Core Java

Course Introduction

Course contents -- Java 8

- Language features (Installation, Buzzwords, History, JRE/JDK/JVM, Compilation, Operators, Data types, Narrowing/Widening)
- Control Structures (if-else, loops, switch, recursion, wrapper classes, boxing, value/reference type)
- Input/Output: Commandline args, Scanner, Console
- OOP basics (OOP pillars, class/object, class methods, ctor chaining, overloading, final field, toString()), enum
- Static (static fields/method/block), Singleton, Package (classpath), static import
- Arrays (1-D & 2-D array, primitive and object array, java.util.Arrays)
- OOP advanced (aggregation/association/composition, Inheritance, super, Method overriding, instanceof, final method/class, Abstract class, Java 7 Interfaces, Marker interfaces)
- Java classes (java.lang.Object, Date/LocalDate/Calendar, String/StringBuilder/StringBuffer, etc)
- Exception Handling (try-catch-throw, throws, finally, try-with-resource/Closeable, errors, custom exception, checked/unchecked, overriding rules)
- Java Generics (methods, classes, interfaces), Comparable/Comparator interface, Java 8 Interfaces
- Nested classes, Functional Interfaces, Lambda expressions, Method references
- Java Collections (Hierarchy, Lists, Iterator, Sets, Maps, Queue/Stack, Fail-safe/Fail-fast Iterators, Legacy collections, Collections)
- Functional programming (Concepts, Java 8 interfaces, Stream characteristics, Stream operations, Collectors)
- Java IO (Binary vs Text streams, File IO, Stream chaining, Data stream, Serialization, Reader/Writers)
- Multi-threading (Thread vs Runnable, thread methods, Thread group, Synchronization, Deadlock)
- JVM Architecture, Reflection, Annotation, Garbage Collection

Reference Books

- Core Java Volume 1 & 2 - Horstmann
- Java Complete Reference - Herbert Schildt
- Java 8 Certification - Khalid Mughal
- Java Certification - Kathy Sierra

Java History

- Java goes back to 1991, when a group of Sun engineers, led by Patrick Naughton and James Gosling, wanted to design a small computer language that could be used for consumer devices like cable TV switchboxes.
- Since these devices do not have a lot of power or memory, the language had to be small and generate very tight code. Also, as different manufacturers may choose different CPU's, it was important that the language not be tied to any single architecture.
- The project was code-named "Green".

- The requirements for small, tight, and platform-neutral code led the team to design a portable language that generated intermediate code for a virtual machine.
- The Sun people came from a UNIX (Solaris) background, so they based their language on C++.
- Gosling decided to call his language "Oak". However, Oak was the name of an existing computer language, so they changed the name to Java.
- In 1992, the Green project delivered its first product, called "*7" - a smart remote control.
- Unfortunately, Sun was not interested in producing this, and the Green people had to find other ways to market their technology. However, none of the standard consumer electronics companies were interested either.
- The Green team spent all of 1993 and half of 1994 looking for sponsors to buy its technology.
- Meanwhile, the World Wide Web (WWW) was growing bigger and bigger. The key to the WWW was the browser translating hypertext pages to the screen.
- In 1994, most people were using Mosaic, a noncommercial web browser by University of Illinois.
- The Java language developers developed a cool browser - HotJava browser. This was client/server architecture-neutral application that was real-time and reliable.
- The developers made the browser capable of executing Java code inside web pages - called Applets. This POC was demonstrated at SunWorld on 23-May-1995.

Java versions

- JDK Beta - 1995
- **JDK 1.0 - January 23, 1996**
- JDK 1.1 - February 19, 1997
- **J2SE 1.2 - December 8, 1998**
  - Java collections
- J2SE 1.3 - May 8, 2000
- J2SE 1.4 - February 6, 2002
- **J2SE 5.0 - September 30, 2004**
  - enum
  - Generics
  - Annotations
- Java SE 6 - December 11, 2006
- Java SE 7 - July 28, 2011
- **Java SE 8 (LTS) - March 18, 2014**
  - Functional programming: Streams, Lambda expressions
- Java SE 9 - September 21, 2017
- Java SE 10 - March 20, 2018
- **Java SE 11 (LTS) - September 25, 2018**
- Java SE 12 - March 19, 2019
- Java SE 13 - September 17, 2019
- Java SE 14 - March 17, 2020
- Java SE 15 - September 15, 2020
- Java SE 16 - March 16, 2021
- **Java SE 17 (LTS) - September 14, 2021**
  - Jakarta SE 17
- Java SE 18 - March 22, 2022
- Java SE 19 - September 20, 2022

