Lecture 8

Data Types:- Data type defines the type/kind of data a variable can store. Data Type also determines the size of data that a variable can store

e.g:- `int a = 15;`

`float b = 10.20f;`

→ Here a is a variable which can store only integer value and if we try to store string ~~or character~~ then it will give error.

IMP And because of this reason Java is referred to as a statically typed language as the type of the variable is static once defined means type can not be changed.

* And this type checking is performed at compile time only So it is referred as static type checking

IMP But in some languages like Python, Javascript same variable can hold different type of data means at one point of time in a program it can hold numeric value while later at some other point it can hold string value. And because of this such languages are known as dynamically typed languages. as the type of the variable is dynamic

eg: in python `x = 5;`

`x = "Jenny";` } both are allowed.

* here the type is determined or checked at run time based on the value assigned to the variable So this is called Dynamic type checking

Static type checking is good in some cases because it helps in earlier detection of programming errors as it allows variable types to be checked at compile time itself.

→ But in dynamic checking, majority of type checking is performed at runtime So we detect errors later.

e.g:-

in python we can write:	in Java:
a = 5;	int a = 5;
:	:
a = "Jenny";	a = "Jenny"; ⇒ error at compile time
:	
ab = a - 1; → at runtime checking would be done & it would give error	

→ But in java reassignment (a = "Jenny") would give compilation error as type of a was declared int in java. So we can't store a string literal here

→ Java has 2 main categories of data types:

① Primitive Types
② Non-Primitive Types (More complex that refer to objects)

① Primitive :- built-in data types that represents simple values.
there are 8 primitive data types in Java

- boolean (true, false)
- Number
  - integer → byte, short, int, long ⇒ Contains whole numbers
  - Floating-point → float, double
  - character → char → internally char is also represented as an unsigned integer

Question:- Why there are 4 different types to store an integer?

Answer Main purpose was to allow programmers to optimize memory usage & handle different range of values.

→ Each of these types has a different range of values and occupies different amounts of memory. So it provides the flexibility on the requirements of the program.

Real-life example:-

In kitchen we have different - different size of containers.

→ Variables are like containers only. So we have one large container only but we want to store small data (number) then it would be wastage of memory. And if we have a single small container for large integers then it would be difficult to store large number.

Hence the data types (byte, short, int, long) has different sizes but they are represent integers

① **Integers Data types:-** whole numbers | fixed-point numbers

e.g:- 10, -5

Type	Size/bit depth	Value range	Default value
Byte	8 bits	-128 to 127 (-2^7 to 2^7-1)	0
Short	16 bits	-32768 to 32767 (-2^{15} to $2^{15}-1$)	0
int	32 bits	-2147483648 to 2147483647 (-2^{31} to $2^{31}-1$)	0
long	64 bits	-2^{63} to $2^{63}-1$	0

Note - Default values is assigned to the instance variables only not to the local variables.

Computers use 2's Complement Representation for signed integers

Practice Time:-

```
Class Student {
    byte rollNo = 22;        // integer literal (int literal) ⇒ ok, No error
    byte jeeRank = 200;      // ⇒ will give error.
    int phoneNo = 9981012310; // ⇒ will give error.
    short jeeRank = 200      // ⇒ ok, no error
    long phoneNo = 9981012310 // ⇒ error (integer number too large)
    int age = 31L;           // ⇒ error (long literal)
    int age = 31;            // ⇒ No error
    long largeNumber = 12567; // ⇒ No error
    void display()
    {
        System.out.println(rollNo)
    }
}
```

Java still treats this no. as integer so we have to write 'L' or 'l' as suffix to tell Java to treat it as long literal

Java will convert int to long and no error is there No need to specify L as suffix in this case.

We can separate large numbers with an underscore

eg:- `long largeNumber = 9_123_507_678_127L;`

Short intro of wrapper class:-

```
int minValue = Integer.MIN_VALUE;   // Integer → wrapper class for int.
int maxValue = Integer.MAX_VALUE;
```

When you print these:-

```
System.out.println(minValue);  ⇒ -2147483648
System.out.println(maxValue);  ⇒ 2147483647
```

→ Wrapper class provides a way to use primitive data types (int, char, boolean, byte etc.) as objects.

→ For each primitive type we have wrapper class in Java.

