CHAPTER 2: VARIABLES AND DATATYPES

2.1 Variables

What are variables?

Variables are identifiers that are used to store data values. They are used to reserve a storage space, so that data can be accessed and modified.

2.2 Variable Naming Convention

In order to create variables in java, certain rules or conventions must be followed. They are:

- Must begin with either letters (A-Z or a-z), dollar sign ($) or underscore (_).
- They are case sensitive, i.e.- “name” and “Name” are two separate variables.
- Cannot have any white spaces.
- Can have numbers (0-9).
- Cannot begin with a number or any special signs.
- Can use **camel casing** (yourFirstName) or **snake casing** (your_first_name) for multiple words in a variable.
- Cannot be a reserved keyword (table given below).
- Should be meaningful.
- Should not be a single character.

Valid		Invalid	
a123bc	abc123	123Abc	ab cd
first_name	firstName	first name	Ab.cd
Int	$ami	#number	if
_ami	class_1	class	int

Always try to keep your variable names meaningful and close to the context. Otherwise, the code will not be readable by you or anyone else.

2.3 Reserved Keywords in Java

The following table contains a list of words that have a special meaning in java and cannot be used as the name of an identifier or a variable. Java is a case-sensitive language. Therefore, you can use the reserved words after changing the case of any of the characters. However, this is not recommended. While writing your code, you should be cautious towards not using any of the reserved keywords.

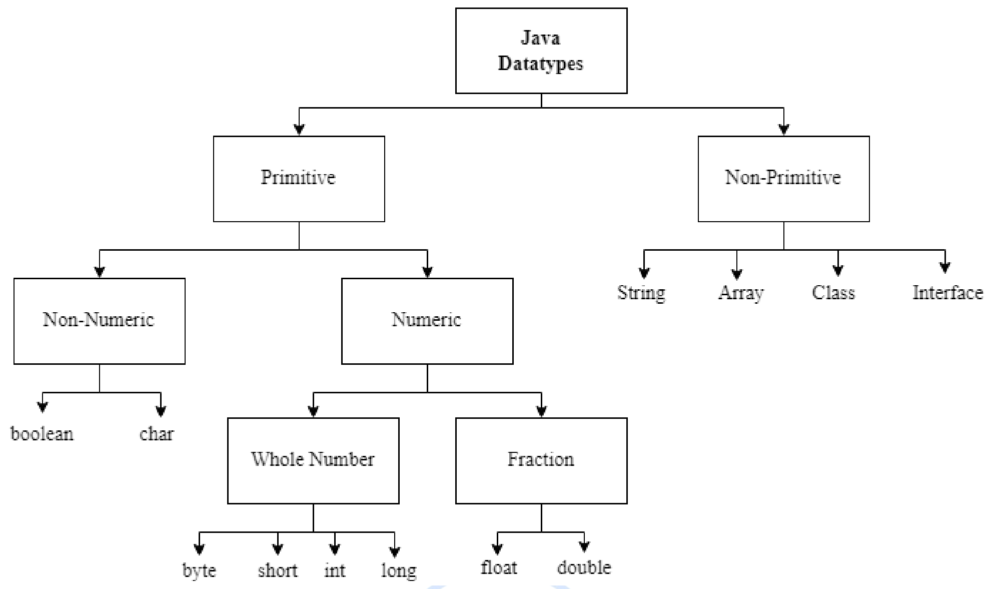
abstract	assert	boolean	synchronized	byte	case	catch	char	class
continue	default	do	double	else	enum	extends	final	finally
float	for	if	implements	import	int	long	native	new
null	package	private	instanceOf	public	return	short	static	strictfp
super	switch	break	interface	this	throw	throws	transient	try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

*New in Java 9 | **New in Java 10

2.4 DATATYPES

Every variable in Java must be of a specific datatype. Datatypes can be categorized as follows:

- Primitive Data Types: Holds a value, represents the size and type of a variable
- Non-Primitive Data Types: Holds a reference to an object or location



Category	Name	Size (bytes)	Possible Values/Explanation	Examples
Primitive	**byte**	1	Whole numbers from -128 or – (2^7) to 127 or (2^7)-1	byte b1 = 51;
	short	2	Whole numbers from -32,768 or – (2^15) to 32,767 or (2^15)-1	short s1 = 513;
	int	4	Whole numbers from -2,147,483,648 or – (2 ^31) to 2,147,483,647 or (2^31)-1	int i1 = 5456;
	long	8	Whole numbers from -2^63 to 2^63-1	long l1= 12345788;
	float	4	Fractional numbers with at most 6 to 7 decimal digits	float f1= 5.0f;
	double	8	Fractional numbers with at most 15 decimal digits	double d1= 123.4558428743d;
	boolean	1	Either true or false	boolean b1 = true;
	char	2	A single character from the ASCII code	char c1 = 'A';
Non-Primitive	**String**	Same size	A sequence of characters	String s1 = "Hello";
	Arrays		Multiple data of the same datatype	int [ ] arr = {1, 2, 3};

Double and double, Integer and int, Float and float all exist in Java. However, Double, Integer, and Float etc are non-primitive datatypes (classes) where double, int and float are primitive. We shall learn more about classes and interfaces later on.

