Programming Fundamentals
Assignment 2

1. Elucidate the following concepts: 'Statically Typed Language', 'Dynamically Typed Language', 'Strongly Typed Language', and 'Loosely Typed Language'? Also, into which of these categories would Java fall?"

Statically Typed languages – Statically Typed Languages means that kind of languages declares the variables types in the source code at compile time.

Dynamically typed languages -Dynamically typed languages means these kinds of languages check the types of variables in the source code during the runtime procedure,and if there is a type mismatch, errors may occur during program execution.

Strongly Typed language – Strongly Typed languages means these type of languages stricly concern about the variable types in the source code and it doesn't allow implicit type conversions between different data types.

Loosely Typed Language- Loosely typed language means this type of languages do not care more about the variable types in the source code and that allows implicit type conversions between different data types.

- Java is a statically typed language as well as the dynamically typed language. In the other hand java is a strongly typed language.

2. "Could you clarify the meanings of 'Case Sensitive', 'Case Insensitive', and 'Case Sensitive-Insensitive' as they relate to programming languages with some examples? Furthermore, how would you classify Java in relation to these terms?"

Case Sensitive - Case sensitivity in a programming language refers to whether the language identify the difference between uppercase and lowercase letters in identifiers, such as variable names, function names, and keywords. In a case-sensitive language, "Identifier" and "identifier" are considered two different identifiers and can be used to represent different entities.
Ex -Python language,Java language

Case Insensitive - case-insensitive language treats uppercase and lowercase letters as equivalent when parsing identifiers. That means "Identifier" and "identifier" would be considered the same identifier, and they cannot be used to represent different entities. The language ignores the case when performing identifier lookup or comparisons.
Ex-Visual basic

Case Sensitive-Insensitive - In some programming languages, the case sensitivity can be a select. Developers can choose whether identifiers are treated as case-sensitive or case-insensitive based on their needs. In such languages, some parts of the code may be case-sensitive while others are not.
Ex – Java Script

- Java is a case-sensitive language.

3. Explain the concept of Identity Conversion in Java? Please provide two examples to substantiate your explanation.

Type/identity conversion, also known as typecasting or type coercion, is the process of converting data from one data type to another in a programming language. Java supports type conversion for both primitive and reference data types. The goal of type conversion is to change the data type of a value while retaining its original value. In Java, programmers can perform type conversion to convert a variable from one primitive data type to another or to convert objects between compatible reference data types. However, it's important to note that type conversion is not applicable to Boolean data type variables. Overall, type conversion in Java enables developers to manipulate and use data in different ways by changing its data type, allowing for flexibility and adaptability in programming solutions.

Ex - `byte myByte = 10;` - here integer type 10, convert into type byte
`short myShort2 = 355;` - here integer type 355, covert into type short

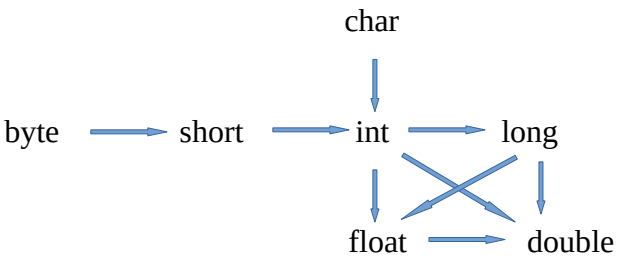
4. Explain the concept of Primitive Widening Conversion in Java with examples and diagrams.

In Java, primitive widening conversion is a type conversion that occurs automatically when a value of a smaller data type is assigned to a variable of a larger data type. In other words, it involves converting a value from one primitive data type to another larger data type without any loss of data.

Java supports the following primitive widening conversions, listed in increasing order of size:

- byte to short, int, long, float, or double
- short to int, long, float, or double
- char to int, long, float, or double
- int to long, float, or double
- long to float or double
- float to double

The order of primitive widening conversions in Java can be simply represented as follows:



According to the above graph the type conversions occurs automatically.
Ex –
```
float myFloat = 3.14f;
double myDouble = myFloat; // Widening conversion from float to double

byte myByte = 100;
```

```
int myInt = myByte; // Widening conversion from byte to int
```

5. Explain the the difference between run-time constant and Compile-time constant in java with examples.

A compile-time constant is a value that is known and fixed at compile time.
Ex- `final int MY_CONST = 10;`

A run-time constant is a value that is determined and known only during the program's execution at runtime. These constants are not resolved at compile time but are computed and assigned values during program execution.
Ex- `final int MY_CONST2 = 10 * (int) Math.random();`

6. Explain the difference between Implicit (Automatic) Narrowing Primitive Conversions and Explicit Narrowing Conversions (Casting) and what conditions must be met for an implicit narrowing primitive conversion to occur?

