Assignment 02

(01) A statically typed programming language is one in which the type of a variable is known at compile time. This means that the compiler can check for type problems before executing the program. Statically typed languages are frequently seen as more dependable and efficient than dynamically typed languages. C, C++, Java, and Pascal are examples of statically typed languages.

A dynamically typed language is one in which the type of a variable is unknown until runtime. This means that the compiler cannot check for type problems before executing the program. Dynamically typed languages are typically thought to be more expressive and flexible than statically typed languages. Python, Ruby, and JavaScript are examples of dynamically typed languages.

Strongly typed language is a programming language in which type conversions are checked strictly. This means that the compiler will not allow a value of one type to be assigned to a variable of another type, unless the types are compatible. Strongly typed languages are often seen as being more reliable than weakly typed languages. Some examples of strongly typed languages include C, C++, and Java.

Loosely typed language is a programming language in which type conversions are checked loosely. This means that the compiler may allow a value of one type to be assigned to a variable of another type, even if the types are not compatible. Loosely typed languages are often seen as being more expressive than strongly typed languages.

Java is a statically typed language. This means that the type of a variable is known at compile time, and the compiler will check for type errors before the program is executed. Java is also a strongly typed language, which means that type conversions are checked strictly. This makes Java a reliable and efficient language, but it can also make it more difficult to learn and use.

02) In programming languages, case sensitivity refers to whether the capitalization of identifiers (such as variable names, function names, and class names) is significant. In a case-sensitive language, the capitalization of an identifier matters. For example, the identifiers counter and Counter are distinct in a case-sensitive language. In a case-insensitive language, the capitalization of an identifier does not matter. For example, the identifiers counter and Counter are the same in a case-insensitive language.

Here are some examples of case-sensitive and case-insensitive programming languages:

Case-sensitive languages: C, C++, Java, C#, Python, Ruby, Swift
Case-insensitive languages: PHP, BASIC, Pascal, Ada, SQL

Case sensitive-insensitive languages are a bit of a hybrid. In these languages, the capitalization of an identifier is significant only in certain contexts. For example, in the language Haskell, the capitalization of an identifier matters when the identifier is used to define a type. However, the capitalization of an identifier does not matter when the identifier is used to refer to a value.

03)

```
int i = 10;
int j = i;
```

In this code, the value of i, which is 10, is assigned to the variable j. This is an example of identity conversion because the type of the value being assigned (an int) is the same as the type of the variable being assigned to (also an int).

```
String s1 = "Hello";
String s2 = s1;
```

In this code, the value of s1, which is the string "Hello", is assigned to the variable s2. This is an example of identity conversion because the type of the value being assigned (a String) is the same as the type of the variable being assigned to (also a String).

04)

In Java, primitive widening conversion is a type conversion that turns a smaller primitive type to a bigger primitive type automatically. The following code, for example, demonstrates a primitive widening conversion.

```
byte b = 1;
int i = b;
```

When the value of b, which is a byte, is assigned to the variable i, it is automatically transformed to an int. This is due to the fact that an int is a larger primitive type than a byte, and there is no information loss when converting a byte to an int.

Here is a diagram that shows the widening primitive conversions in Java:

```
byte < short < int < long < float < double
```

As you can see from the diagram, a byte can be widened to a short, an int, a long, a float, or a double. A short can be widened to an int, a long, a float, or a double. An int can be widened to a long, a float, or a double. A long can be widened to a float or a double. A float can be widened to a double. Here are some other examples of primitive widening conversion in Java:

Java

```
short s = 1;
long l = s;
float f = l;
double d = f;
```

05)
Compile time constants:

- Compile-time constants are values that the compiler knows and evaluates during the compilation phase.
- At compile time, they are substituted with their real values, and the generated code incorporates these values directly.
- Compile-time constants are often used for values that can be determined at compile time without the need for computation or access to other resources.

Run-Time Constants:

- Run-time constants are values that can only be assessed at runtime during program execution. They are determined by the state of the program or may necessitate calculation or access to other resources.
- Unlike compile-time constants, the compiler cannot determine their values during the compilation phase, and they are not directly replaced with their values in the compiled code.

06)
Explicit Narrowing Conversions (Casting):

Explicit narrowing conversions, often known as casting, necessitate the programmer explicitly stating their purpose to convert a value from one data type to another. When the target type has a smaller range or precision than the source type and there is a risk of data loss or overflow, this is required. Casting allows the programmer to accept responsibility for the conversion and indicate that they are aware of the potential data loss.

Implicit Narrowing Primitive Conversion Requirements: Certain requirements must be met in order to perform an implicit narrowing primitive conversion:

1. The target data type must be smaller than the source data type.
2. The source value must be in the target data type's range. In other words, it must not exceed the target type's maximum or minimum representable value.
3. The conversion must not result in data or precision loss. Converting a floating-point value to an integer, for example, may result in a loss of decimal places, which must be considered.

07)
When assigning a long data type (64 bits) to a float data type (32 bits) in Java, there is a risk of accuracy loss. The assignment can be done automatically via an implicit narrowing primitive conversion, but the long value may not be exactly representable in the float data type due to differences in size and representation.
Long and float data types represent numbers in different ways:
• long: A long data type is a 64-bit signed two's complement integer. It can represent a large range of whole numbers, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
• float: A float data type is a 32-bit single-precision floating-point number according to the IEEE 754 standard. It may represent a large range of decimal values with little precision. It has an accuracy of about 7 decimal digits and a range of numbers it can represent of about 3.40282347E+38.

08)
The rationale behind setting int and double as the default data types for integer literals and floating point literals in Java is as follows:

• int is the most common data type for integer literals. Most integer literals in Java programs fall within the range of values that can be represented by an int. This means that using int as the default data type for integer literals is efficient in terms of memory usage.

• double is the most common data type for floating point literals. Most floating point literals in Java programs fall within the range of values that can be represented by a double. This means that using double as the default data type for floating point literals is also efficient in terms of memory usage.

•The use of int and double as the default data types makes it simple for programmers to develop portable and efficient code. If a programmer uses int for all integer literals and double for all floating point literals, their code will most likely be portable to other Java-supporting platforms. This is due to the fact that all Java platforms offer the int and double data types.

•

09)
Implicit narrowing primitive conversion only takes place among byte, char, int, and short because these are the only primitive types that can be represented in a single byte of memory. This means that converting a value of one of these types to another of these types can be done without losing any information.

10)

In Java, there are two forms of implicit conversions: widening primitive conversion and narrowing primitive conversion. When a value of a smaller primitive type is converted to a value of a bigger primitive type, this is known as widening conversion. A byte, for example, can be extended to an int, and a short can be broadened to an int. When a value of a bigger primitive type is converted to a value of a smaller primitive type, this is known as narrowing conversion. An int, for example, can be narrowed to a byte, while a long can be narrowed to a short.

Because it is a lossless conversion, the conversion from short to char is not characterized as a widening and narrowing primitive conversion. Because a short can represent all of the values that a char can, there is no information loss when converting a short to a char. This means that the short-to-char conversion is always safe and does not require any additional handling by the Java compiler.