Need of wrapper class is in the collection frameworks such as Arraylist and vectors, Hashmap, they store only objects (reference types) not primitive types. And like this there are so many uses of wrapper class but we will discuss this thing in detail in later videos

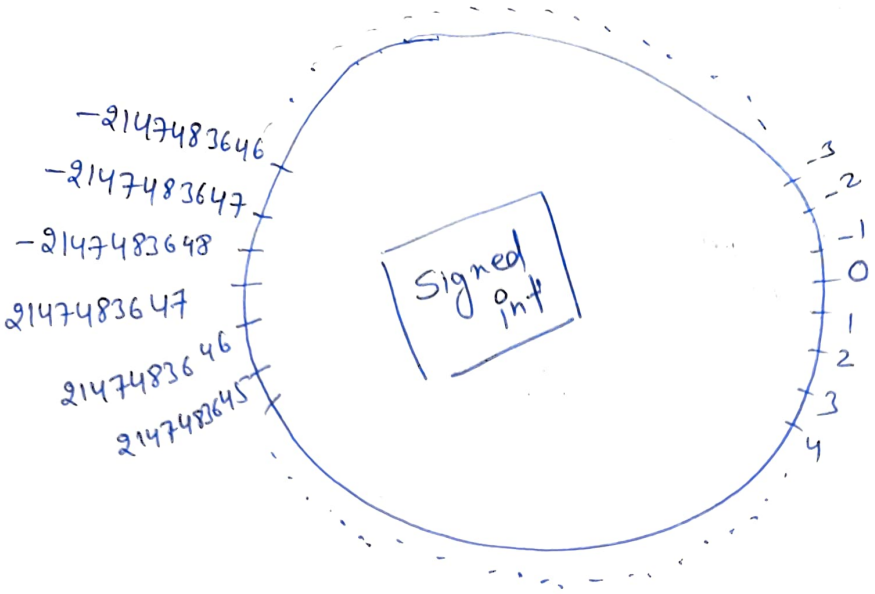
Primitive Types	Wrapper Class
int	Integer
byte	Byte
short	Short
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Something interesting (Overflow and Underflow) :-

```
int minValue = Integer.MIN_VALUE;
int maxValue = Integer.MAX_VALUE;
```

System.out.println(MIN_VALUE - 1); ⇒ 2147483647
System.out.println(MAX_VALUE + 1); ⇒ -2147483648

↳ Overflow } ⇒ this happens when the result of an arithmetic operation exceeds the maximum or minimum value that can be stored in a particular data type
↳ Underflow



→ When you try to store a value larger than 2147483647 in an int, overflow occurs because the binary representation "wraps around"

→ Java uses twos complement representation for signed integers. In this, when a number exceeds the maximum positive value, it wraps around to the minimum negative value.

e.g:-
```
int a = 2147483647
a = a+1;
System.out.println(a); ⇒ -2147483648
```

2147483647 → 0111 1111 1111 1111 1111 1111 1111 1111

after adding 1 → 1000 0000 0000 0000 0000 0000 0000 0000

↓

This is binary representation of -2147483648 in 2's complement.

⇒ We can use some other Integer literal also other than Decimal points like

- Octal
- Hexadecimal
- Binary

Octal :- Combination of digits from the set 0 through 7, with a leading 0

e.g :- 031, 0, 0431

Hexadecimal :- A sequence of digits preceded by 0x or 0X is considered as hexadecimal integer. They may includes alphabets A through F or a through f. Here A = 10, B = 11, C = 12, D = 13, E = 14, F = 15

e.g :- 0x21, 0xABC, 0X789A etc.

Binary :- preceded by 0b or 0B and having combination of 0s & 1s only

eg: 0B1001, 0b010101 etc.

Practice Time :-

```
public class Demo{
    int octNumber = 037;
    int hexNumber = 0X239;
    int binNumber = 0B101;

    void display(){
        System.out.println("octNumber = " + octNumber);
```

```
        System.out.println("hexNumber = " + hexNumber);
        System.out.println("binNumber = " + binNumber);
    }
    public static void main(String[] args)
    {
        Demo d = new Demo();
        d.display();
    }
}
```

Output :-

```
octNumber = 31
hexNumber = 570
binNumber = 5
```

NOTE :- We generally don't use these formats. We use only decimal format.

