enter Address   // ...and this line is executed and waits for the input
```

Scanner skipping nextLine() after use of other nextFoo()s**

**where Foo is any primitive type such as Int, Float, Double, etc

Problem Explanation:

When you use methods like nextInt(), nextDouble(), or next(), they do not consume the newline character (\n) that is entered when the user presses the Enter key after their input. As a result, after calling one of these methods, the newline character remains in the input buffer. E.g. nextInt() ->When you call nextInt(), it reads the integer input from the user but leaves the newline character (\n) in the buffer.

When you then call nextLine(), it immediately consumes the leftover newline character (\n), instead of waiting for the user to enter a new line. This can lead to unexpected behavior where nextLine() might return an empty string.

Solution

you should place sc.nextLine() immediately after

rollNo=sc.nextInt();

or

balance=sc.nextDouble()

or

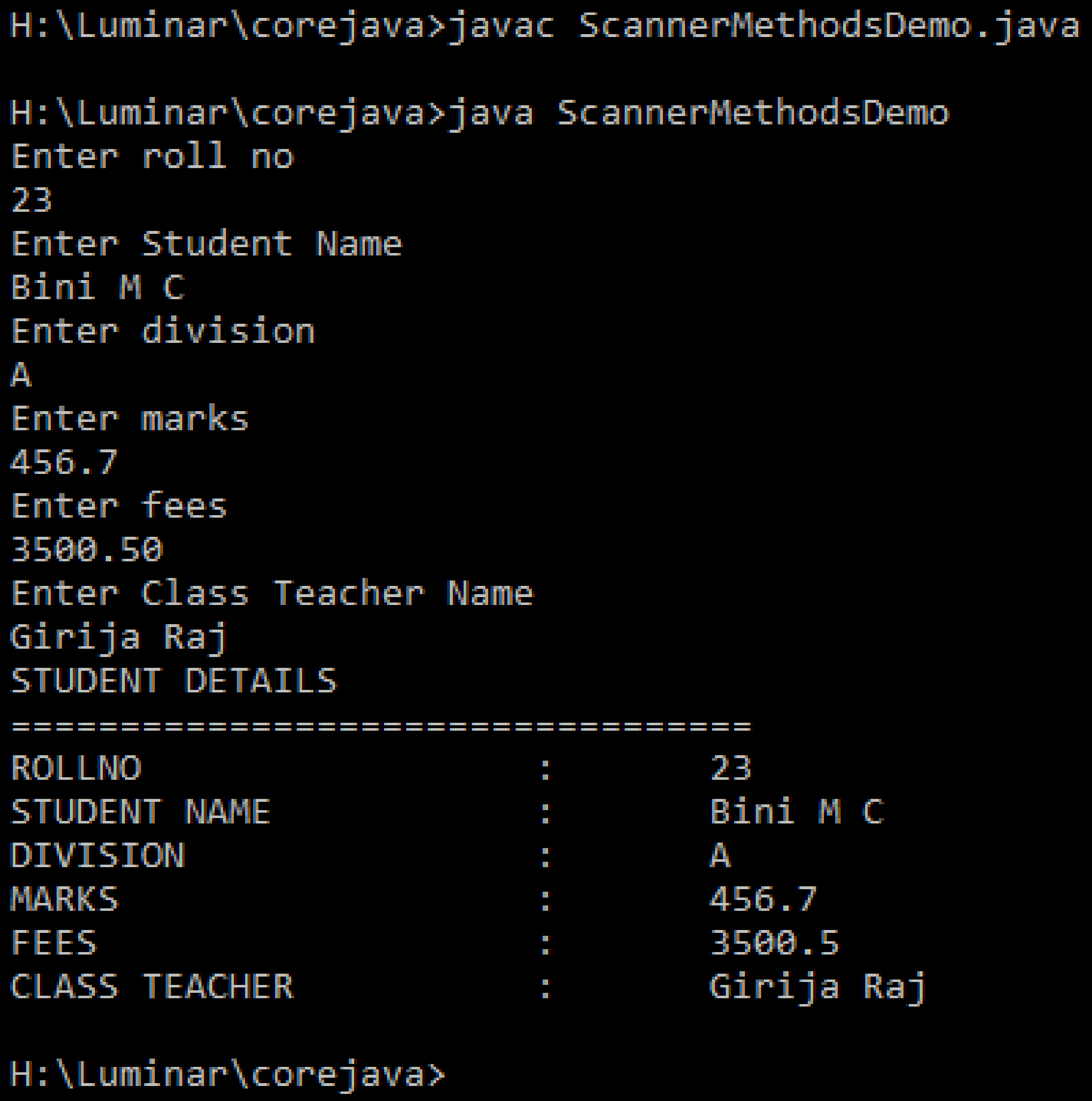
marks=sc.nextFloat() ..etc

to consume the leftover newline character.

