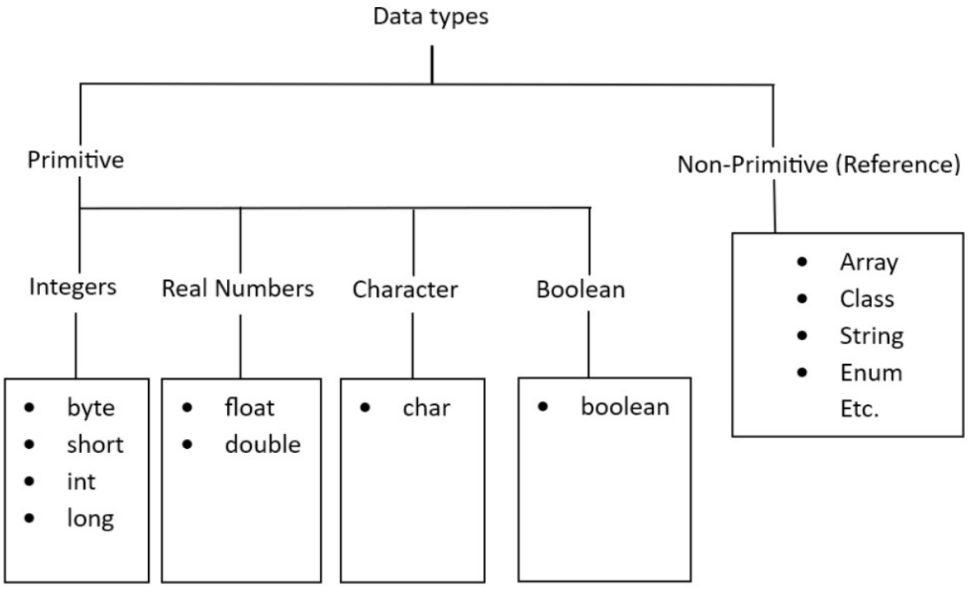
Values and data types (module 2)

Data Types:



Non- primitive types are also known as Composite type or reference type

Data Types, their Sizes, and default values

Data Type	size in bytes	size in bits	default value
byte	1	8	0
char	2	16	'\u0000'
short	2	16	0
int	4	32	0
long	8	64	0 or 0L
float	4	32	0 or 0.0f
double	8	64	0 or 0.0d

Implicit Type Conversion/Casting or Coercion

Explicit Type Conversion/Casting

1 byte = 8 bits

In Java, the **size of a boolean** is **not precisely defined** in terms of memory because its actual storage depends on how the JVM implements it. However, When used **individually**, a boolean **typically takes 1 byte (8 bits)** in memory.

Range of data types:

If a data type's size is k bits then its range is : -2^{k-1} to $2^{k-1}-1$

Range of byte: -128 to 127
Range of short: -2^{15} to $2^{15}-1$ (-32,768 to 32,767)
Range of int: -2^{31} to $2^{31}-1$ (-2,147,483,648 to 2,147,483,647)
Range of long: -2^{63} to $2^{63}-1$ (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
Range of float: $\approx -3.4 \times 10^{38}$ to 3.4×10^{38}
Range of double: $\approx -1.7 \times 10^{308}$ to 1.7×10^{308}

**DO NOT MEMORIZE THIS. (KEEP THE FORMULA IN MIND)
** ALWAYS STORE PHONE NUMBERS IN USING long data type
In modern java, Floating-point types (float, double) store values in **IEEE 754 format** and have a much wider range.

Suffix for literals

1. f or F (Float Literal)

- Used to denote a **float** literal.
- Without it, decimal numbers are treated as **double** by default.
- 15.5 is double by default, but we can use f or F as suffix to make it float: 15.5f or 15.5F

2. d or D (Double Literal)

- Used to denote a **double** literal.
- **Optional**, because decimal numbers are treated as double by default.
- 15.5 is double by default, We can write 15.5D or 15.5d but it is optional as it's already double

3. l or L (Long Literal)

- Used to denote a **long** literal.
- Without it, integer literals are treated as **int** by default.
- L is preferred over l(small L) because l looks similar to 1 (digit one).
- Example: 15L , 120L, 15 is int by default but 15L is 15 as long data type.

Keywords in java:

Java has **52 keywords**, which are **reserved words** that cannot be used as variable names, class names, or method names. (DO NOT MEMORIZE)

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

Note: goto and const are **reserved but not used (**no functionality) in Java.

