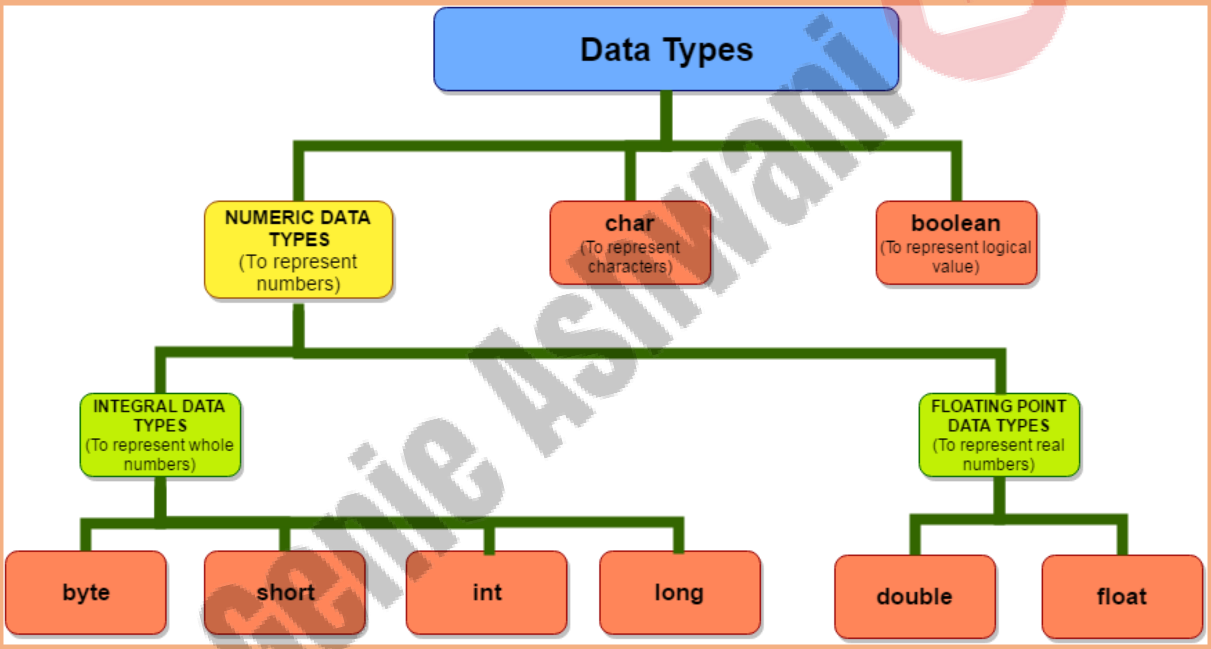
Introduction to Data Types

Data Types:

In Java, data types are used to define the type of data that a variable can hold. They help the compiler understand the kind of operations that can be performed on the data and allocate memory accordingly. Java has two categories of data types: primitive data types and reference data types.


Data Types
NUMERIC DATA TYPES (To represent numbers)
char (To represent characters)
boolean (To represent logical value)
INTEGRAL DATA TYPES (To represent whole numbers)
FLOATING POINT DATA TYPES (To represent real numbers)
byte
short
int
long
double
float

Note:

Except Boolean and char all remaining data types are considered as signed data types because we can represent both "+ve" and"-ve" numbers.

Primitive Data Types:

Integral data types:

byte: 8-bit 1 Byte Range: -128 to 127.

short: 16-bit 2 Byte Range: -32,768 to 32,767.

int: 32-bit 4 Byte Range: -2^31 to 2^31 - 1.

long: 64-bit 8 Byte Range: -2^63 to 2^63 - 1.

Byte

Why -128 to 127 for byte if it takes 8 bit?

A byte in Java is an 8-bit data type. This means it uses 8 binary digits (bits) to represent values. In binary, each bit can have a value of either 0 or 1. Using 8 bits, we have a total of 2^8 = 256 possible combinations.

Two's Complement Representation:

In computers, negative integers are often represented using a scheme called two's complement. In this scheme, the leftmost bit (the most significant bit) represents the sign of the number. If it's 0, the number is positive; if it's 1, the number is negative.

The remaining bits represent the magnitude of the number's absolute value in binary form.

For byte, since we have 8 bits, one bit is used for the sign, leaving 7 bits for the magnitude.