- Java SE 20 - March 21, 2023

Java platforms

- Java is not specific to any processor or operating system as Java platforms have been implemented for a wide variety of hardware and operating systems with a view to enable Java programs to run identically on all of them. Different platforms target different classes of device and application domains:
- **Java Card**: A technology that allows small Java-based applications to be run securely on smart cards and similar small-memory devices.
- **Java ME (Micro Edition)**: Specifies several different sets of libraries (known as profiles) for devices with limited storage, display, and power capacities. It is often used to develop applications for mobile devices, PDAs, TV set-top boxes, and printers.
- **Java SE (Standard Edition)**: Java Platform, Standard Edition or Java SE is a widely used platform for development and deployment of portable code for desktop environments
- **Java EE (Enterprise Edition)**: Java Platform, Enterprise Edition or Java EE is a widely used enterprise computing platform. The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications.

Object Oriented

- Basic principles of Object-oriented Language
  - class
  - object
- class
  - User defined data type (similar to struct in C)
  - Has fields (data members) and methods (member functions)
    - static members: Accessed using class name directly
    - non-static members: Accessed using object
  - Defines the structure/blueprint of the created object/instance
  - Logical entity
- object
  - Instance of the class
  - One class can have multiple objects
  - Physical entity (occupies memory)

Hello World - Code, Compilation and Execution

- Code

```java
// Program.java
class Program {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

- Explanation - main():
  - In Java, each variable/method must be in some class.
  - JVM calls main() method without creating object of the class, so method must be static.
  - main() doesn't return any value to JVM, so return type is void.
  - main() takes command line arguments - String args[]
  - main() should be callable from outside the class directly - public access.
- Explanation - System.out.println():
  - System is predefined Java class (java.lang.System).
  - out is public static field of the System class --> System.out.
  - out is object of PrintStream class (java.io.PrintStream).
  - println() is public non-static method of PrintStream class --> System.out.println("...");
- Compilation and Execution (in same directory)
  - terminal> javac Program.java
  - terminal> java Program

Entry point method

- main() is considered as entry poit method in java.

  ```java
  public static void main(String[] args) {
      // code
  }
  ```

- JVM invokes main method.
- Can be overloaded.
- Can write one entry-point in each Java class.

java.lang.System class

