Data Types

In java every variable and expression have some type, and every type is strictly defined.

Every assignment should be checked by the compiler for compatibility.

Because of the above reason we can conclude java language is strongly typed programming language.

Java is not considered as pure Object-Oriented Programming Language.

Because of primitive datatypes which are not objects.

Because of few OOPs features (operator overloading, multiple inheritance, etc.) are not satisfied by java.

Data Types

Non-numeric (Unsigned)

boolean

char

Numeric (Signed)

Integral

Floating

Except boolean and char data types, remaining data types are signed data types, because we can save both the positive and negative numbers.

|  |
| --- |
| **public** **class** Type {  **int** i = 10;  **int** j = -20;  // char c1 = -'1'; //CE: incompatible types: possible lossy conversion from int to char  **char** c2 = '2';  **boolean** b1 = **true**;  //boolean b2 = -false; // CE: bad operand type boolean for unary operator '---'  **public** **static** **void** main(String args[]) {  }  } |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| MSB – X  Sign Bit | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

The Most Significant Bit act as Signed Bit

0 means positive number, Positive numbers will be represented directly in the memory.

1 means negative number, Negative numbers will be represented in 2’s complement factor.

**byte**

Rarely used data type in java, suitable for handling data in terms of streams to and from the file or network.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | size | Max value | Min value | range |
| byte | 1 byte | 127 | -128 | -128 to 127 |

|  |
| --- |
| **public** **class** Byte {  **byte** b1 = 127;  **byte** b2 = 128; // CE: incompatible types: possible lossy conversion from int to byte  **byte** b3 = 1.1; // CE: incompatible types: possible lossy conversion from double to byte  **byte** b4 = **true**; // CE: incompatible types: boolean cannot be converted to byte  **byte** b5 = "Surya"; //CE: incompatible types: String cannot be converted to byte  **public** **static** **void** main(String args[]) {  }  } |

**short**

Rarely used datatype in java, suitable for 16 bit processors like 8080 but these processors are outdated and hence short datatype is outdated datatype

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | size | Max value | Min value | range |
| short | 2 bytes | 32767 | -32768 | -32768to 32767 |

|  |
| --- |
| **public** **class** Short {  **short** s1 = -32768;  **short** s2 = 32767;  **short** s3 = 32768; // CE: incompatible types: possible lossy conversion from int to short  **short** s4 = 1.1; // CE: incompatible types: possible lossy conversion from bouble to short  **short** s5 = **true**; // CE: incompatible types: boolean cannot be converted to short  **short** s6 = "Surya"; // CE: incompatible types: String cannot be converted to short  **public** **static** **void** main(String args[]) {  }  } |

**int**

Mostly used data type in java

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | size | Max value | Min value | range |
| Int | 4 bytes | 2147483647 | -2147483648 | -2147483648 to 2147483647 |

|  |
| --- |
| **public** **class** Int {  **int** i1 = -2147483647;  **int** i2 = 2147483647;  **int** i3 = 2147483648; // CE: integer number too large:2147483648  **int** i4 = 1.1; // CE: incompatible types: possible lossy conversion from bouble to int  **int** i5 = **true**; // CE: incompatible types: boolean cannot be converted to int  **int** i6 = "Surya"; // CE: incompatible types: String cannot be converted to int  **public** **static** **void** main(String args[]) {  }  } |

**long**

Sometimes int may not suitable to hold big values, then we should go for long datatype.

Example

1. To save phone number
2. Distance travelled by light in 100 days
3. Number of characters presented in a big file

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | size | Max value | Min value | range |
| Int | 8 bytes | 2 power 63 -1 | - (2 power 63) | -2 power 63 to (2 power 63)-1 |

**float**

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

Float follows single precision – less accuracy.

**double**

If you want 14 to 15 places of accuracy, then we should go for double.

Double follows double precision – more accuracy.

**boolean**

For performing logical and decision-making operations

**char**

In old languages (like c and c++) are ASCII based and number of ASCII characters are less than or equal to 256

To represent these 256 characters 8 bits are enough, hence size of char in old languages is 1 byte.

But java is UNICODE based and the number of Unicode characters are >256 and <= 65536, hence to represent these many characters 8 bits are not enough, compulsory we should go for 16 bits, due to this the size of char in java is 2 bytes.

**null**

null is the default value for Object reference, and it can’t apply for primitive data types, if we are trying to assign null value for primitive data types, then we will get compile time error.