```
import java.util.Scanner;

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

		Scanner sc = new Scanner(System.in);

		int rollNo;
		String studentName;
		char division;
		float marks;
		double fees;
		String classTeacherName;

		System.out.println("Enter roll no");
		rollNo=sc.nextInt();
		sc.nextLine();
```

```
System.out.println("Enter Student Name");
studentName=sc.nextLine();

System.out.println("Enter division");
division=sc.next().charAt(0);

System.out.println("Enter marks");
marks=sc.nextFloat();

System.out.println("Enter fees");
fees=sc.nextDouble();

sc.nextLine();

System.out.println("Enter Class Teacher Name");
classTeacherName=sc.nextLine();
```

```
		System.out.println("STUDENT DETAILS \n=====================================");
		System.out.println("ROLLNO \t\t\t:\t"+rollNo);
		System.out.println("STUDENT NAME \t\t:\t"+studentName);
		System.out.println("DIVISION \t\t:\t"+division);
		System.out.println("MARKS \t\t\t:\t"+marks);
		System.out.println("FEES \t\t\t:\t"+fees);
		System.out.println("CLASS TEACHER \t\t:\t"+classTeacherName);

	}
}
```



```
H:\Luminar\corejava>javac ScannerMethodsDemo.java

H:\Luminar\corejava>java ScannerMethodsDemo
Enter roll no
23
Enter Student Name
Bini M C
Enter division
A
Enter marks
456.7
Enter fees
3500.50
Enter Class Teacher Name
Girija Raj
STUDENT DETAILS
==================================
ROLLNO                  :       23
STUDENT NAME            :       Bini M C
DIVISION                :       A
MARKS                   :       456.7
FEES                    :       3500.5
CLASS TEACHER           :       Girija Raj

