1.

Statically typed	A language that consider the type (variables, expressions, etc) while in compile time.
Dynamically typed	A language that consider type while in run time.
Strongly typed	A language strictly consider about the type.
Loosely typed	A language that doesn't strictly consider the type.

Java is strongly typed and is both statically and dynamically typed.

2. i. Case sensitive

When a programming language is case-sensitive, it means that it distinguishes between uppercase and lowercase letters in identifiers. This means that "file" and "File" would be considered as two different names.

Ex; in java

```
int x = 10;
int X = 10;
```

ii. Case insensitive

When a programming language is case-insensitive, it means that it does not differentiate between uppercase and lowercase letters in identifiers. So, "file" and "File" would be considered the same name.

Ex; in SQL

```
SELECT * FROM customer
select * from customer
```

In SQL, the keywords like "SELECT" and "FROM" are case-insensitive. Both uppercase and lowercase versions of these keywords will work identically.

iii. Case sensitive insensitive

This term generally refers to a situation where a programming language is partially case-sensitive and partially case-insensitive. Some parts of the language may be case-sensitive, while others are case-insensitive.

Ex; in JS

```
let myVariable = 10;
 let myvariable = 20;
```

In JavaScript, variable names are case-sensitive (as shown by "myVariable" and "myvariable"). However, function names are case-insensitive,

Ex;

```
function sayHello() {
  console.log("Hello!");
  }

   function sayhello() {
```

```
 console.log("Hello!");
}
```

3. Identity conversion

In Java, a type conversion is referred to be an identity conversion when a value is cast to the same type without affecting or losing any data. Because it just validates that the value is consistent with its own type without changing the original value, this method of conversion is considered the safest.

Certain identity conversions are permitted by Java because they are inherently secure and don't call for explicit casting. The guidelines for identity conversion are laid down in the Java Language Specification, which also specifies which conversions are permitted without any additional syntax.

Ex;
int num = 10;
int num2 = num;
System.out.print(num) → prints 10

In this example, the variable “num” is of type int, and we perform an identity conversion by assigning its value to another variable “num2”, which is also of type int. The value num is simply copied to result without any modification, as both variables are of the same type (int).

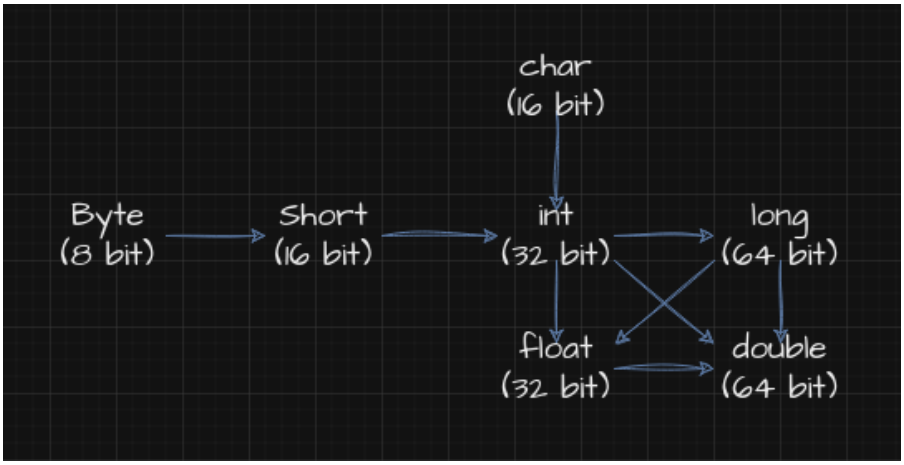
Ex;
char a = 'J';
char b = a;

System.out.print(b); → prints J

Here, we have a variable “a” of type char, which stores the character 'J'. We then perform an identity conversion by assigning the value of a to another variable “b”, which is also of type char. As a result, the character 'J' is copied to b.

4. Primitive Widening Conversion

Primitive widening conversion in Java is an automatic type conversion that occurs when a value of a smaller data type is assigned to a variable of a larger data type. It is also known as implicit type conversion. The conversion is safe and does not result in any loss of data because the target data type can hold all possible values of the source data type. In Java, the following primitive data types are ordered by size, from smallest to largest:



Ex;

int numInt = 100;

long numLong = numInt;

System.out.println(numLong);

In this example, the value stored in the numInt variable, which is of type int, is implicitly converted to a long type and assigned to the numLong variable. Since a long can accommodate all possible values of an int, no data loss occurs.

