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?”

Answer:

Statically type Language - An explicitly specified variable type at compile time and the inability for runtime type changes define a statically typed language. Variable and expression types are checked by the compiler at compilation time, and any type mismatch or error must be fixed before the program can run. Languages that are statically typed frequently perform better and catch type-related issues early in the development cycle.

Dynamically Typed Language - Programming languages that use implicit runtime variable type determination are referred to as dynamically typed languages. Variable types do not require explicit declaration, in contrast to languages with statically typed data. More flexibility and simple code are made possible by dynamic typing, but if the types are incompatible, runtime issues could result.as in Python

Strongly Typed Language - In a strongly typed language, the type of a variable is strictly enforced, and implicit type conversions between different types are limited. Ex: C++

Loosely Typed Language - In a loosely typed language, also known as weakly typed, there are fewer restrictions on mixing different types. Implicit type conversions (coercion) are more common, and variables can change types during runtime without explicit type declarations. Ex: JavaScript

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?”

Answer:

Case Sensitive: The distinction between capital and lowercase characters in a computer language is known as case sensitivity. Variables, functions, keywords, and identifiers must all be spelled with consistent capitalization in case-sensitive languages in order to be understood.

Case Insensitive: Uppercase and lowercase characters are treated equally in case-insensitive languages. In other words, "A" and "a" are regarded as the same character. Example: Visual Basic and Pascal

Case Sensitive-Insensitive: Some programming languages offer options to toggle between case-sensitive and case-insensitive behavior for specific elements (e.g., identifiers, keywords). This means that certain parts of the language might be case-sensitive while others are case-insensitive. Ex: SQL (Structured Query Language)

Java is a case-sensitive language. This means that Java distinguishes between uppercase and lowercase characters in its identifiers, keywords, and any other user-defined names. For example, "myVariable" and "myvariable" are considered two distinct variable names in Java.

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

Answer:

```
          String text = "Hello, World!";
          String message = text; // Identity conversion - 'text' and 'message' are
both      references to String objects.
```

In this example, the variable `text` holds a reference to a String object with the value "Hello, World!". When it is assigned to the variable `message`, it undergoes an identity conversion as both `text` and `message` are references to String objects.

```
Int x =10;

int y;

y = x;  // Identity conversion - y and x are both  references to Integer objects.
```

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

Answer:

Java allows for the automatic and secure conversion of smaller data type values to bigger data type values without any data loss. The Java compiler does this conversion at compile time to verify that the value can fit into the larger data type without losing precision. Java enables unidirectional widening conversions, or conversions from smaller to larger data types, between a few primitive data types.

byte → short → int → long → float → double

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

Answer:

A run-time constant is a constant whose value is not known or determined until the program is running. In following example, `MAX_VALUE` is a run-time constant because its value is determined by the result of the method calculateMaxValue(), which is called at runtime.

Ex: `final int MAX_VALUE = calculateMaxValue();`

Compile-time constants have their values known and resolved during the compile-time phase, and their values remain fixed throughout the program's execution.

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?

Answer:

Implicit narrowing conversions occur when a value of a larger data type is assigned to a variable of a smaller data type without the need for explicit casting. This type of conversion is allowed if there is no potential loss of data, meaning the value being assigned can be safely represented within the smaller data type's range.

Ex:
```
int intValue = 10;
byte byteValue = intValue;
```

Explicit narrowing conversions, also known as casting, occur when a value of a larger data type is explicitly converted to a variable of a smaller data type using a cast operator. This type of conversion is performed when there is a possibility of data loss, and the programmer explicitly indicates their intention to perform the conversion.

Ex:
```
double doubleValue = 3.14;
int intValue = (int) doubleValue;
```

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?”

Answer:

Java allows assigning a `long` value to a `float` variable without explicit casting because it is considered a widening conversion, where a value is converted to a larger data type. This is a special case of widening conversion known as a "narrowing primitive conversion" with a special rule for integral-to-float conversions.

Ex:
```
long longValue = 1234567890123456789L;
float floatValue = longValue;
System.out.println("longValue: " + longValue);
System.out.println("floatValue: " + floatValue);
```

Output:

```
longValue: 1234567890123456789
floatValue: 1.23456788E18
```

In this example, the `long` value 1234567890123456789L is assigned to the `float` variable `floatValue`. You can see that some precision is lost in the `float` representation, and the result is in scientific notation (1.23456788E18).

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?

Answer:

By default, integer literals in Java are considered to be of type int. For example, if you write int x = 42;, the value 42 is treated as an int.
When Java was originally designed, the int type was chosen as the default for integer literals to maintain compatibility with C and C++, as many C/C++ programmers were expected to transition to Java. C and C++ both treat integer literals without any suffix as int

By default, floating-point literals in Java are considered to be of type double. For example, if you write double y = 3.14;, the value 3.14 is treated as a double. This design decision was made for the improve precision and accuracy, common usage.

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

Answer:

Implicit narrowing primitive conversions, also known as automatic narrowing conversions, only take place among byte, char, int, and short because these data types have the same size (16 bits or 32 bits) and similar ranges of representable values. This similarity allows Java to perform these conversions safely without the risk of data loss.

The data types byte, char, int, and short are all considered integral types in Java:

1. byte: 8 bits, range: -128 to 127 (signed)

2. char: 16 bits, range: 0 to 65535 (unsigned)

3. int: 32 bits, range: -2^31 to 2^31 - 1 (signed)

4. short: 16 bits, range: -32,768 to 32,767 (signed)

The key reason for allowing implicit narrowing conversions among these integral types is that their value ranges partially overlap, and no information is lost when converting from a larger data type to a smaller one within these overlapping ranges.

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

Primitive Conversion?

Answer:

A widening conversion, also known as implicit promotion, occurs when a value of a smaller data type is automatically and safely converted to a value of a larger data type. For example, converting from **`byte`** to **`int`**, **`short`** to **`long`**, and **`int`** to **`float`** are all examples of widening conversions

A narrowing conversion, also known as explicit casting, occurs when a value of a larger data type is explicitly converted to a value of a smaller data type. For example, converting from **`int`** to **`byte`**, **`long`** to **`short`**, and **`double`** to **`float`** are examples of narrowing conversions.

the conversion from **`short`** to **`char`** does not fit the strict definitions of widening or narrowing conversions, Java allows this conversion due to the historical design and the similarity in their size and representable range. It is important to understand the distinctions between widening, narrowing, and other special conversions in Java to ensure proper type handling and avoid potential data loss.