H:\Luminar\corejava>
```

Escape Sequences

\t: Inserts a tab
This sequence inserts a tab in the text where it’s used.

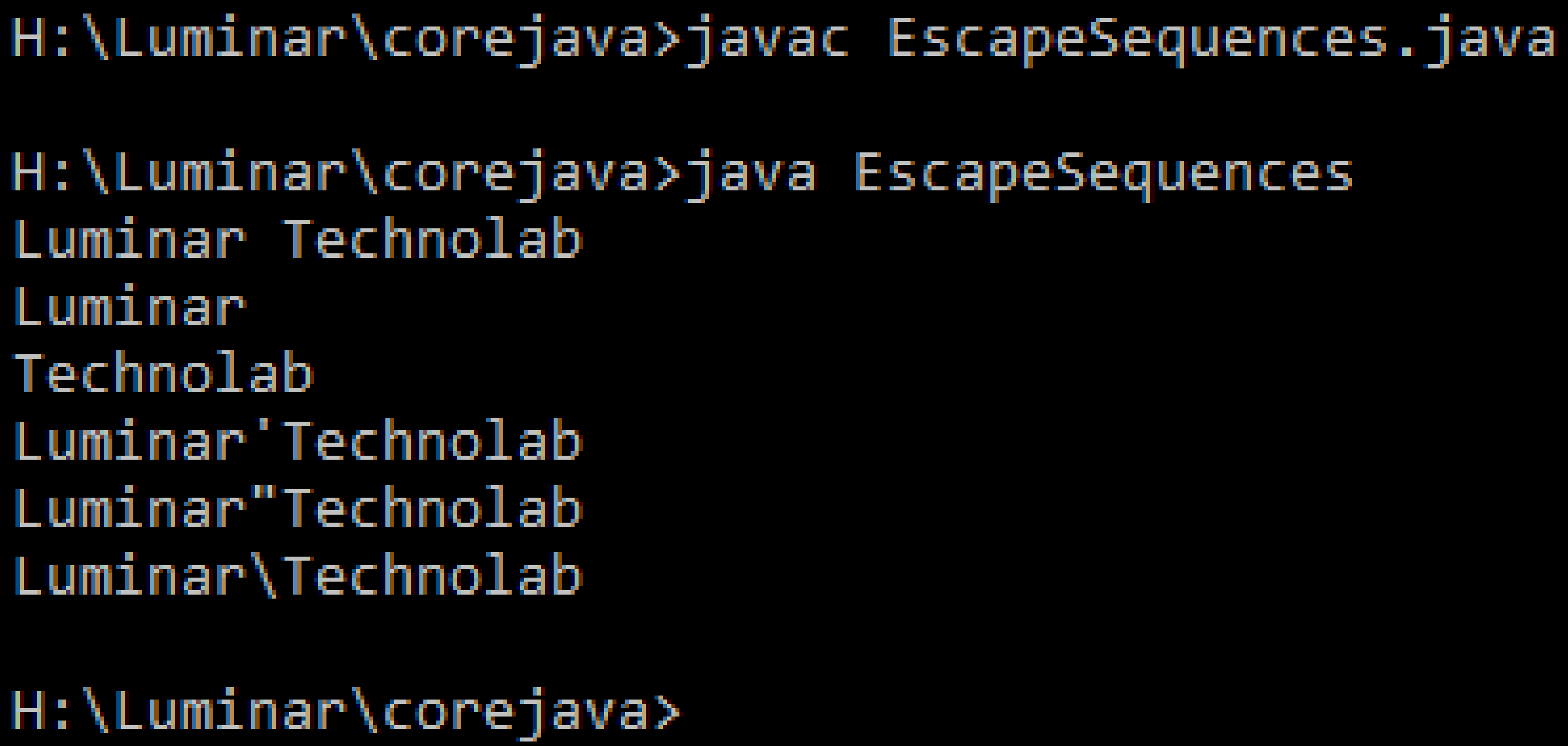
\n: Inserts a new line
This sequence inserts a new line in the text where it’s used

\': Inserts a single quote
This sequence inserts a single quote character in the text where it’s used.

\": Inserts a double quote
This sequence inserts a double quote character in the text where it’s used.

\\: Inserts a backslash
This sequence inserts a backslash character in the text where it’s used.

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

        // Add a tab between Luminar and Technolab
        System.out.println( "Luminar\tTechnolab " );

        // Add a new line after Luminar
        System.out.println( "Luminar\nTechnolab" );

       // Add a single quote between Luminar and Technolab
        System.out.println( "Luminar\'Technolab" );

        // Add a double quote between Luminar and Technolab
        System.out.println( "Luminar\"Technolab" );

        // Add a backslash between Luminar and Technolab
        System.out.println( "Luminar\\Technolab" );
    }
}
```



```
H:\Luminar\corejava>javac EscapeSequences.java

H:\Luminar\corejava>java EscapeSequences
Luminar Technolab
Luminar
Technolab
Luminar'Technolab
Luminar"Technolab
Luminar\Technolab

H:\Luminar\corejava>
```

Java Operators

Operand:

Arguments on which operation is performed are known as operand.

$$\begin{tikzcd}
 & \text{Types of operand} \arrow[dl] \arrow[dr] & \\
\text{Constant} & & \text{Variable}
\end{tikzcd}$$

(Whose values can't be changed During program execution)

(Whose values can be changed During program execution)

Eg. A+3 : A(Variable) and 3(Constants) are operands)

Operator:

Operator is a special symbol used for performing a specific task. e.g. A+B. Here + symbol represents the operator.

Expression:

A sequence of operands connected by operators is known as expression. e.g. A+B*5.

