Assignment- 02
Answer Sheet

01.
Java is a statically typed and strongly typed language.

- Strongly typed languages strongly consider the data types . Like, we cannot put an integer in a String variable. We have to strictly follow the language structure and declare data type accurately.
- Loosely typed languages are the opposite of strongly typed languages these languages don’t use a strong and complex data type system for execution.
- Statically typed languages check the data type during compile time and programs are checked before being executed, and a program might be rejected before it starts. It is required to have a compiler for a language to be statically typed(compiled languages or intermediate languages.)
- Dynamically typed means that they checks the type during run time. Error may occur in the run time rather in the development phase.

Therefore in most cases, enterprise applications prefer strongly typed and statically typed languages. Because it shows type mistakes before runtime.

02.

Case Sensitive

Case Sensitive means that it distinguishes between uppercase and lowercase letters in identifiers (variable names, function names, etc.) and when comparing strings. This means that "A" and "a" are treated as distinct characters

Ex: **C#, Java, Ruby, and XML**.

In java.

String AB =”Hello”;

String Ab =”Hello”;

AB and Ab are two different identifiers that's why we can declare the variable again and for key word “String ” we have type it’s is if we type as “string” it’s wrong.

Case insensitive

A language is considered case insensitive if it does not distinguish between uppercase and lowercase letters in identifiers and when comparing strings. In this case, "A" and "a" are considered equivalent, and "Hello" and "hello" are treated as the same string.

Case Sensitive-insensitive Language

A language with a mixed approach where some parts are case sensitive while others are case insensitive. For example, variable names might be case sensitive, but certain keywords or function names might be case insensitive.

Ex SQL

In SQL not all case insensitive but some are insensitive,

SELECT*FROM items:

select*from items;

both statements give the same results but

SELECT*FROM ITEMS;

SELECT*FROM items;

not the same.

Java is Case Sensitive Language.

03.

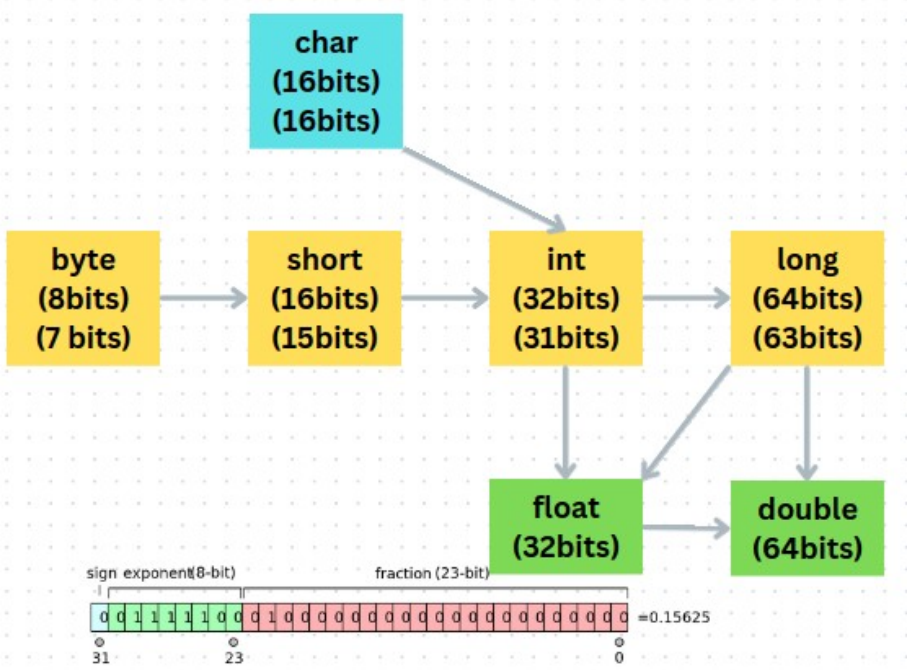
Conversion from one data type to itself, where the source and target types are identical and doesn't require any explicit casting because it’s built in implicit conversion.(exchanging value between the same data types)

Ex;

```
int x = 42;
int y = x; // Identity conversion (int to int)
```

04.

Following shows the hierarchy of primitive data types in Java,



Type conversion in Java that automatically promotes a data type to a larger data type to accommodate larger values without any loss of information. This is a implicit conversion occurs as show in above diagram.
Ex:
Converting a byte to a int is a widening conversion is implicit process since byte have 7 bits for value and int has has 31 bit for value and it can be easily accommodate to a int from byte.

