Programming Fundamentals - Assignment 2

Similar to the first assignment, you should start by initializing a new Git repository and adhering to best practices. Create a blank text or Word document named 'Programming-Fundamentals-Assignment-2' and add it to your repository. This assignment consists of ten questions, the majority of which align with the topics we've discussed during the course. However, some questions might necessitate a bit of independent research. Make sure to commit after answering each question.

Once you've answered all the questions, convert your document into a PDF file and remove the original text or Word document from the repository. This should be your final commit. Subsequently, establish a new cloud repository in your GitHub Playground to host your local repository, and push everything to this GitHub cloud repository.

When the reviewer navigates to your GitHub repository, they should only see the PDF file; no other files should be visible. Finally, once you've completed all these steps, submit the link to your GitHub repository on Google Classroom.

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	**Dynamically**	**Strongly**	**Loosely**
In a statically typed language, the data types of variables are known and checked at compile-time.	In a dynamically typed language, the data types of variables are determined at runtime.	In a strongly typed language, the type of a variable is strictly enforced, and type conversions are not automatically performed unless explicitly defined by the programmer.	In a loosely typed language, the type of a variable is not strictly enforced, and type conversions can happen implicitly or automatically. Variables can change their type during execution, and operations between different data types may not generate errors or may perform implicit conversions.

Java is not dynamically typed. It is a statically typed language, which means that the types of variables and expressions are known at compile time. This means that Java compilers can catch type errors before the program is run, which can help to prevent runtime errors.

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?"

In a case-sensitive programming language, the distinction between uppercase and lowercase characters is significant. This means that variables, function names, keywords, and any other identifiers in the code are treated as distinct and separate entities based on the case of their letters. For example, "Variable", "variable", and "vArIaBlE" are considered three different identifiers in a case-sensitive language.

In a case-insensitive programming language, the distinction between uppercase and lowercase characters is not considered significant. This means that variables and identifiers are treated as the same regardless of the case of their letters. For example, "Variable", "variable", and "vArIaBlE" are treated as the same identifier in a case-insensitive language.

Java is a case-sensitive programming language. Identifiers such as variable names, method names, class names, etc., are treated as distinct and separate entities based on the case of their letters. For example, in Java, "age", "Age", and "AGE" are considered three different identifiers.

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

In Java, Identity Conversion is a type conversion or casting that occurs when a value is assigned to a variable of the same data type without any explicit casting. It is the most straightforward type of conversion because the data types of both the source and destination variables are the same. Identity conversion is a safe operation, as there is no risk of data loss or precision issues.

int x = 10 int y = // Identity conversion: assigning the value of 'x' to 'y' without explicit casting

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

Primitive widening conversion in Java is the automatic conversion of a smaller primitive data type to a larger primitive data type without any loss of information. For example, an int can be automatically converted to a long, or a char can be automatically converted to an int.

```
public class Demo {
    byte b = 10;
   int i = b; // widening conversion from byte to int

   short s = 20;
   long l = s; // widening conversion from short to long

  char c = 'A';
  int i2 = c; // widening conversion from char to int
    }
}
```

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

Compile-time constants are constants whose values are known and determined at compile-time, EX, during the compilation phase of the program. The compiler replaces references to these constants with their actual values directly in the bytecode, resulting in more efficient code execution.

```
public class Demo {
    final int AGE = 30;
    final String NAME = "John";

    public static void main(String[] args) {
        int totalAge = AGE + 5; // The value of AGE is known at compile-time.
        System.out.println("Name: " + NAME + ", Age: " + totalAge);
    }
}

```

Run-time constants are constants whose values are determined at runtime, during the execution of the program. These constants cannot be evaluated by the compiler during compilation, and their values are calculated during program execution.

```
public class Demo {
    final int AGE = getRandomAge();

    public static int getRandomAge() {
        Random random = new Random();
        return random.nextInt(50) + 20; //dont know the age before running
    }

    public static void main(String[] args) {
        System.out.println("Random Age: " + AGE);
    }
}
```

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 and explicit narrowing conversions (casting) are two types of primitive type conversions in Java.

Implicit (automatic) narrowing primitive conversions are conversions that are performed by the Java compiler automatically. These conversions are safe, as they do not lose any information. For example, an int can be implicitly converted to a byte, or a long can be implicitly converted to a short.

Explicit narrowing conversions (casting) are conversions that are performed by the programmer explicitly. These conversions can be unsafe, as they can lose information. For example, a long cannot be explicitly converted to an int without losing the value of the lower 32 bits of the long.

The following conditions must be met for an implicit narrowing primitive conversion to occur:

The destination type must be smaller than the source type.

The value of the source type must be representable in the destination type.

For example, the following implicit narrowing primitive conversions are allowed:

int -> byte

long -> short

float -> byte

The following implicit narrowing primitive conversions are not allowed:

long -> int

double -> byte

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?"

long data type (which is 64 bits) to a float data type (which is 32 bits), there can be a potential loss of precision. This is because float uses 32 bits to represent a floating-point number while long uses 64 bits to represent an integer value.

The discrepancy arises due to the fundamental differences between how integers (represented by long) and floating-point numbers (represented by float) are stored in memory.

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?

The default data types `int` for integer literals and `double` for floating-point literals in Java were chosen for historical performance and memory considerations, and to align with common mathematical conventions, making the language beginner-friendly and easy to use.

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

Implicit narrowing primitive conversion in Java only takes place among byte, char, int, and short because they have a well-defined and consistent size relationship, ensuring safety and avoiding ambiguity in conversions. Other data types require explicit type casting to handle potential loss of precision.

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

The conversion from `short` to `char` is not classified as widening or narrowing primitive conversion because it is a "numeric promotion" where no information is lost. It can be done implicitly without explicit type casting.

Programming Fundamentals - Assignment 2 2