Implicit (Automatic) Narrowing Primitive Conversions - Implicit narrowing conversions occur automatically by the Java compiler when assigning a value of a larger data type to a variable of a smaller data type. The conversion happens without the need for explicit syntax (such as casting), but it's crucial to note that data loss may occur if the value cannot be precisely represented in the target data type. An implicit narrowing conversion can occur if and only if the following conditions are met:

- The target data type (the one being assigned to) has a smaller size (in terms of bits) than the source data type (the one being assigned from).
- The value being assigned fits within the range of the target data type. If the value is outside the valid range of the target data type, data loss will occur, and the compiler will generate a compilation error.
- The value of the source data type should be compile time constant.

Explicit Narrowing Conversions (Casting) - Explicit narrowing conversions (or casting) are performed when a developer explicitly instructs the compiler to convert a value from a larger data type to a smaller data type. It's used to indicate that the developer is aware of the potential data loss and accepts the consequences.

7. How can a long data type, which is 64 bits in Java, be assigned into a float data type that's only 32 bits? Could you explain this seeming discrepancy?”

That logic can not explain by using number of bits. Even float uses 32 bits but can represent larger values through the use of the scientific notation method. This method allows float values to cover a broader range of magnitudes, enabling conversion from any value stored in the long data type into a value that can be represented within the 32 bits of the float data type. However, because the float data type has only 23 bits available for the significant, some precision is lost in representing the original long value, which has 64 bits. Thus, the converted float value is an approximation of the original long value, with some potential loss of precision.

8. Why are int and double set as the default data types for integer literals and floating point literals respectively in Java? Could you elucidate the rationale behind this design decision?

int as the Default Data Type for Integer Literals: The int data type in Java is a 32-bit signed integer type. On most modern processors, the native word size (the size of the processor's registers) is also

32 bits or larger. As a result, operations involving 32-bit integers (such as addition, subtraction, multiplication, and division) can be performed efficiently by the processor in a single operation. Additionally, many computer architectures are optimized for handling 32-bit integer arithmetic, making int calculations very fast and efficient. Storing and processing integers in a 32-bit format is also memory-efficient and aligns well with the memory alignment constraints on most systems. Since int is the most commonly used integer type, making it the default data type for integer literals in Java results in concise code and avoids the need for explicit type declarations in many situations. This enhances code readability and reduces the risk of introducing errors related to type conversions.

Double as the Default Data Type for Floating-Point Literals: The double data type in Java is a 64-bit double-precision floating-point type. Unlike integers, floating-point operations involve more complex calculations, such as handling fractions and approximations. However, many modern processors have specialized hardware support for double-precision floating-point arithmetic, making double operations fast and efficient. Double-precision arithmetic is commonly used in scientific calculations, financial applications, and simulations where higher precision is required. The wider range and increased precision provided by double-precision floating-point numbers make them more suitable for representing a broader spectrum of real-world values. By using double as the default data type for floating-point literals, Java encourages developers to use higher precision by default, which can be beneficial in many mathematical and scientific contexts. This choice also aligns with the IEEE 754 standard for floating-point representation, ensuring consistency across platforms.

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

In Java, implicit narrowing primitive conversion is limited to the data types byte, char, int, and short because these types have sizes smaller than int, the default integral type in Java (32 bits). This kind of conversion can take place logically among these data types because their sizes are compatible for representing values within their respective ranges. For example, an int value can fit into a short or a byte because the short and byte data types have sufficient bits to represent the smaller range of values that an int can hold. This restriction ensures data type consistency and helps prevent unintentional data loss when converting between different types. By allowing implicit narrowing only among these compatible types, Java encourages developers to be more explicit about narrowing conversions and forces them to consider potential data loss. This design decision also enhances code clarity and consistency by identifying situations where narrowing conversion is taking place between data types of similar sizes. For conversions involving larger data types like int, long, or double to smaller types, explicit casting is required to acknowledge the possibility of data loss, promoting safer and more intentional handling of data type conversions in Java programs.

10. Explain “Widening and Narrowing Primitive Conversion”. Why isn't the conversion from short to char classified as Widening and Narrowing Primitive Conversion?

The conversions which combine both widening and narrowing primitive conversions can be identified as the Widening and Narrowing Primitive Conversions.

EX : byte to char conversation - 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 conversions. Here all the conditions for implicit narrowing primitive conversions are applied as usual.

The conversion from short to char in Java is classified as a special conversion rather than a widening or narrowing primitive conversion. Because both short and char being 16-bit data types,

their ranges and interpretations differ. short is a signed 16-bit integer type representing positive and negative values, whereas char is an unsigned 16-bit integer type representing Unicode characters with non-negative values. When converting from short to char, the conversion involves changing the interpretation of the underlying bits rather than simply resizing the data. As a result, the conversion is considered special, as it maps short values directly to char, preserving the range but potentially resulting in different character representations, particularly for negative values. Because of this specific mapping between short and char, the conversion is considered a special conversion rather than a typical widening or narrowing primitive conversion.