~~~
```Java
public final class System
{
    //Fields
    public static final InputStream in;     // stdin
    public static final PrintStream out;    // stdout
    public static final PrintStream err;    // stderr
    //Methods
    public static Console console();
    public static void exit(int status);
    public static void gc();
    // ...
}
```
~~~

System.out.println()

- System: Final class declared in java.lang package and java.lang package is declared in rt.jar file.
- out: Object of PrintStream class declared as public static final field in System class.
- println(): Non-static method of PrintStream class.

C/C++ Program Compilation and Execution

- Main.cpp --> Compiler --> Main.obj --> Linker --> Main.exe
  - Main.cpp - Source code
  - Main.obj - Object code
  - Main.exe - Program = Executable code (contains machine level code)
- terminal> ./Main.exe
- Operating system creates a process to execute the program.
- Process sections
  - Text:
  - Data:
  - Rodata:
  - Stack:
  - Heap

Java Program Compilation and Execution

- Main.java --> Compiler --> Main.class --> JVM
  - Main.java --> Source code
  - Main.class --> Byte code (Intermediate Language code)
- terminal> javac Main.java
- terminal> java Main

JDK vs JRE vs JVM

- SDK is Software Development Kit required to develop application.
- SDK = Software Development Tools + Libraries + Runtime environment + Documentation + IDE.
  - Software Development Tools = Compiler, Debugger, etc.
  - Libraries = Set of functions/classes.
- JDK is Java platform SDK. It is a software development environment used for developing Java applications.
- JDK = Java Development Tools + JRE + Java docs.
  - Required to develop Java applications.
- JRE = Java API (Java class libraries) + Java Virtual Machine (JVM).
  - All core java fundamental classes are part of rt.jar file.
  - Required to develop and execute Java applications.

Hello World - Variations

- In STS Eclipse, classes are written under "src" directory. They are auto-compiled and generated .class files are placed under "bin" directory.
- One Java project can have multiple .java files. Each file can have main() method which can be executed separately.
- The main() method must be public static void. Missing any of them raise compiler error.

- The entry-point method must be main(String[] args). Otherwise, raise runtime error - main() method not found.
- The main() method can be overloaded i.e. method with same name but different parameters (in same class).
- If a .java file contains multiple classes, for each class a separate .class file is created.
- Name of (non-public) Java class may be different than the file name. The name of generated .class file is same as class name.
- Name of public class in Java file must be same as file-name. One Java file can have only one public class.

PATH vs CLASSPATH

- Enviorment variables: Contains important information about the system e.g. OS, CPU, PATH, USER, etc.
- PATH: Contains set of directories separated by ; (Windows) or : (Linux).
  - When any program (executable file) is executed without its full path (on terminal/Run), then OS search it in all directories given in PATH variable.
  - terminal> mspaint.exe
  - terminal> notepad.exe
  - terminal> taskmgr.exe
  - terminal> java.exe -version
  - terminal> javac.exe -version
  - To display PATH variable
    - Windows cmd> set PATH
    - Linux terminal> echo $PATH
  - PATH variable can be modified using "set" command (Windows) or "export" command (Linux).
  - PATH variable can be modified permanently in Windows System settings or Linux ~/.bashrc.
- CLASSPATH: Contains set of directories separated by ; (Windows) or : (Linux).
  - Java's environment variable by which one can inform Java compiler, application launcher, JVM and other Java tools about the directories in which Java classes/packages are kept.
  - CLASSPATH variable can be modified using "set" command (Windows) or "export" command (Linux).
    - Windows cmd> set CLASSPATH=\path\to\set;%CLASSPATH%
    - Linux terminal> export CLASSPATH=/path/to/set:$CLASSPATH
  - To display CLASSPATH variable
    - Windows cmd> set CLASSPATH
    - Linux terminal> echo $CLASSPATH
- Compilation and Execution (source code in "src" directory and .class file in "bin" directory)
  - terminal> cd \path\of\src directory
  - terminal> javac -d ..\bin Program.java
  - terminal> set CLASSPATH=..\bin
  - terminal> java Program

Console Input/Output

- Java has several ways to take input and print output. Most popular ways in Java 8 are given below:
- Using java.util.Scanner and System.out