Java provides a rich set of operators’ environment. Java operators can be divided into following categories:

1. Arithmetic operators
2. Relation operators
3. Logical operators
4. Bitwise operators
5. Assignment operators
6. Conditional operators
7. Misc operators

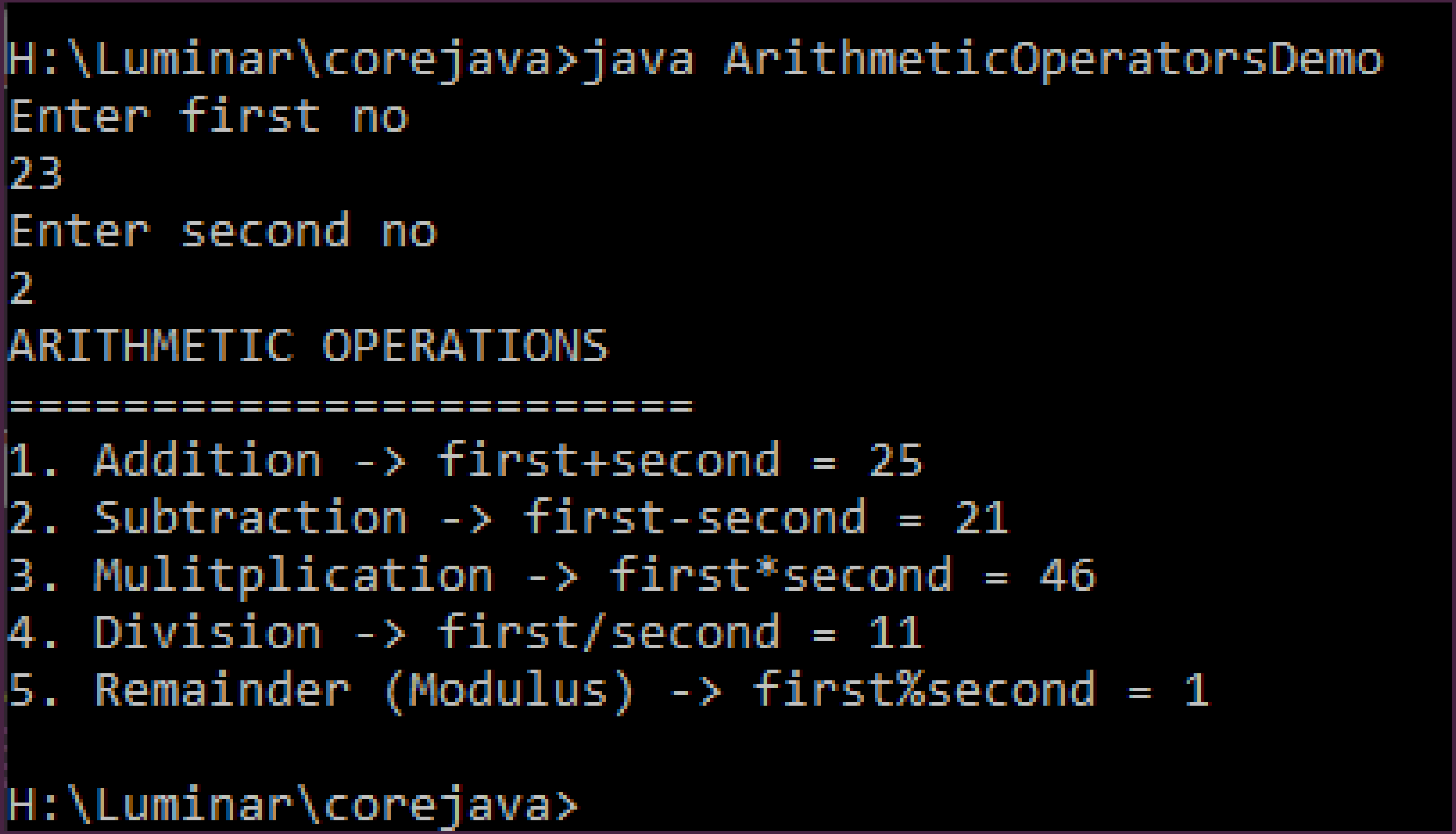
Arithmetic operators

Arithmetic operators are used in mathematical expression in the same way that are used in algebra.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by enumerator
%	remainder of division
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one