2.5 PRIMITIVE VS NON-PRIMITIVE

The differences of primitive and non-primitive datatype are shown below:

Primitive	Non-Primitive

Predefined, no need to create before using	Must be created (except for String) before using
Contains a value	Contains a reference
Cannot be null	Can be null
The size of a primitive type depends on the data type	All non-primitive types have the same size.

2.6 Declaration, Assignment and Initialization

Declaration: Allocating memory space while specifying the datatype of a variable.

Assignment: Assigning a value to a variable.

Initialization: Allocating memory space while specifying the datatype and assigning a value.

int a; //declaration
a= 5; //assignment
	int a = 5; //initialization

How to initialize a variable?

<datatype> <variable name> = <value> < format specifiers >; // format specifiers are only for double and float

Examples:
byte b1 = 51;
short s1 = 513;
int i1 = 5456;
long l1= 12345788;
float f1= 5.0f;
double d1= 123.4558428743d;
boolean b1 = true;
char c1 = 'A';
String s1 = "Hello";
int [] a= {10, 20, 100};

If you paid attention, you should have noticed that for float and double there are an f and d respectively after the values. These are format specifiers for float and double literals. You do not need format specifiers for any other datatype. Values of char datatype are enclosed in single quotation (') and values of String datatype are enclosed in double quotation ("). Length of a String can be 1 or more but char can have only one character.

String a = "ABC"; //Valid	byte a = 1234; //Invalid
String b = "A"; //Valid	byte a = 123; //Valid
String c = 'A'; //Invalid, wrong quotation	int a = 12.3; //Invalid
char d = 'C'; //Valid	float a = 12.3; //Invalid, f missing
char e = "C"; //Invalid, wrong quotation	double a = 12.3f; //Valid, datatype converted
char f = "ABC"; //Invalid, more than 1 character	float a = 12.3d; //Invalid, lossy conversion
boolean b= True; //Invalid, T should be smaller case	int [] a = {5.3, 1, true} //Invalid, all data must be of int type

~ Self-Study ~

Refer to the table above and experiment to understand why some of the lines were invalid. You shall learn more about lossy conversion in 2.7 Datatype Conversion (Type Casting).

2.7 Datatype Conversion Basics

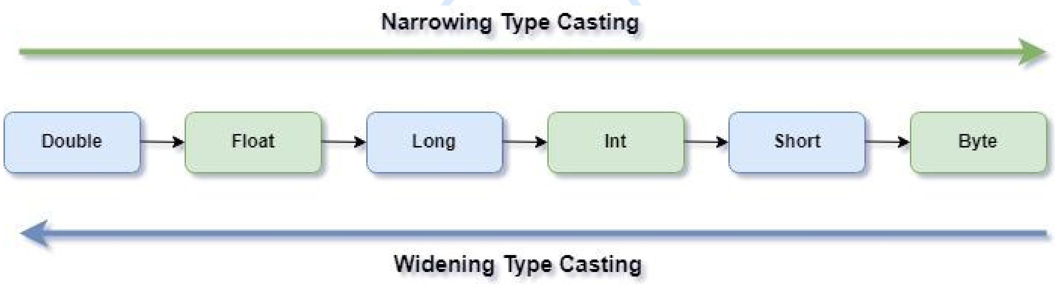
Type refers to the data type of a variable or entity and casting is the process of changing an entity's data type to another datatype. Moreover, type casting is the process of converting a value between two different data types. Only the data type can be changed by casting; the data itself cannot be changed.

In Java, there are two types of casting:

- Widening/Implicit Casting
- Narrowing/Explicit Casting

Widening/Implicit Casting:

Converting a lower data type to a higher data type (from right to left in the following diagram) is known as widening typecasting. As a result, there is no data loss. Since Java does this type of casting automatically and without any explicit coding, it is often referred to as automatic type casting. To implement widening type casting, we do not need to create any new code or modify any of the ones we already have. Example:
int num = 10;
double data = num;



Narrowing/Explicit Casting:
Narrowing type casting is the process of converting higher data types into lower data types (from left to right in the following diagram). As we convert larger size data types into smaller ones, data loss occurs. Because of this, the conversion has been made manual rather than automatic. This type conversion is also known as Explicit type conversion. Example:
double num = 10.99;
int data = (int)num;

Data Type Form/To	Byte	Char	Short	Int	Long	Float	Double
Byte	Auto Assignable	Cast Needed	Cast Needed	Cast Needed	Cast Needed	Cast Needed	Cast Needed
Char	Cast Needed	Auto Assignable	Cast Needed	Cast Needed	Cast Needed	Cast Needed	Cast Needed
Short	Auto Assignable	Cast Needed	Auto Assignable	Cast Needed	Cast Needed	Cast Needed	Cast Needed
Int	Auto Assignable	Auto Assignable	Auto Assignable	Auto Assignable	Cast Needed	Cast Needed	Cast Needed

Long	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Cast Needed	Cast Needed
Float	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Cast Needed
Double	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e	Auto Assignabl e

Here, auto assignable represents the widening or implicit type casting and cast needed represents the narrowing or explicit type casting.