A widening primitive conversion from int to float, or from long to float, or from long to double, may result in *loss of precision* - that is, the result may lose some of the least significant bits of the value. In this case, the resulting floating-point value will be a correctly rounded version of the integer value, using IEEE 754 round-to-nearest mode

5. Java constants

i. Compile-time constants

Java compiler identify the value of the constant in compile time.

Ex;

byte myByte = 10;

final short MY_SHORT = 10;

char myChar = MY_SHORT;

ii. Run-time constants

Java compiler identify the value of the constant in run time through JVM.

Ex;

```
short MY_SHORT = 10;

char myChar = MY_SHORT;
```

6. When a user performs operations or assignments involving several numerical data types in Java, data type conversions take place. Narrowing conversions take place when the user changes a data type with a wide range to one with a narrow range, perhaps losing data in the process.

i. Implicit (Automatic) Narrowing Primitive Conversions
- These conversions are performed automatically by the Java compiler.
- No data losses during the conversion.
- Only occur within byte,short,int,char data types only.

*To perform this conversion there should be some facts fulfilled,
1. It should be a assignment context
2. Assigning value must be within a range of data type
3. Assigning value must be compile-time constant

ii. Explicit Narrowing Conversions (Casting)
- These conversions require explicit instructions from the programmer to inform the compiler that the conversion is intentional.
- Data losses may occur during the conversion.
- Casting is done by specifying the target data type in parentheses before the value or expression to be converted.

7. long data type assign to float data type.

The IEEE 754 standard specifies that the float (and double) type holds integers. After Integer has been normalized, it stores it. It can therefore store a large variety of data outside the allowable data range. However, the data's accuracy will decline. Because when data is rounded to the nearest value, some of the least significant bits of the value may be lost.

8. Why are int and double set as the default data types.

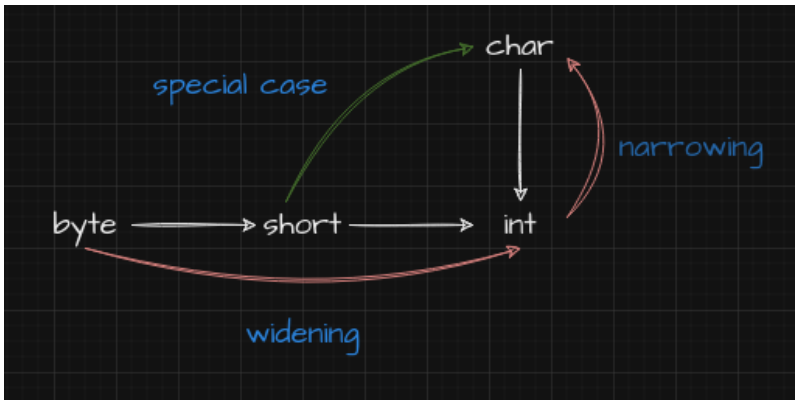
Java was initially designed in the mid-1990s, and its creators aimed to make it a simple, easy-to-learn language with similarities to C and C++. In C and C++, int is the default data type for integer literals, and double is the default data type for floating-point literals. Java borrowed this convention to maintain familiarity with programmers coming from C/C++ backgrounds. Using int as the default for integer literals makes sense because, on most platforms, int operations are typically faster and more efficient than those involving larger integer types like long. The decision to default to double for floating-point literals aligns with the IEEE 754 standard, which recommends using the double-precision format (64-bit) as the default for floating-point calculations due to its wider range and better precision compared to single-precision (32-bit) floats. Using int for integer literals and double for

floating-point literals strikes a balance between practicality and usability for common programming scenarios.

9. Why does implicit narrowing primitive conversion only take place among byte, char, int, and short?

   Implicit narrowing primitive conversion is limited to byte, char, int, and short in Java to prevent accidental data loss and maintain consistency and safety in the language. When converting from larger data types like long, float, or double to smaller data types, explicit type casting should be used to indicate that the programmer is aware of the potential data loss and has intentionally made the conversion. This approach promotes code clarity, reduces potential errors, and aligns with Java's design principles.

10.



In java, implicitly widening conversions occur through the white arrow direction. So that the byte and short can easily conversion to type int. But byte, short → char conversion is somewhat different. First byte convert to int by widening conversion and int convert to char by narrow conversion. This scenario called as “widening and narrowing conversion”. And there’s a special case short doesn’t contribute to this process. It will directly convert to char.