Question 01

Statically Typed Language: In a statically typed language, the data types of variables are determined and checked at compile time. it means that we must explicitly declare the data type of each variable before using it, and the compiler enforces strict type checking during the compilation process. Once a variable is assigned a particular data type, it cannot be changed to a different data type later in the program.

Dynamically Typed Language: In a dynamically typed language, the data types of variables are determined and checked at runtime. We don't need to explicitly specify the data type when declaring variables. Instead, the data type is inferred based on the value assigned to the variable at runtime. Also, the type of a variable can be changed during the execution of the program.

Strongly Typed Language: A strongly typed language enforces strict type checking and doesn't allow automatic type conversions between different data types without explicit conversion by the programmer. This means that we cannot perform operations between incompatible data types without explicitly converting them. It helps catch potential type related errors at compile time. Both statically and dynamically typed languages can be strongly typed.

Loosely Typed Language:In contrast, a loosely typed language allows automatic type conversion between different data types without requiring explicit conversion by the programmer. The language system performs implicit type coercion to make the data types compatible for specific operations. This can lead to more flexible coding but might also introduce unexpected behaviors if not used carefully.

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

Question 02

Case Sensitive: A language is considered case sensitive when it treats uppercase and lowercase letters as distinct and significant. This means that using different cases (e.g., uppercase and lowercase) for the same word will result in different meanings or interpretations. For example, "Variable" and "variable" are treated as two different identifiers in a case-sensitive language.

Case Insensitive: A language is considered case insensitive when it treats uppercase and lowercase letters as equivalent. This means that using different cases for the same word will not change the interpretation or meaning of the word. For example, "Variable" and "variable" would be considered the same identifier in a case-insensitive language.

Case Sensitive-Insensitive: Some programming languages and systems can have a mix of case sensitivity in different parts. For example, variable names might be case sensitive, while function names or keywords are case insensitive. It can lead to varying behavior depending on the context.

Question 03

In Java, an identity conversion is a type of type conversion where no actual conversion is performed. It means that the value and the type of the expression remain the same after the conversion. This conversion is called "identity" because it doesn't change anything about the data; it merely reaffirms the type of the expression.

In Java, identity conversion occurs under the following conditions:

1.When you assign a value to a variable of the same type, no conversion is needed, and it is considered an identity conversion.

2.When you pass an argument to a method parameter of the same type, no conversion is needed, and it is considered an identity conversion.

Question 04

Primitive widening conversion, also known as automatic type promotion, is a type of type conversion in Java where a value of a smaller data type is automatically converted to a larger data type without any explicit casting or loss of information. This conversion is considered safe because there is no risk of losing data when converting from a smaller data type to a larger one.

In Java, primitive widening conversions occur when you perform operations involving different data types, and the smaller data type is automatically promoted to the larger data type to perform the operation.

Question 05

Run-time constant: A run-time constant is a constant whose value is determined and evaluated at run-time, as an example while the program is executing. Unlike a compile-time constant, the value of a run-time constant can change during the program's execution.

Compile-time constant: A compile-time constant is a constant whose value is known and can be evaluated by the compiler at compile-time, as an example when the code is being converted into executable bytecode. The value of a compile-time constant cannot change during the program's execution, and the compiler replaces all occurrences of the constant with its actual value at compile-time.

Question 06

Implicit (Automatic) Narrowing Primitive Conversions: Implicit narrowing primitive conversions happen automatically and are performed by the Java compiler when a value of a larger data type is assigned to a variable of a smaller data type. The conversion is considered narrowing because it may result in the loss of information if the value being assigned cannot be accurately represented in the smaller data type. Despite the potential loss of information, Java allows these conversions when it believes that the programmer is aware of the possible loss and has explicitly decided to perform the conversion automatically.

Explicit Narrowing Conversions (Casting): Explicit narrowing conversions, also known as casting, are conversions in which the programmer explicitly tells the compiler to convert a value from a larger data type to a smaller data type. This is done by using a cast operator. The cast operator is used to inform the compiler that the programmer is aware of the potential loss of information and still wants to perform the conversion.

Question 07

The assignment of a 64-bit long data type to a 32-bit float data type in Java is an example of a narrowing primitive conversion. This conversion may result in potential loss of precision because the float data type has a smaller range and cannot accurately represent all the values that can be represented by a long data type. The reason this conversion is allowed is due to the fact that the long data type has a wider range than the float data type. Java allows such conversions because it assumes that the programmer is aware of the potential loss of precision and has explicitly decided to perform the conversion.

Here's how the conversion works:

1.When assigning a long value to a float variable, the Java compiler automatically performs an implicit narrowing primitive conversion. The compiler takes the 64-bit long value and converts it to a 32-bit float value.

2.In this conversion, the compiler discards some of the least significant bits of the long value, as they cannot be represented in the float data type. This is the step where potential loss of precision occurs.

3.The float data type is capable of representing a smaller range of values than the long data type. The float type can represent approximately 7 decimal digits accurately.

4.Any value of a long data type that falls outside the representable range of the float data type may lose precision or get rounded to the nearest representable float value.

Question 08

The choice of using `int` as the default data type for integer literals and `double` as the default data type for floating-point literals in Java is primarily driven by a balance between efficiency, precision, and backward compatibility.

1. Efficiency:
`int` is chosen as the default data type for integer literals because it is the most common and efficient data type for representing whole numbers in Java. It typically uses 32 bits of memory, which makes it more efficient in terms of memory usage and arithmetic operations on most modern processors.

2. Precision:
`double` is chosen as the default data type for floating-point literals because it offers higher precision compared to `float`. A `double` data type uses 64 bits of memory, allowing it to represent a wider range of decimal numbers and provide more significant digits compared to `float`, which uses 32 bits.

3. Backward Compatibility:
When Java was designed, the creators wanted to ensure backward compatibility with C and C++, which are influential programming languages in the history of computing. In C and C++, `int` and `double` are the default data types for integer and floating-point literals, respectively. By using the same defaults in Java, developers coming from C/C++ backgrounds would find the transition to Java more familiar and seamless.

4. Avoiding Potential Loss of Data:
By using `int` as the default for integer literals, Java avoids potential loss of data when programmers use integer literals without specifying data types explicitly. For example, if the default were `long`, code like `int x = 1000000;` would lead to a compilation error because `1000000` would be treated as a `long` value, and it wouldn't fit into an `int`. Using `int` as the default ensures that integer literals behave as expected in most scenarios.

Overall, the choice of `int` and `double` as default data types for integer and floating-point literals, respectively, strikes a balance between memory efficiency, precision, and backward compatibility. It aligns with the goals of simplicity, ease of use, and maintaining a familiar syntax for developers coming from other languages. Programmers can always explicitly specify the data types if they require different precisions or ranges, ensuring that they have control over how their data is represented and manipulated in their Java programs.

Question 09

Implicit narrowing conversion occurs only among `byte`, `char`, `int`, and `short` because they are smaller types and can safely store values from larger types without losing data. Larger types like `long`, `float`, and `double` need explicit casting to avoid potential data loss.

Question 10

"Widening primitive conversion" happens when you assign a smaller data type to a larger one without losing data. For example, assigning an int to a long is widening.
"Narrowing primitive conversion" happens when you assign a larger data type to a smaller one, which may result in data loss. For example, assigning a long to an int is narrowing.
The conversion from short to char is neither widening nor narrowing because both short and char have the same size (16 bits). It is considered an "identity conversion" as no change in size or data loss occurs.