```java
Scanner sc = new Scanner(System.in);
System.out.print("Enter name: ");
String name = sc.nextLine();
System.out.print("Enter age: ");
int age = sc.nextInt();
System.out.println("Name: " + name + ", Age: " + age);
System.out.printf("Name: %s, Age: %s\n", name, age);
```

Language Fundamentals

Naming conventions

- Names for variables, methods, and types should follow Java naming convention.
- Camel notation for variables, methods, and parameters.
    - First letter each word except first word should be capital.
    - For example:

    ```java
    public double calculateTotalSalary(double basicSalary, double
    incentives) {
        double totalSalary = basicSalary + incentives;
        return totalSalary;
    }
    ```

- Pascal notation for type names (i.e. class, interface, enum)
    - First letter each word should be capital.
    - For example:

    ```java
    class CompanyEmployeeManagement {
        // ...
    }
    ```

- Package names must be in lower case only.
    - For example: javax.servlet.http;
- Constant fields must be in upper case only.
    - For example:

    ```java
    final double PI = 3.14;
    final int WEEKDAYS = 7;
    final String COMPANY_NAME = "Sunbeam Infotech";
    ```

Keywords

- Keywords are the words whose meaning is already known to Java compiler.

- These words are reserved i.e. cannot be used to declare variable, function or class.
- Java 8 Keywords
    1. abstract - Specifies that a class or method will be implemented later, in a subclass
    2. assert - Verifies the condition. Throws error if false.
    3. boolean- A data type that can hold true and false values only
    4. break - A control statement for breaking out of loops.
    5. byte - A data type that can hold 8-bit data values
    6. case - Used in switch statements to mark blocks of text
    7. catch - Catches exceptions generated by try statements
    8. char - A data type that can hold unsigned 16-bit Unicode characters
    9. class - Declares a new class
    10. continue - Sends control back outside a loop
    11. default - Specifies the default block of code in a switch statement
    12. do - Starts a do-while loop
    13. double - A data type that can hold 64-bit floating-point numbers
    14. else - Indicates alternative branches in an if statement
    15. enum - A Java keyword is used to declare an enumerated type. Enumerations extend the base class.
    16. extends - Indicates that a class is derived from another class or interface
    17. final - Indicates that a variable holds a constant value or that a method will not be overridden
    18. finally - Indicates a block of code in a try-catch structure that will always be executed
    19. float - A data type that holds a 32-bit floating-point number
    20. for - Used to start a for loop
    21. if - Tests a true/false expression and branches accordingly
    22. implements - Specifies that a class implements an interface
    23. import - References other classes
    24. instanceof - Indicates whether an object is an instance of a specific class or implements an interface
    25. int - A data type that can hold a 32-bit signed integer
    26. interface- Declares an interface
    27. long - A data type that holds a 64-bit integer
    28. native - Specifies that a method is implemented with native (platform-specific) code
    29. new - Creates new objects
    30. null - This indicates that a reference does not refer to anything
    31. package - Declares a Java package
    32. private - An access specifier indicating that a method or variable may be accessed only in the class it’s declared in
    33. protected - An access specifier indicating that a method or variable may only be accessed in the class it’s declared in (or a subclass of the class it’s declared in or other classes in the same package)
    34. public - An access specifier used for classes, interfaces, methods, and variables indicating that an item is accessible throughout the application (or where the class that defines it is accessible)
    35. return - Sends control and possibly a return value back from a called method
    36. short - A data type that can hold a 16-bit integer
    37. static - Indicates that a variable or method is a class method (rather than being limited to one particular object)

38. strictfp - A Java keyword is used to restrict the precision and rounding of floating-point calculations to ensure portability.
39. super - Refers to a class's base class (used in a method or class constructor)
40. switch - A statement that executes code based on a test value
41. synchronized - Specifies critical sections or methods in multithreaded code
42. this - Refers to the current object in a method or constructor
43. throw - Creates an exception
44. throws - Indicates what exceptions may be thrown by a method
45. transient - Specifies that a variable is not part of an object's persistent state
46. try - Starts a block of code that will be tested for exceptions
47. void - Specifies that a method does not have a return value
48. volatile - This indicates that a variable may change asynchronously
49. while - Starts a while loop
50. goto, const - Unused keywords (Reserved words)
51. true, false, null - Literals (Reserved words)

Data types

- Data type describes:
  - Memory is required to store the data
  - Kind of data memory holds
  - Operations to perform on the data
- Java is strictly type checked language.
- In java, data types are classified as:
  - Primitive types or Value types
  - Non-primitive types or Reference types

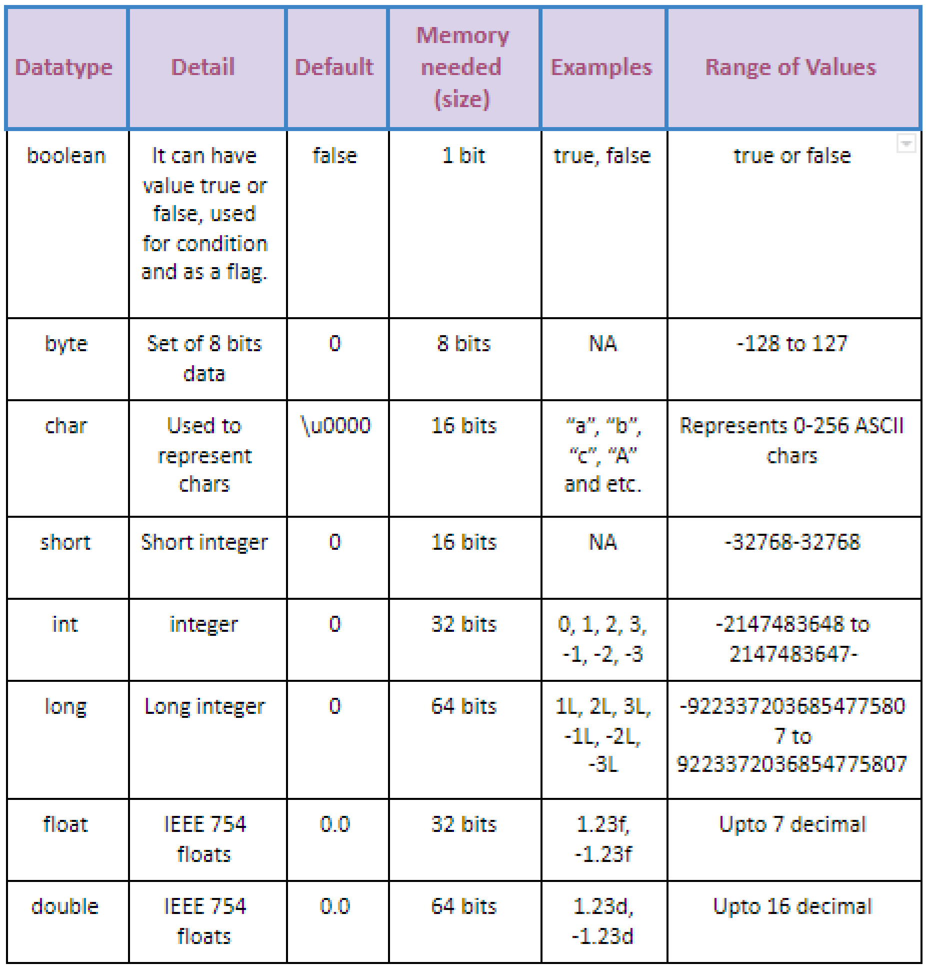
```
Data types
    |- Primitive types (Value types)
    |      |- Boolean: boolean
    |      |- Character: char
    |      |- Integral: byte, short, int, long
    |      |- Floating-point: float, double
    |
    |- Non-Primitive types (Reference types)
           |- class
           |- interface
           |- enum
           |- Array
```

1. boolean (size is not specified)
2. byte (size is 1 byte)
3. char (size is 2 bytes)
4. short (size is 2 bytes)
5. int (size is 4 bytes)
6. float (size is 4 bytes)
7. double (size is 8 bytes)

8. long (size is 8 bytes)

- primitive types(boolean, byte, char, short, int ,float, double, long) are not classes in Java.

