Programming Fundamentals - Assignment 2

1) **Statically typed language** : In this type languages, the type of a variable or expression is checked at compile time.

   Dynamically typed language: In this type languages, the type of a variable or expression is checked at run time.

   Strongly typed languages: This type of languages strictly considered about the type when creating variables and objects.

   Loosely typed languages: This type of languages are not considered about the type when creating variables and objects.

   Java would fall into all statically typed, dynamically typed and strongly type languages.

2) **Case sensitive** means if the programming language needs to match two strings they can be equal when both strings have same characters respectively. As an example there are two variables called x and y.

   String x = " my name";
   String y = " my Name" ;

   if the programming language is case sensitive above variables are not equal. And if the opposite of this happens in a programming language it is **case insensitive**. Because case insensitive means it ignores the case (lowercase/ uppercase). When considering above example when it is case insensitive both x and y are equal.

   Some programming languages have varying case sensitivity; in PHP, for example, variable names are case-sensitive but function names are not case-sensitive. This is called **case sensitive-insensitive**

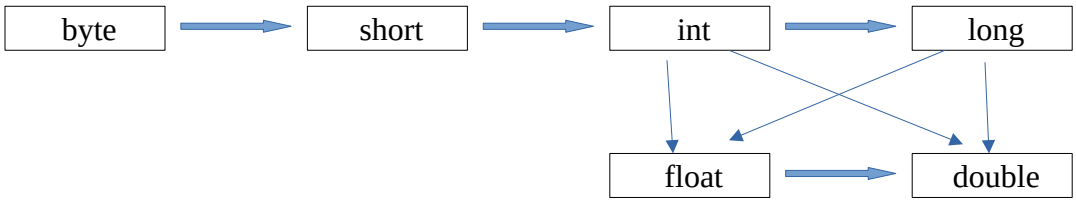
   - **java** is case-sensitive (for identifiers)

3) **Identity Conversion**: Assigning same type variables

   Examples:
   ```
   int number1 = 10;
   int number2 = number1;

   float myFloat1 = 10.55;
   float my Float2 = myFlaot2;
   ```

4) **Primitive widening conversion**



This conversion is happening implicitly when passing a small size type into a larger size type.

Examples:

- byte myByte =10;
- short myShort = myByte; // 16 bits ← 8 bits

- int myInt = myShort; // 32 bits ← 16 bits

- char myChar = 'A';
- int myInt2 = myChar; // 32 bits ← 16 bits

- long myLong = myInt; // 64 bits ← 32 bits

- float myFloat = myInt; // 32 bits ← 32 bits
- float myFloat2 = myLong; // 32 bits ← 64 bits

- double myDouble = myFloat; // 64 bits ← 32 bits
- double myDouble2 = myLong; // 64 bits ← 64 bits

When considering widening conversion from long to float, long to double and int to float precision of the result may loss. That is, the result may lose some of the least significant bits of t he value. Because main objective of a float/double is storing data in a higher range in scientific notation. They do not consider much about the precision. In this case, the resulting floating point value will be correctly rounded version of the integer value according to IEEE 754.

5) **Run-timeconstant** is computed at the run time of application while **compile-time constant** is computed at the compile time. Further more compile time constant has only one value at each time and run time constant would change at each run time.

- final int value1 = 10; // compile-time constant

- final int value2 = 10 * (int) Math.random(); // run-time constant

6) **Narrowing primitive conversion** happens when passing a larger size type into a small size type. **Implicit (automatic) narrowing primitive conversion** occurs when it is following some rules while it should be manually converted in **Explicit(casting) narrowing primitive conversion.**

   Rules that should be satisfied to occur Implicit narrowing primitive conversion :

   - There should be an assignment context
   - The value should be fallen within the range of both types
   - The value should be compile-time constant

7) When considering widening conversion from long to float precision of the result may loss. That is, the result may lose some of the least significant bits of the value. Because main objective of a float/double is storing data in a higher range in scientific notation. They do not consider much about the precision. In this case, the resulting floating point value will be correctly rounded version of the integer value according to IEEE 754.

8) The use of int as a default for integer literals makes it possible to make effective use of memory and computation, which means less space is taken by int relative to another type of integer data types. Similarly, more precise calculations can be made when using double as default for a floating-point literal than with other data types. This design decision also aligns with Java's goal of being a simple and easy-to-use language.

9) Implicit narrowing primitive conversion only takes place among byte, char, int, and short because these are the most commonly used integer data types, and they can be safely converted without loss of information. Explicit narrowing of the conversion is excluded for other data types, such as long and floats, because they have more range and precision as compared to smaller types that can lead to a loss of information when converted.

10) Widening and narrowing primitive conversion occur when both widening and narrowing conversion occur. As an example when converting from byte to char byte value convert into the int type using widening conversion and when converting from int to char narrowing conversion occur.