Positive Numbers:

For positive numbers, the sign bit is 0, indicating positivity.

With 7 bits, you can represent values from 0000000 (binary for 0) to 1111111 (binary for 127). This covers a range of 0 to 127.

Negative Numbers:

For negative numbers, the sign bit is 1, indicating negativity.

To represent negative numbers using two's complement, you invert the bits of the positive counterpart (flipping 0s to 1s and 1s to 0s) and add 1 to the result.

In the case of byte, the two's complement of 0000000 (binary for 0) is 10000000, which represents -128.

Putting it all together:

Positive values use 7 bits to represent the range from 0 to 127.

Negative values use 7 bits to represent the range from -1 to -128.

Note : byte data type is best suitable if we are handling data in terms of streams either from the file or from the network.

Example:

```
byte b=10;

byte b2=130;//C.E:possible loss of precision

byte b=10.5;//C.E:possible loss of precision

byte b=true;//C.E:incompatible types

byte b="ashwani";//C.E:incompatible types
```

Short:

The most rarely used data type in java is short.

Size: 2 bytes

Range: -32768 to 32767(-215 to 215-1)

Example:

```
short s=130;

short s1=32788;//C.E:possible loss of precision

short s2=true;//C.E:incompatible types
```

Int:

This is most commonly used data type in java.
Size: 4 bytes

Example:

```
int i=123;
int j=9.6;//C.E:possible loss of precision
int k=true;//C.E:incompatible types
int l=false;//C.E:incompatible types
```

long:

Why log required if have int data type?

Whenever int is not enough to hold big values then we should go for long data type.

Example:

```
int smallNumber = 1000;// Suitable for small to medium-sized values
long bigNumber = 10000000000L; // Used for larger values
```

Floating Point Data types:

Floating-point data types in Java are used to represent real numbers, including those with decimal points. They enable the representation of a wide range of values, from tiny fractions to enormous numbers, while providing a trade-off between precision and storage.

Types of Floating-Point Data:

floating-point data types: float and double.

float:

Uses Size: 4 bytes to store a floating-point number.

Suitable If we want to 5 to 6 decimal places of accuracy then we should go for float.

Example: float pi = 3.14159f;

double:

Uses Size: 8 bytes to store a floating-point number.

Suitable If we want to 14 to 15 decimal places of accuracy then we should go for double.

Generally preferred for most applications unless memory efficiency is crucial.

Example: double distance = 299792458.0;

what is precision?

Precision refers to the level of detail or accuracy in representing numbers, especially decimal numbers. In the context of floating-point numbers, precision determines how many significant digits can be reliably stored and manipulated.

For example, let's consider the decimal number 1/3, which is a repeating fraction in decimal form: 0.333333...

Single-Precision (float):

If we store 1/3 as a float, the limited precision of 32 bits means that only a certain number of decimal places can be accurately represented. The result might be something like.

float oneThird = 0.33333334f;

Here, the precision is limited due to the available bits, and the value is rounded after a certain number of decimal places.

Double-Precision (double):

Using a double, with its 64-bit precision, allows for more accurate representation:

double oneThird = 0.3333333333333333;

In this case, the greater precision of the double allows more decimal places to be stored accurately.

boolean data type:

Size: Not applicable

Range: Not applicable but allowed values are true or false.

Which of the following boolean declarations are valid?

Example 1:

- boolean b=true;
- boolean b=True;
- boolean b="True";
- boolean b=0;

Char data type:

Size: 2 bytes

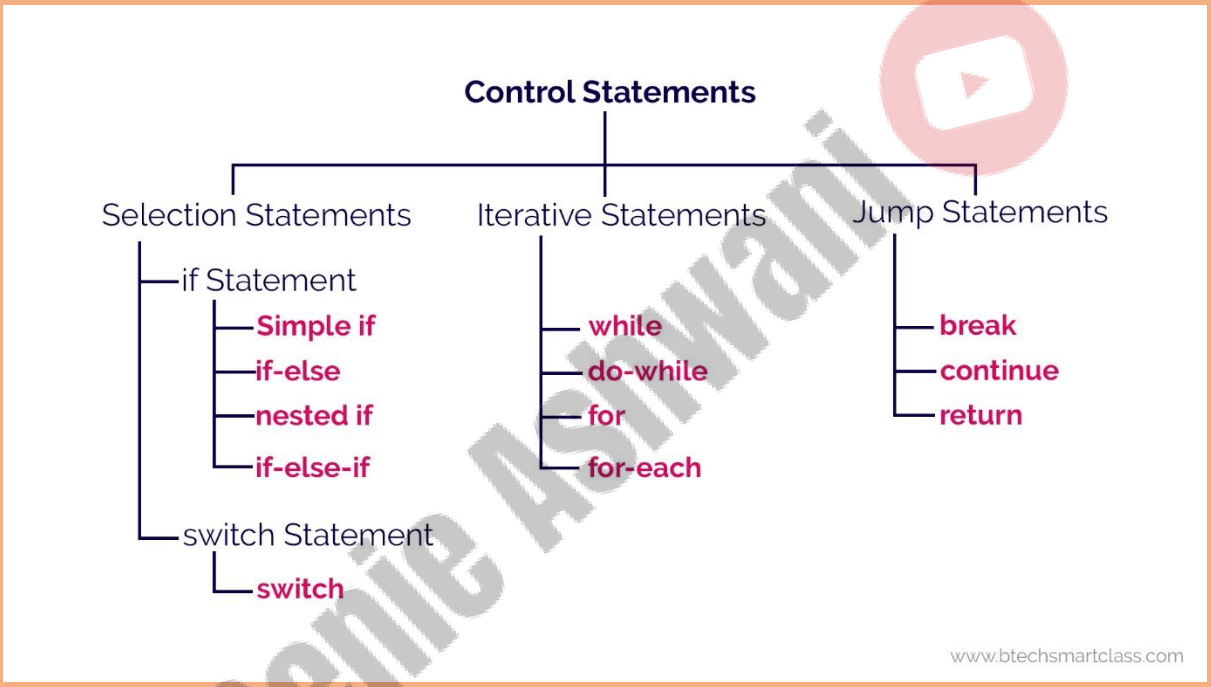
Range: 0 to 65535

Note:

In old languages like C & C++ are ASCII code based the no.Of ASCII code characters

are < 256 to represent these 256 characters 8 - bits enough hence char size in old languages 1 byte. In java we are allowed to use any worldwide alphabets character and java is Unicode based and no.Of unicode characters are > 256 and <= 65536 to represent all these characters one byte is not enough compulsory we should go for 2 bytes.

Control Flow



if-else Statements: Making Choices

Syntax:

```
if (condition) {
   // Code to execute if condition is true
} else {
   // Code to execute if condition is false
}
```

Note1: The argument to the if statement should be Boolean by mistake if we are providing any other type we will get "compile time error".

Decision-Making Statements (if-else):

```
int sal = 20000;

if (sal > 10000) {
    System.out.println("Sal is greater than 10000.");
}
```

```
int sal = 30000;
if (sal > 30000) {
    System.out.println("Sal is greater than 30000.");
} else {
    System.out.println("Sal is not greater than 30000.");
}
```

Note1: Both else part and curly braces are optional. Without curly braces we can take only one statement under if, but it should not be declarative statement.

```
public class IfElseExample {
    public static void main(String[] args) {
        int number = 7;

        if (number % 2 == 0) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
        }
    }
}
```

---

else-if Ladder:

```
int num = 75;
if (num >= 90) {
    System.out.println("Excellent!");
} else if (num >= 70) {
    System.out.println("Good job!");
} else if (num >= 50) {
    System.out.println("Keep it up!");
} else {
    System.out.println("Needs improvement.");
}
```

Switch

If several options are available then it is not recommended to use if-else we should go for switch statement. Because it improves readability of the code.

Syntax:

```
switch(x)
{
case 1:
action1
case 2:
action2
.
.
.
default:
default action
}
```

Note: Until 1.4 version the allow types for the switch argument are byte, short, char, int but from 1.5 version on wards the corresponding wrapper classes (Byte, Short, Character, Integer) and "enum" types also allowed.

Examples

```
public class SwitchExmp{
public static void main(String args[]){
int x=10;
switch(x)
{
System.out.println("hello");
}}}
```

Compile time error

Note: Every case label should be "compile time constant" otherwise we will get compile time error and Duplicate case labels are not allowed

---

```
public class ExampleSwitch{

public static void main(String args[]){

int x=10;

int y=20;

switch(x)

{

case 10:

System.out.println("10");

case y:

System.out.println("20");

}

}

}
```