Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java : Data Types/5a9ff6e53dde0810c26d5314

TUTORIAL

Java : Data Types

Topics

1.13 Using Tabs in printing

Data Types

Java is a strongly typed language. This means that every variable must have a declared type. There are eight primitive types in Java. Four of them are integer types; two are floating-point number types; one is the character type char, used for code units in the Unicode encoding scheme and one is a boolean type for truth values.

Integers: Java defines four integer types: **byte, short, int,** and **long**. All of these are signed, positive and negative values. Java does not support unsigned, positive-only integers. Many other computer languages, including C/C++, support both signed and unsigned integers. The width and ranges of these integer types vary widely.

```
Name              Width(bits)               Range

long                 64                    -
9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int                  32                    -2,147,483,648 to
2,147,483,647
short                16                    -32,768 to 32,767
byte                 8                     -128 to 127
```

byte: It is the smallest integer type. These variables are defined as below; -

```
byte b,c;
```

This is a signed 8-bit type that has a range from –128 to 127. Variables of type byte are especially useful when you're working with a stream of data from a network or file.

short: short is a signed 16-bit type. It has a range from –32,768 to 32,767. It is probably the least-used Java type, since it is defined as having its high byte first (called big-endian format). This type is mostly applicable to 16-bit computers. To define a short variable type: -

```
short s;            short, p,q,r
```

int: It is a signed 32-bit type that has a range from –2,147,483,648 to 2,147,483,647. In addition to other uses, variables of type int are commonly employed to control loops and to index arrays. Any time you have an integer expression involving bytes, shorts, ints, and literal numbers, the entire expression is automatically promoted to int before the calculation is done in Java.

long: long is a signed 64-bit type and is useful for those occasions where an int type is not large enough to hold the desired value. The range of a long is quite large. This makes it useful when big, whole numbers are needed. For example, computing the miles traveled by light in a specific range of days.

Floating: These are used to deal with fractional numbers. In Java two types of floating numbers float and double. There attributes are as below: -

```
Name              Width(bits)
                                    Range

float                  32               approximately
±3.40282347E+38F (6–7 significant decimal digits)
double                 64               approximately
±1.79769313486231570E+308   (15 significant decimal digits)
```

float: The type float specifies a single-precision value that uses 32 bits of storage. Single precision is faster on some processors and takes half as much space as double precision, but will become imprecise when the values are either very large or very small.

Variables of type float are useful when you need a fractional component, but don't require a large degree of precision. For example, to represent Rupees with paise. To declare float variable type: -

```
float paise;        float f;
```

double: It uses 64 bits to store a value. Double precision is actually faster than single precision on some modern processors that have been optimized for high-speed mathematical calculations. All built-in mathematical functions return a value of type double. To declare double type variables type: -

```
double pi;        double d;
```

All floating-point computations follow the IEEE 754 specification. In particular, there are three special floating-point values to denote overflows and errors:

• Positive infinity

• Negative infinity

• NaN (not a number)

For example, the result of dividing a positive number by 0 is positive infinity. Computing 0/0 or the square root of a negative number yields NaN. The constants Double.POSITIVE_INFINITY, Double.NEGATIVE_INFINITY, and Double.NaN (as well as corresponding Float constants) represent these special values, but they are rarely used in practice. In particular, you cannot test

```
if (x == Double.NaN) // is never true
```

to check whether a particular result equals Double.NaN. All "not a number" values are considered distinct. However, you can use the Double.isNaN method:

```
if (Double.isNaN(x)) // check whether x is "not a number"
```

Floating-point numbers are not suitable for financial calculation in which roundoff errors cannot be tolerated. For example, the command System.out.println(2.0 - 1.1) prints 0.8999999999999999, not 0.9 as you would expect. Such roundoff errors are caused by the fact that floating-point numbers are represented in the binary number system. There is no precise binary representation of the fraction 1/10, just as there is no accurate representation of the fraction 1/3 in the decimal system.

Characters: **char** is used to represent character variables. Char in Java are different than char in C/C++. In C/C++ char is datatype to store 8-bits and represent ASCII values of characters. Since Java is designed to allow applets to be written for worldwide use, it makes sense that it would use Unicode to represent characters. Of course, the use of Unicode is somewhat inefficient for languages such as English, German, Spanish, or French, whose characters can easily be contained within 8 bits. But such is the price that must be paid for global portability. So Java uses Unicode to represent characters. Unicode defines a fully international character set that can represent all of the characters found in all human languages. Thus, in Java char is a 16-bit type. The range of a char is 0 to 65,536. There are no negative chars. To declare character variables type: -

```
char ch1;          char ch2='G';
```

```
Escape Sequences for Special Characters
Escape Sequence       Name               Unicode value
\b                    Backspace          0x0008
\t                    Tab                0x0009
\n                    Linefeed           0x000A
\r                    Carriage Return    0x000D
\"                    Double Quote       0x0022
\'                    Single Quote       0x0027
\\                    Backslash          0x005C
```

Booleans: For logical variables Java has data type called boolean. It can have only one of two possible values, true or false. This is the type returned by all relational operators, such as a < b. boolean is also the type required by the conditional expressions that govern the control statements such as if and for. To declare boolean variables type: -

```
boolean decision;        boolean var1;
```

In C++, numbers and even pointers can be used in place of boolean values. The value 0 is equivalent to the bool value false, and a non-zero value is equivalent to true. This is not the case in Java. For example,

Following C++ program will run smoothely,

```
#include<iostream>
using namespace std;
int main()
{
  int i=5;
  if( i )
    cout<<"i is non-zero";
  else
    cout<<"i is zero";
}
```

But the following JAVA program will not run: -

```
class Main
{
  public static void main(String ab[])
  {
    int i=5;
    if( i )
      System.out.println("i is non-zero");
```

Java

```
    else
      System.out.println("i is zero");
  }
}

```

Thus, Java programmers are shielded from accidents such as

```
if (x = 0) // oops...meant x == 0
```

In C/C++, this test compiles and runs, always evaluating to false. In Java, the test does not compile because the integer expression x = 0 cannot be converted to a boolean value.

Using Tabs in printing

While using \t in printing strings, generally \t will align the text for multiple of eight characters e.g. if we execute print("aman\t"), then it will print aman followed by 4 spaces to make a total of 8 characters.

Similarly, if we execute print("Ram\tShyam\t"), then it will print Ram followed by 5 spaces and then Shyam followed by 3 spaces.

For example,

Java

```
public static void main(String[] args)
{
  System.out.println("Ram \t");
  System.out.println("Ram \tShyam \t");
  System.out.println("Ram \tShyam");
}

```