2.8 Numeric-to-Numeric Conversion

Conversio n	Type	Code	Explanation

int to float and double	Widenin g/ Implicit	int num = 10;
		double data = num;
		//10.0
		float data2= num;
		//10.0
			Assigning the int type variable named num to a double type variable named data. Here, Java first converts the int type data into the double type. And then assigns it to the double variable.

double and float to int	Narrowin g/ Explicit	double num = 10.99d;
		float num2 = 10.25f;
		int data = (int)num;
		//10
		int data2= (int)num2
		//10
			Assigning the double type variable named num to an int type variable named data. Here, the int keyword inside the parentheses indicates that the num variable is converted into the int data type.

float to double	Widenin g/ Implicit	float num = 127.45f;
		double d = (double) num;
		System.out.println(d);
		// 127.44999694824219
			Assigning the float type variable named num to a double type variable named d. Here, Java first converts the float type data into the double type. And then assigns it to the double variable.

double to float	Narrowin g/ Explicit	double num=1.123456789d;
		float f= (float)num;
		System.out.println(f);
		// 1.1234568
			Assigning the double type variable named num to a float type variable named f. Here, the float keyword inside the parentheses indicates that the num variable is converted into the float data type.

int to byte	Narrowin g/ Explicit	int num = 127;
		byte b = (byte) num;
		System.out.println(b);
		//127
			This will only work for int values between -128 to 127. Otherwise, it will give an error because byte can only store numbers from -128 to 127.

2.9 Numeric-to-Non-Numeric and Vise-Versa Conversion

Conversio n	Code	Explanation

integer to String	String x= String.valueOf(15);
	String y= Integer.toString(20);
	System.out.println(x+y);
	//1520
		String.valueOf() and **Integer.toString()** both converts the int to a String and returns it.
		Therefore, the output is "15"+"20"="1520".

String to Integer	int n = **Integer.parseInt("25");** int m = Integer.valueOf("35"); System.out.println(m+n); //60	**Integer.parseInt()** method returns the string as a primitive type int. If the string does not contain a valid integer, then it will throw a NumberFormatException. **Integer.valueOf()** method returns the string as an integer object. If the string does not contain a valid integer, then it will also throw a NumberFormatException. Therefore, the output is 25+35= 60.
int to char	int a=65; char c=(char)a; System.out.println(c); //A	The ASCII character of the given value is stored as the character.
char to int	char a='Z'; int c=(int)a; System.out.println(c); //90	The ASCII value of the given character is stored as the integer.

You will learn how to convert char to String and vise-versa later on in this course in Chapter 9: Strings. You are also not familiar with how methods work. For now, just keep in mind that methods may take a certain value, perform certain operations, and may return a different value.

~ Self-Study ~

Experiment on every possible kind of datatype conversion. Look up lossy conversion on the internet.

2.10 WORKSHEET

A. Trace the following code and write the outputs.

Code	Output
class B { public static void main(String[] args) {	

int a= 15; //Value of a is 5
int b= 4; //Value of b is 7	

System.out.println(a+b);
float a1= a;
float b1= b;	

String a2= Integer.toString(a);
String b2= Integer.toString(b);	

System.out.println(a1+b1);	
System.out.println(a1 == a);	

System.out.println(a2+b2);
System.out.println(a2+"Hi"+b2);	

System.out.println("Hi"+Integer.toString(b+1));
}
}	

B. Imagine you have the following lines of codes. Your task is to complete the code so that the desired output is generated.

Code	Output
class A {	12
public static void main(String[] args) {	12.0
int x= 5; //Value of x is 5	57
int y= 7; //Value of y is 7	
System.out.println(x+y);	
//Your code here	
System.out.println(x1+y1);	
System.out.println(x2+y2);	
}	
}	

C. Which of the following are invalid variable names and why?
N1, num2, 2num, num_1, firstName, first1name, hi 1, first_1_name, first-name, import, long, import1

Invalid Variable names	Reason for Being Invalid

D. What is the difference between **declaring**, **assigning** and **initializing** a variable? Can we declare, assign or initialize the same variable multiple times in Java?

E. Explain lossy conversion with code examples.