```
Stack<int> s1 =  new Stack<int>( ); //Not OK
Stack<Integer> s1 =  new Stack<Integer>( ); //OK
```

- 

Datatype	Detail	Default	Memory needed (size)	Examples	Range of Values
boolean	It can have value true or false, used for condition and as a flag.	false	1 bit	true, false	true or false
byte	Set of 8 bits data	0	8 bits	NA	-128 to 127
char	Used to represent chars	\u0000	16 bits	"a", "b", "c", "A" and etc.	Represents 0-256 ASCII chars
short	Short integer	0	16 bits	NA	-32768-32768
int	integer	0	32 bits	0, 1, 2, 3, -1, -2, -3	-2147483648 to 2147483647-
long	Long integer	0	64 bits	1L, 2L, 3L, -1L, -2L, -3L	-9223372036854775807 to 9223372036854775807
float	IEEE 754 floats	0.0	32 bits	1.23f, -1.23f	Upto 7 decimal
double	IEEE 754 floats	0.0	64 bits	1.23d, -1.23d	Upto 16 decimal

- Widening: We can convert state of object of narrower type into wider type. it is called as "widening".

  ```
  int num1 = 10;
  double num2 = num1; //widening
  ```

- Narrowing: We can convert state of object of wider type into narrower type. It is called "narrowing".