Floating-point Numbers :- In Java we have 2 data types to represent floating point numbers :- float & double

They are known as real-numbers as well.

eg:- 3.14, 10.125, -0.1402 etc.

Floating point numbers are used when we need more precision in calculations.

e.g. - financial calculations,
Calculating interest rates, tax calculations

Scientific Calculations

Temperature measurements like 98.6 or time values like 2.5 seconds.

Type	Size	Range	Precision	Default
float	32 bits	-3.4e38 to 3.4e38	6-7 decimal digits	0.0f
double	64 bits	-1.7e308 to 1.7e308	15-16 decimal digits	0.0d

Scientific notation

eg:- float marks = 98.9; ⇒ error

float marks = 98.9f; ⇒ No error ok

NOTE:- in Java by default data type for decimal numbers is double

So 98.9 will be considered as double but we are declaring it as float. So it gives error.

To specify that the decimal number is float, we have to add 'f' as suffix to the number like 98.9f.

f can be small or capital (F)

→ double is more precise. That's why it is default type for floating point numbers

→ Smallest positive non-zero value for float = 1.4e-45

$= 1.4 \times 10^{-45}$

Smallest positive non-zero value for double = 4.9e-324

$= 4.9 \times 10^{-324}$

Practice Time

float marks = 98; ⇒ No error

↳ 98.0 will be printed

float marks = 98.9; → will give error

↓

to fix this error we can either add 'f' as suffix or type cast this

→ float marks = 98.9f;

→ float marks = (float) 98.9;

} both are correct but generally programmers prefer the first one

double marks = 98; } Correct, No error

double marks = 98.9; } Correct, No error

double marks = 98.9d; } Correct, No error

double x = 1.7e308 Correct.

double d = 1.1234567890123456789; Prints → 1.1234567890123457

float f = 1.1234567890123456789; Prints → 1.1234568

Output

upto 7 decimal points

upto 16 decimal points

NOTE: → So here in the output number is not accurate but its approximate number. But double is more precise than float.

So double is default type for floating point numbers & int is default type for integer numbers

→ But we don't use float & double when we want more precise calculations eg:- in billing or in some financial calculations. In Java there is class bigDecimal that would be used in more precise calculations

Free Question :- Convert your height (feet) to meters

eg:- Your height is : 5.3 feet

in meters : ? (5.3 * 0.3048)

Take / declare a variable of type double & assign your height value to it.

Now convert your height into meters.

Now Print.

Solution :-

```
public Class HeightConverter {
    double myHeight = 5.3d;
    double result;
    void compute() {
        result = myHeight * 0.3048;
        System.out.println(result);
    }
public static void main(String[] args) {
    HeightConverter obj = new HeightConverter();
        obj.compute();
    }
}
```

Characters (char data Type) :- To represent a single character we use char data type.

e.g:- 'A', 'a', '#', '1', ' '

```
Char myCharVariable = 'A';
Char myCharVariable = A;  // error, not allowed without single quotes
```

A char can store any character from the Unicode Character Set including letters, digits, symbols & special characters

Type	Size	Range	Default Value
char	16 bits	0-65535 (0 to $2^{16}-1$)	'\u0000'

Internally char is represented as integers (unsigned integers) in memory

e.g:- char myCharVariable = 'A';

here A is represented as 65 internally

Curious? - We deal only with few characters like A-Z, a-z, 0-9, & some symbols like ($, ! # & * -- etc)

Why do we need a range of 0-65535 for characters?

→ In world there are so many languages & each language has its own characters & symbols.

Que How did these characters & symbols are represented internally in Computer and digital devices?

Ans. We know the characters are represented as integers internally So there is a mapping between the character & the integer value stored internally & this mapping is defined by Unicode Character Set.

The Unicode character Set is a standardized collection of characters that includes letters, Digits, Symbols, emojis, Control characters of all the languages across the world.

Unicode is a standard specification and it has different encoding schemes (implementation) like UTF-8, UTF-16, UTF-32

Java follows UTF-16 encoding scheme

So this encoding scheme provides a unique number (or code point) for every character across different writing systems, symbols, emojis, allowing for consistant representation and manipulation of text in computers & digital devices.

→ e.g:- A is encoded as U+0041 in UTF-16 encoding scheme

and in ASCII it has a value 65.