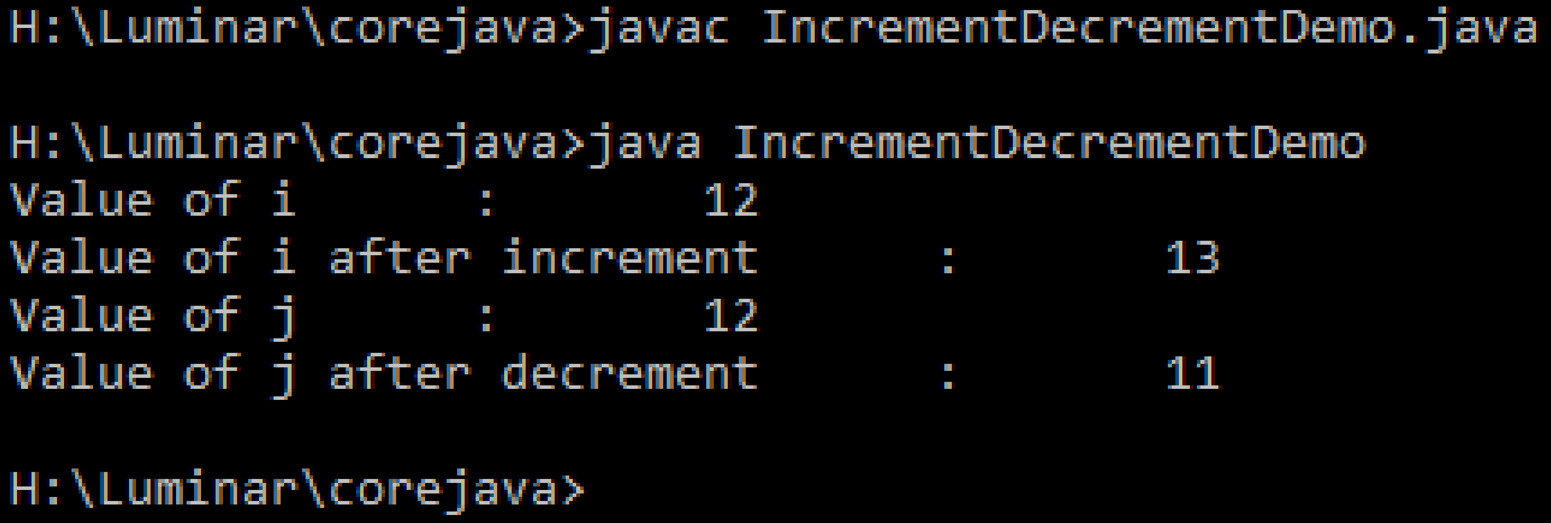
```
import java.util.Scanner;
class ArithmeticOperatorsDemo {
	public static void main(String args[]) {
		Scanner scan=new Scanner(System.in);
		int first;
		int second;
		System.out.println("Enter first no");
		first=scan.nextInt();
		System.out.println("Enter second no");
		second=scan.nextInt();

		System.out.println("ARITHMETIC OPERATIONS\n========================");

		System.out.println("1. Addition -> first+second = "+(first+second));
		System.out.println("2. Subtraction -> first-second = "+(first-second));
		System.out.println("3. Mulitplication -> first*second = "+(first*second));
		System.out.println("4. Division -> first/second = "+(first/second));
		System.out.println("5. Remainder (Modulus) -> first%second = "+(first%second));
	}
}
```

Output



```
H:\Luminar\corejava>java ArithmeticOperatorsDemo
Enter first no
23
Enter second no
2
ARITHMETIC OPERATIONS
========================
1. Addition -> first+second = 25
2. Subtraction -> first-second = 21
3. Mulitplication -> first*second = 46
4. Division -> first/second = 11
5. Remainder (Modulus) -> first%second = 1

H:\Luminar\corejava>
```

```
public class IncrementDecrementDemo {
	public static void main(String[] args) {
		int i=12;
		int j=12;

		System.out.println("Value of i \t:\t"+i);
		i++;
		System.out.println("Value of i after increment \t:\t"+i);

		System.out.println("Value of j \t:\t"+j);
		j--;
		System.out.println("Value of j after decrement \t:\t"+j);
	}
}
```



```
H:\Luminar\corejava>javac IncrementDecrementDemo.java

H:\Luminar\corejava>java IncrementDecrementDemo
Value of i      :       12
Value of i after increment      :       13
Value of j      :       12
Value of j after decrement      :       11

H:\Luminar\corejava>
```

Thank you ☺ Happy Learning ☺