  ```
  double num1 = 10.5;
  int num2 = (int) num1; //narrowing
  ```

- Rules of conversion
  - source and destination must be compatible i.e. destination data type must be able to store larger/equal magnitude of values than that of source data type.
  - Rule 1: Arithmetic operation involving byte, short automatically promoted to int.

    - Rule 2: Arithmetic operation involving int and long promoted to long.
    - Rule 3: Arithmetic operation involving float and long promoted to float.
    - Rule 4: Arithmetic operation involving double and any other type promoted to double.
- Type Conversions

Literals

- Six types of Literals:
    - Integral Literals
    - Floating-point Literals
    - Char Literals
    - String Literals
    - Boolean Literals
    - null Literal

Integral Literals

- Decimal: It has a base of ten, and digits from 0 to 9.
- Octal: It has base eight and allows digits from 0 to 7. Has a prefix 0.
- Hexadecimal: It has base sixteen and allows digits from 0 to 9 and A to F. Has a prefix 0x.
- Binary: It has base 2 and allows digits 0 and 1.
- For example:

```
int x = 65; // decimal const don't need prefix
int y = 0101; // octal values start from 0
int z = 0x41; // hexadecimal values start from 0x
int w = 0b01000001; // binary values start with 0b
```

- Literals may have suffix like U, L.
    - L -- represents long value.

```
long x = 123L; // long const assigned to long variable
long y = 123; // int const assigned to long variable -- widening
```

Floating-Point Literals

- Expressed using decimal fractions or exponential (e) notation.
- Single precision (4 bytes) floating-point number. Suffix f or F.
- Double precision (8 bytes) floating-point number. Suffix d or D.
- For example:

```
float x = 123.456f;
float y = 1.23456e+2;   // 1.23456 x 10^2 = 123.456
double z = 3.142857d;
```

Char Literals

- Each char is internally represented as integer number - ASCII/Unicode value.
- Java follows Unicode char encoding scheme to support multiple langauges.
- For example:

  ```
  char x = 'A';       // char representation
  char y = '\101';    // octal value
  char z = '\u0041';  // unicode value in hex
  char w = 65;        // unicode value in dec as int
  ```

- There are few special char literals referred as escape sequences.
  - \n -- newline char -- takes cursor to next line
  - \r -- carriage return -- takes cursor to start of current line
  - \t -- tab (group of 8 spaces)
  - \b -- backspace -- takes cursor one position back (on same line)
  - ' -- single quote
  - " -- double quote
  - \ -- prints single \
  - \0 -- ascii/unicode value 0 -- null character

String Literals

- A sequence of zero or more unicode characters in double quotes.
- For example:

  ```
  String s1 = "Sunbeam";
  ```

Boolean Literals

- Boolean literals allow only two values i.e. true and false. Not compatible with 1 and 0.
- For example:

  ```
  boolean b = true;
  boolean d = false;
  ```

Null Literal

- "null" represents nothing/no value.
- Used with reference/non-primitive types.