Literals:

Integer Literals: 10, 20, -50, 0
Real Numbers/ Floating point literals: 12.5, -50.5, 0.0, .7
Character Literals: 'A' , 'B', 'C', '#', '*' , '5'
String Literals: "Hello", "Yo", "I am Iron Man"
Boolean Literal: true , false
null literal: null

Identifiers: int ***a***=10;

class ***Apple***{

}

Identifier/Variable Naming Rules:

Can contain letters (A-Z, a-z), digits (0-9), _ (underscore), and $ (dollar sign).
Must **start with a letter, _, or $** (cannot start with a digit).
Cannot be a **Java reserved keyword** (e.g., int, class, public).
Java is **case-sensitive** (myVar and myvar are different).
Must not contain space

Variable Declaration:

int a,b,c; or int a,b,c;

Variable pre-declaration and initialization:

```
int a,b,c;
a=10;
b=20;
c=30;
```

Declaration and initialization

```
int a=10, b=20, c=30;
```

Chained Initialization:

```
int a = 10, b, c, d;
b = c = d = a;
```

**For chained initialization, all variables must be declared beforehand.

ASCII (American Standard Code For Information Interchange)

‘A’ to ‘Z’ : 65 to 90
‘a’ to ‘z’ : 97 to 122
‘0’ to ‘9’ : 48 to 57
Space : 32

Type Conversion/ Type Casting:

(1) Implicit type conversion: smaller data type to larger data type

Example 1:
```
char ch= ‘A’;
int b=ch;
```

Example 2:
```
int a=10;
float f=a;
```

Example 3:
```
double b= 10;
```

Example 4:
```
int a= ‘A’;
```

(2) Explicit type casting: larger type to smaller type (Lossy conversion possible)

Example 1:
```
int a= (int) 13.5;
```

Example 2:
```
char ch= (char) 66;
```

Example 3;
```
float f=(float) 15.6d;
```

Example 3.
```
int a= (int) 5L;
```

Data type of result operations: The result of an expression with multiple data types always gets promoted to the largest type among operands.

int a=10; float f= 10.4; ***double*** k=12.4;
double k=a+f+k;

int p=12; ***float*** m=2.5f; char ch= 'A';
float f= p+m+ch;

int a=10, b=20, c=30;
int k= a+b+c;

However, if a non-primitive type is involved, result is stored in non-primitive one.
Example: int a=10; double d=20.5; **String** t= "Hello";
String s= a+d+t;

***Exception*:** char + char = int

char ch1= 'A', ch2= 'B';
System.out.println(ch1+ch2); output: 131 (65 + 66)

That means to store char + char in a char variable, we need explicit type casting
char ch = (char) 'A' + 'B';

***Exception on Exception* (Updation using +=)**

+= updates a variable

char ch= 'A';
ch+= 2;
System.out.println(ch); output: C

Safe side:
instead of writing ch=ch+2, write **ch+=2** as sometimes you might get error due old java version.

Write either ch+=2 or ch= (char) (ch+2);

It's better not writing ch=ch+2 without explicit casting, but you can surely write ch+=2 without any worry.

** We will discuss about += (shorthand/ Arithmetic-Assignment operators) in next chapter.

Initialization:

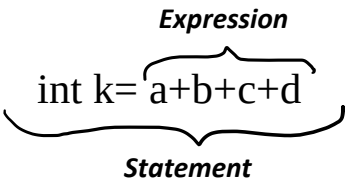
(1) Static Initialisation: directly assigning values:

```
int a=10;
float f=1.4f;
```

(2) Dynamic Initialisation: getting value through a process

```
int a= 5+5;
float f= 1.0f + 0.4f;
double d= 15.6+12.5;
```

Statement and expression:



Type of expressions:

(1) Pure Expression: Involves data of same type

Example 1:

```
int a=10, b=15, c=20;
a+b+c
```

Example 2: 12.3 + 15.5 + 0.7

(2) Impure Expression: Involves data of more than one data type.

Example 1:

```
int a=10; float b=15.5f; double c=20.0, k;
a+b+c
```

Example 2: 10 + 12.7 + 'A'

* Separators: comma ,
  parenthesis ( )
  Curly braces { }
  Square Brackets [ ]

* Punctuators: Terminator ;
  Member operator **.** **(dot)**
  Ternary operator ?

21/03/2025

Subhasis Das