Ex:

```
        float floatValue = 3.14f;

        double doubleValue = floatValue; // Widening conversion (float to
double)

        System.out.println("floatValue: " + floatValue);
        System.out.println("doubleValue: " + doubleValue);
```

output

floatValue: 3.14

doubleValue: 3.14

05.

Compile-time constants, also known as "compile-time literals," are constants whose values are known and fixed during the compile-time of the program. These values are directly embedded into the bytecode generated by the Java compiler, and their evaluation happens during the compilation phase, not at runtime.

Ex:

```
final int COMPLITIME_CONST=10;
```

value 10 is is fixed and identified at compile time.

Run-time constants are constants whose values are not known during the compile-time but are determined and assigned during the execution of the program at runtime.

Ex:

```
Scanner scanner = new Scanner(System.in);

final int RUNTIME_CONST=scanner.nextInt();
```

Runtime const is identified at runtime and input give at runtime by the user.

06.

Implicit Narrowing Primitive Conversions

Implicit narrowing primitive conversions, also known as automatic type conversions, occur when you assign a value of a larger data type to a variable of a smaller data type without explicitly indicating the conversion. Java automatically converts the value to fit into the smaller data type.

Implicit narrowing conversion is allowed in assignment context and following requirement are met,
01.The range of the source value falls within the range of the target data type.
02.Data type between conversions are byte,short,int,char.
03. source value should be compile time constant.

Explicit Narrowing Conversions (Casting)

Explicit narrowing conversions, also known as casting, occur when you explicitly tell the compiler to convert a value of a larger data type to a smaller data type. This is done by placing the target data type in parentheses before the value to be converted. Casting allows us to indicate that we are aware of the potential data loss, and the compiler will perform the conversion based on our instruction

```
double doubleValue = 3.14159;
int intValue = (int) doubleValue;
```

07.
In the case of assigning a long to a float, Java performs a narrowing conversion, as long is 64 bits and float is 32 bits. This conversion happens implicitly due to difference in the representations of these data types rather than the number of bits they occupy.

long Data Type (64 bits):
The long data type in Java represents 64-bit signed two's complement integers. It has a range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 with a precision of 1.

float Data Type (32 bits):
The float data type in Java represents 32-bit IEEE 754 floating-point numbers. It can represent a wider range of values but with lower precision compared to long. It has a range of approximately ±3.40282347 x 10^38 with a precision of approximately 7 significant digits.

When a long value is assigned to a float variable, the value undergoes a narrowing conversion. During this conversion, the long value is converted to the closest representable float value. However, since float has a lower precision than long, there may be some loss of precision in the conversion.

08.
There could be three main reasons for that selection,
- Performance
- Maintaining balance between precision and memory usage.
- Backward compatibility.

Performance

JVM(Java virtual machine) architecture is optimized to work efficiently with ‘int’ and ‘double’ data types. Since int and floating point numbers are among the most commonly used data types in programming.

Precision and Memory usage

‘int’ is a 32bit data type and can represent a wide range of whole numbers while ‘double’ is a 64 bits data type and provides high level of precision most of practical purposes than floats. As a example using ‘long’ as the default for integers would consume more memory and using ‘float’ as the default for floating point numbers lead to less precision.

Backward compatibility

Before java there was C and C++ and when java initially designed one the key goal was to be compatible with existing languages. In C and C++ integer literals without suffix are treated as ‘int’ values by default and similler for float and double. Adapting to the same would easier for developers.

09.
The decision to allow implicit narrowing only in byte, char, int, and short in Java has historical roots, practical use cases related to array indexing and looping constructs, and a perceived lesser impact on data loss compared to long to int conversion. While it may seem inconsistent at first, this behavior aligns with Java's goal of maintaining compatibility with C/C++, practicality in array usage, and a balance between safety and convenience
As a Example,
While implicit narrowing from int to byte may lead to data loss due to the reduced range of byte, the impact of such data loss is generally considered less severe compared to narrowing from long to int. byte can represent values in the range of -128 to 127, which is still reasonably useful in many applications.

10.
Under Widening and Narrowing primitive conversion only byte to char is classified and 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.
‘short’ to ‘char ‘ isn’t classified as widening and narrowing conversion since it’s directly converted to char from short.
short is a signed data type with a range of -32,768 to 32,767. ‘char’ is an unsigned data type representing a 16-bit Unicode character with a range of 0 to 65,535.

Even though both types have the same size in memory, the conversion from short to char is a special conversion because it extends the positive range of short (0 to 32,767) to the full range of char (0 to 65,535). No information is lost during this conversion; it only changes the interpretation of the value.