A → 0041 → 0000 0000 01000001 (65)

char literal — UTF-16 encoding — Binary form — decimal value

Code point:- every character is assigned a unique code point represented in the format U+XXXX where XXXX is a hexadecimal number

We can also write:- (u should be in small letter)

char myCharVariable = '\u0041'; ⟹ will print A

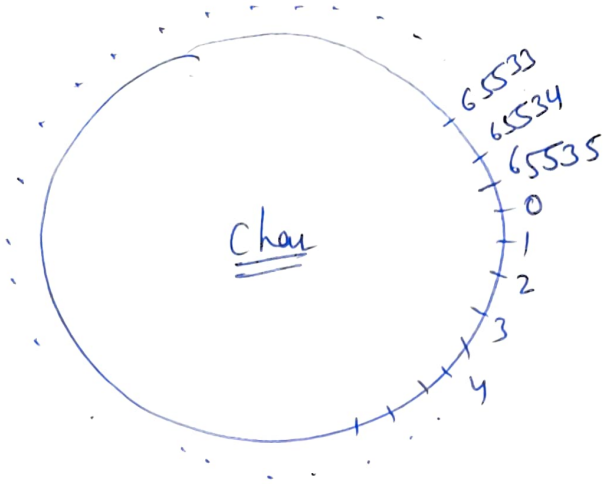
or

char myCharVariable = 65 ⟹ A

Practice Time :-

```
char myCharVariable = 'Aa';   // invalid as only one character is allowed
char myCharVariable = '\u004';  // invalid
char myCharVariable = 65537;  // invalid. gives error.
Char myCharVariable = char(65570);  // valid.
                                    ↳ Prints " (double quotes)
int x = 'A';  // valid
   ↳ Print 65.
```

Wrap around concept



Practice Question:-

① Print min value & max value for char type using Wrapper class.

Sol^n:-

```
public class CharDemo {

    char myMinCharVariable = Character.MIN_VALUE;
    char myMaxCharVariable = Character.MAX_VALUE;
    int a = (int)myMinCharVariable;
    int b = (int)myMinCharVariable;
    public static void main (String[] args) {
        CharDemo obj = new CharDemo();
        System.out.println("Minimum value = " + a);
        System.out.println("Maximum value = " + b);
```

3

3

IMP NOTE:- We can initialize a char variable in 3 ways

(i) initializing with a char literal [char ch = 'A']

(ii) initializing with an int literal [char ch = 65]

(iii) initializing with a unicode escape seq. [char ch = '\u0041']

and internally char is stored as 16-bit unsigned integer

Boolean Data Type:- Used to represent two logical values either true or false

true & false are boolean literals
Default value of a boolean variable is false.
Wrapper class of boolean type is Boolean
Size/ Bit depth actually depends on JVM.

→ It is primarily used in control flow statements (conditional statements) and expressions to control the flow of the program based on certain conditions.

declaration & initialization:-

```
boolean isJavaEasy = false;
```

e.g:-

```
int a = 5;
int b = 10;
boolean result = (a<b);
System.out.println("Is a less than b? " +result);
```

So boolean data type is a fundamental data type that represents true/false and is essential for decision-making in programs.

Some More Interesting facts (Info) about byte, short, int & long

→
```
byte minValue = Byte.MIN_VALUE;
byte a = minValue - 1; → will give error, will not wrap around
```
Why?

Ans is:- When we perform authmetic operation on byte, short or char the operands are automatically promoted to int, regardless of their original type. This promotion is done to ensure that the authmetic operation is performed with sufficient precision.

→ Since the operands are promoted to int, the result is also an int Java doesnot automatically downcast the result back to byte or short because doing so could lead to loss of data (truncation) That's Y you are getting an error.

~~short shortMinValue = Byte.MIN_VALUE;~~

```
short shortMinValue = Short.MIN_VALUE;
short b = shortMinValue - 1; → error, no wrap around
```

But
```
int intMinValue = Integer.MIN_VALUE;
int x = intMinValue - 1;
long longMinValue = Long.MIN_VALUE;
long y = longMinValue - 1;
```
⇒ works fine & wraps around would be done automatically.

Solution of the problem, with byte & short is we can do casting.

e.g:-
```
byte minValue = Byte.MIN_VALUE;
byte a = (byte)(minValue - 1);
```