1.

Statically Typed Language - Type checking of variables done at compile time. Can be seen in Compiled, Intermediate languages. Suitable for enterprise applications.

Dynamically Typed Language - Type checking of variables done at runtime. Can be seen in Interpreted languages.

Strongly Typed Language - High considerations on the type of variables. Operations between variables of different types may not be allowed, and you need to perform explicit typecasting when converting between types.

Loosely Typed Language - Low considerations on the type of variables. The data types of variables are more flexible, and type conversions are often implicit.

Java is both Statically typed and Dynamically typed. Checks the variable type in both compile time and run time. Java is also a Strongly typed language.

2.

Case Sensitive - Identifiers treated distincticaly are as based on their case. For example, "variableName" and "variablename" would be considered as two different identifiers in a case-sensitive language. Eg- java

Case Insensitive - Identifiers are considered the same regardless of whether they are in uppercase or lowercase. As an example, "variableName" and "variablename" will be considered as the same identifier in a case-insensitive language. Eg- SQL

Case Sensitive-Insensitive - Certain parts of the language (Eg-variable names) are case sensitive, while other parts (Eg – keywords) such as keywords, are case insensitive.

Java is a case-sensitive language.

3.

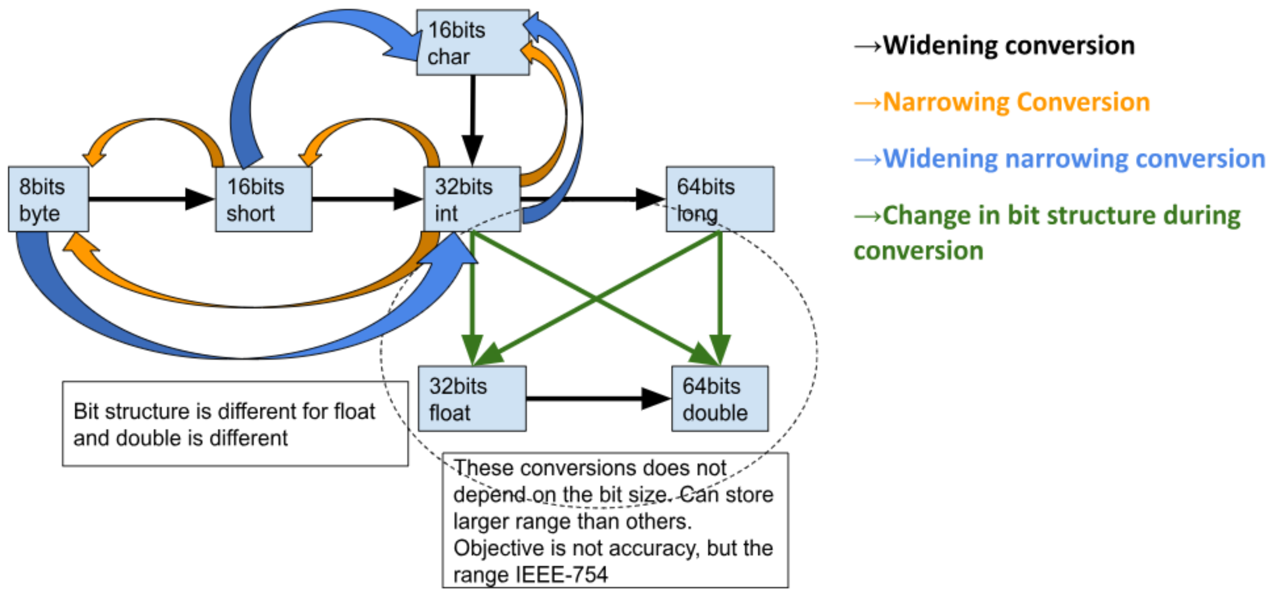
A conversion from a type to that same type is permitted for any type. It has two practical consequences. First, it is always permitted for an expression to have the desired type to begin with, thus allowing the simply stated rule that every expression is subject to conversion, if only a trivial identity conversion. Second, it implies that it is permitted for a program to include redundant cast operators for the sake of clarity.

Simply, Identity Conversion is a type of type conversion that doesn't involve any actual conversion or modification of the data. It occurs when a value is assigned to a variable of the same type or when a method is invoked with arguments of the same type as the parameter.

Eg -
```
int x = 10;
int y = x;

int num1 = 5;
int num2 = 3;
int sum = num1 +  num2;
```

4.


16bits char
8bits byte
16bits short
32bits int
64bits long
32bits float
64bits double
→Widening conversion
→Narrowing Conversion
→Widening narrowing conversion
→Change in bit structure during conversion
Bit structure is different for float and double is different
These conversions does not depend on the bit size. Can store larger range than others. Objective is not accuracy, but the range IEEE-754

Eg – byte myByte = 100;
short myShort = 50,000;

5.

Compile-time constants - Compiler knows the value of the constant
Eg – int MAX_LENGTH = 100;

Run-time constants - Compiler doesn't know the value of the constant. JVM decides the value during run time.

Eg – double BMI = x;

6.

<u>Manually / Explicit narrowing conversions</u> - When this happens manually/ explicitly it is called casting. It occurs when a value of a larger data type is assigned to a variable of a smaller data type without requiring any explicit type casting or conversion operator. The value is automatically truncated or reduced to fit within the target variable's range.

<u>Automatically / Implicit narrowing conversions</u> - Happens automatically only during an assignment context.

Conditions must be met for an implicit narrowing primitive conversion -

- During an assignment context If compiler looks for two conditions.
- The right hand side element is a compile-time constant (Eg - 10, 20, 111) not a variable (Eg - x,y, z).

- And that constant’s value matches with left hand side’s byte range.
- Narrowing conversion is only applicable for byte, short, char, int. Not applicable for long, float, double. That is due to computer architecture.

7.

Conversions to float and double from integrals does not depend on the byte size. A byte structure change occurs during this conversion. float and double can store larger range than others. Objective of float and double is not accuracy, but the range according to IEEE-754 standards.

8.

It strikes a balance between memory usage, processing speed, range, precision, cross-platform compatibility of the language, for a wide range of programming tasks.

<u>Memory usage, processing speed :</u>
Java was designed and developed in the mid-1990s. At that time, 32-bit processors were common, and the int data type was the most efficient choice for representing integers in terms of both memory usage and processing speed.

double data type, which is a 64-bit floating-point type, was chosen as the default for floating-point literals because it offered a good trade-off between precision and performance.

<u>Range, precision :</u>
int and double ensures that developers are less likely to encounter unexpected behavior due to implicit type conversions. Using narrower data types such as byte, short, float as defaults might lead to unintended data loss or precision issues if the literals are too large or precise for those narrow types.

<u>Cross-platform compatibility :</u>
Choosing int and double as the defaults ensured that existing code written for 32-bit systems would continue to work correctly when running on 64-bit systems. Applications targeted for 32bit architecture can run in 64bit architecture. But not vice versa. However, 32bit architecture can process 64bits but not in a multiple runs.

9.

- To prevent unexpected data loss and maintain type safety.
- By allowing only narrowing conversions among these specific types, the Java language ensures that the developer is aware of the potential data loss, as it explicitly signals that the value might not fit perfectly.
- This strikes a balance between flexibility and type safety. This restriction helps developers write more reliable and predictable code by explicitly handling potential data loss when converting

10.

<u>Widening and Narrowing Primitive Conversion</u>
This conversion combines both widening and narrowing primitive conversions.
Eg -
byte to char
short to char

First, the byte is converted to an int via widening primitive conversion, and then the resulting int is converted to a char by narrowing primitive conversion.