

## Agenda

- Console input and output
- Language Fundamentals
- Class Object and Reference
- Widening and Narrowing
- Wrapper classes
- Boxing & UnBoxing
- Command Line Arguments

## Console input and output

- Java has several ways to take input and print output. Most popular ways in Java 8 are
  - 1. using Scanner class (Program01)
    - it is present in java.util package

```
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);
```

- 2. using Console class
  - it is present in java.io package

```
Console console = System.console();
String email = console.readLine("Enter Email: ");
char[] passwd = console.readPassword("Enter Password: ");
console.printf("Email: %s\n", email);
```

## 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:

```
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:

```
class EmployeeManagement{  
}
```

- Constant Fields
  - must be in capital letters only
  - for eg -

```
final double PI = 3.14;  
final double WEEKDAYS = 7;  
final String COMPANY_NAME = "Subeem Infotech";
```

- package names
  - package names should be lower case only
  - for eg -> java.lang

## comments

```
// Single line comment.  
/* Multi line comments */  
/** Documentation comments */
```

## 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)
37. strictfp - A Java keyword is used to restrict the precision and rounding of floating-point calculations to ensure portability.
38. super - Refers to a class's base class (used in a method or class constructor)
39. switch - A statement that executes code based on a test value
40. synchronized - Specifies critical sections or methods in multithreaded code
41. this - Refers to the current object in a method or constructor
42. throw - Creates an exception

- 43. throws - Indicates what exceptions may be thrown by a method
- 44. transient - Specifies that a variable is not part of an object's persistent state
- 45. try - Starts a block of code that will be tested for exceptions
- 46. void - Specifies that a method does not have a return value
- 47. volatile - This indicates that a variable may change asynchronously
- 48. while - Starts a while loop
- 49. goto, const - Unused keywords
- 50. true, false, null - Literals (Reserved words)

## DataTypes

- It defines 3 things
  - 1. Nature (type of data stored)
  - 2. Memory (Memory required to store the data)
  - 3. Operations (what operations we can perform)
- Java is Strictly type checked language
- In java, data types are classified as:
  - 1. Primitive types or Value types
  - 2. 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
```

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

## Literals

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

```
int num1 = 10; // Integral

float num2 = 123.456f; // floating point

char ch = 'c'; // character literal

String name = "sunbeam";// string literal

boolean status = true or false; // boolean literal

String s = null;// 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.
  - representation of floating-point numbers using 32 bits.
  - single precision is known as "binary32".
  - typically provide about 7 decimal digits of precision.
- Double precision (8 bytes) floating-point number. Suffix d or D.
  - representation of floating-point numbers using 64 bits.
  - double precision is known as "binary64".
  - typically provide about 15-16 decimal digits of precision.
- 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 languages.
- 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;
```

## Variable

- 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.

## Java Method

- A method is a block of code (definition). Executes when it is called (method call).
- Method may take inputs known as parameters.
- Method may yield a output known as return value.
- Method is a logical set of instructions and can be called multiple times (reusability).
- Functions in C/CPP are called as Method in java.

## Class,Reference and Object

### class

- Logical entity
- blueprint of an object
- consists of fields and methods
- it is a reference type in java

### Object

- physical entity
- instance of the class
- It defines three things
  - 1. state
  - 2. Behaviour
  - 3. identity
- the objects are created using new operator
- objects of the class are always created on heap.
- process of creating object of the class is also called as instantiation

### Reference

- class is a reference type in java.
- variable created of a class is called as reference.
- reference variables get space on the stack.
- reference variables stores the address

### Points to remember

- in java all local variables need to be initialized before using
- class fields are assigned with the default values
- primitive types are assigned 0 while references are assigned null



## Operators

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

## widening & Narrowing

- converting state of primitive value of narrower type into wider type is called as widening

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

- converting state of primitive value of wider type into narrow type is called as 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.

## Wrapper classes

- In Java primitive types are not classes. So their variables are not objects.
- Java has wrapper class corresponding to each primitive type. Their variables are objects.
- All wrapper classes are final classes i.e we cannot extend it.
- All wrapper classes are declared in java.lang package.

```
Object  
|- Boolean  
|- Character  
|- Number  
    |- Byte  
    |- Short  
    |- Integer  
    |- Long  
    |- Float  
    |- Double
```

## Applications of wrapper classes

1. Use primitive values like objects

```
// int 123 converted to Integer object holding 123.  
Integer i = new Integer(123);
```

2. Convert types

```
Integer i = new Integer(123);
byte b = i.byteValue();
long l = i.longValue();
short s = i.shortValue();
double d = i.doubleValue();
String str = i.toString();
String val = "-12345";
int num = Integer.parseInt(val);
```

### 3. Get size and range of primitive types

```
System.out.printf("int size: %d bytes = %d bits\n", Integer.BYTES,
Integer.SIZE);
System.out.printf("int max: %d, min: %d\n", Integer.MAX_VALUE,
Integer.MIN_VALUE);
```

### 4. Helper/utility methods

```
System.out.println("Sum = " + Integer.sum(22, 7));
System.out.println("Max = " + Integer.max(22, 7));
System.out.println("Min = " + Integer.min(22, 7));
```

- 5. Java collections only store object types and not primitive types

## Boxing & UnBoxing

- Converting value from primitive type to reference type is called as boxing
- Converting value from reference type to primitive type is called as unboxing

```
int num1 = 10;
Integer i1 = new Integer(num1); // boxing
Integer i2 = num1; // auto-boxing

Integer i3 = new Integer(20);
int num2 = i3.intValue(); // unboxing
int num3 = i3; // auto-unboxing
```

## Labwork

1. Scanner and (Console-> Option )demo
2. class, Object, reference
3. Data types (size, max,min)
4. widening and narrowing
5. Wrapper classes

6. Boxing & unBoxing

7. Assignments