  ```
  String s = null;
  Object o = null;
  ```

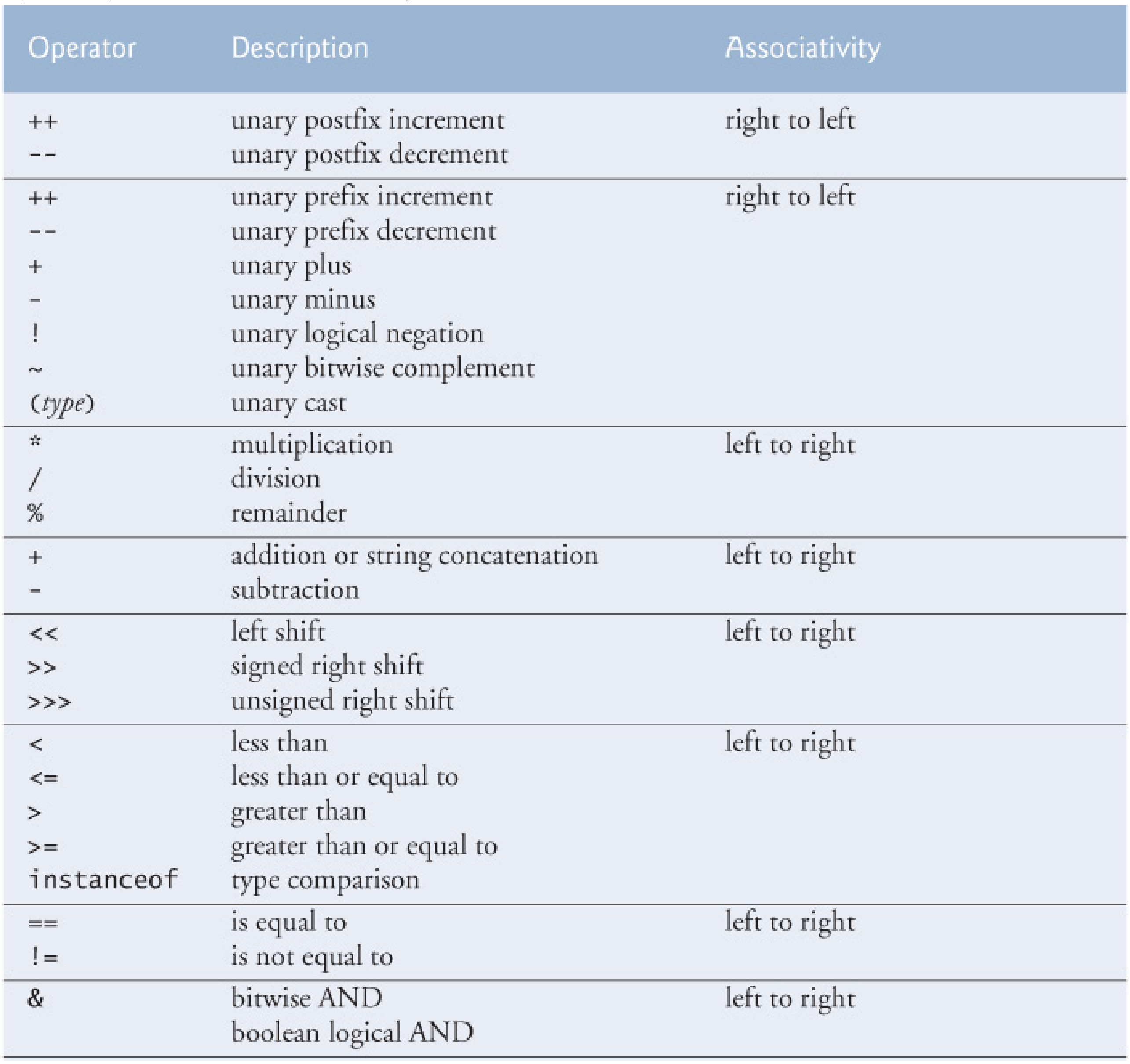
Variables

- A variable is a container which holds a value. It represents a memory location.
- A variable is declared with data type and initialized with another variable or literal.
- In Java, variable can be
  - Local: Within a method -- Created on stack.
  - Non-static/Instance field: Within a class - Accessed using object.
  - Static field: Within a class - Accessed using class-name.

Operators

- Java divides the operators into the following catgories:
  - Arithmetic operators: +, -, *, /, %
  - Assignment operators: =, +=, -=, etc.
  - Comparison operators: ==, !=, <, >, <=, >=, instanceof
  - Logical operators: &&, ||, !
    - Combine the conditions (boolean - true/false)
  - Bitwise operators: &, |, ^, ~, <<, >>, >>>
  - Misc operators: ternary ?:, dot .
    - Dot operator: ClassName.member, objName.member.

- Operator precedence and associativity



Operator	Description	Associativity
`++`	unary postfix increment	right to left
`--`	unary postfix decrement	
`++`	unary prefix increment	right to left
`--`	unary prefix decrement	
`+`	unary plus	
`-`	unary minus	
`!`	unary logical negation	
`~`	unary bitwise complement	
(*type*)	unary cast	
`*`	multiplication	left to right
`/`	division	
`%`	remainder	
`+`	addition or string concatenation	left to right
`-`	subtraction	
`<<`	left shift	left to right
`>>`	signed right shift	
`>>>`	unsigned right shift	
`<`	less than	left to right
`<=`	less than or equal to	
`>`	greater than	
`>=`	greater than or equal to	
`instanceof`	type comparison	
`==`	is equal to	left to right
`!=`	is not equal to	
`&`	bitwise AND	left to right
	boolean logical AND	