

1.

Statically Typed Language - Type checking occurs at compile time instead of run time.

Dynamically Typed Language - Type checking occurs at run time.

Strongly Typed Language - Variables are bound to specific data types that will result type errors if types don't match to the expression that is defined.

Weakly Typed Language - Variables are not bound to specific data type. Type safety constraints are lower than strongly typed languages.

Java can be named as Statically Typed Language as well as Dynamically Typed Language. Also it is a Strongly Typed Language.

2.

Case Sensitive - The programming languages which treat uppercase and lowercase letters as distinct and different from each other.

ex: "Hello" and "hello" are considered as different strings.

Case Insensitive - The programming languages which treat uppercase and lowercase letters as equivalent and does not differentiate between them.

ex: "Hello" and "hello" are considered as equivalent strings.

Case Sensitivity-Insensitive -

By considering the above terms, Java can be classified as a Case Sensitive Language.

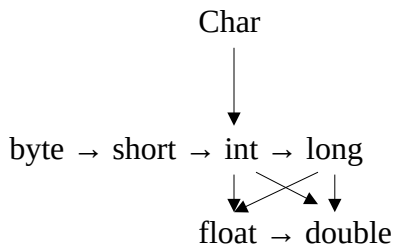
3.

Identity Conversion - It occurs when assign a value to a type that already is, and no actual conversion or modification of the value take place.

ex: int x=50;
 int y=x;
 String item1 = "apple";
 String item2 = item1;

4.

Widening Primitive Conversion - Specific conversions of primitive data types are called as this. This depends on the bit size and does not lose the information about the overall magnitude of the numeric value.



```
J File1.java 9+ x
home > udari > J File1.java > file1 > main(String[])
1 public class file1 {
2
3     Run | Debug
4     public static void main(String[] args) {
5
6         byte myByte = 45;
7         int myInt = myByte;
8         short myShort = myByte;
9         long myLong = myByte;
10        float myFloat = myByte;
11        double myDouble = myByte;
12
13        short myShort1 = 34;
14        int myInt1 = myShort1;
15        long myLong1 = myShort1;
16        float myFloat1 = myShort1;
17        double myDouble1 = myShort1;
18
19        int myInt2 = 30;
20        long myLong2 = myInt2;
21        float myFloat2 = myInt2;
22        double myDouble2 = myInt2;
23
24        long myLong3 = 11224454656l;
25        float myFloat3 = myLong3;
26        double myDouble3 = myLong3;
27
28        float myFloat4 = 12.5657f;
29        double myDouble4 = myFloat4;
30
31        char myChar = 90;
32        int myInt3 = myChar;
33    }
```


```
J File1.java 9+ x
home > udari > J File1.java > file1 > main(String[])
29
30        char myChar = 90;
31        int myInt3 = myChar;
32        long myLong4 = myChar;
33        float myFloat5 = myChar;
34        double myDouble5 = myChar;
35
36    }
37
38 }
```

5.

Run-time constant

- The values which are determined and assigned to variables during the

program execution.



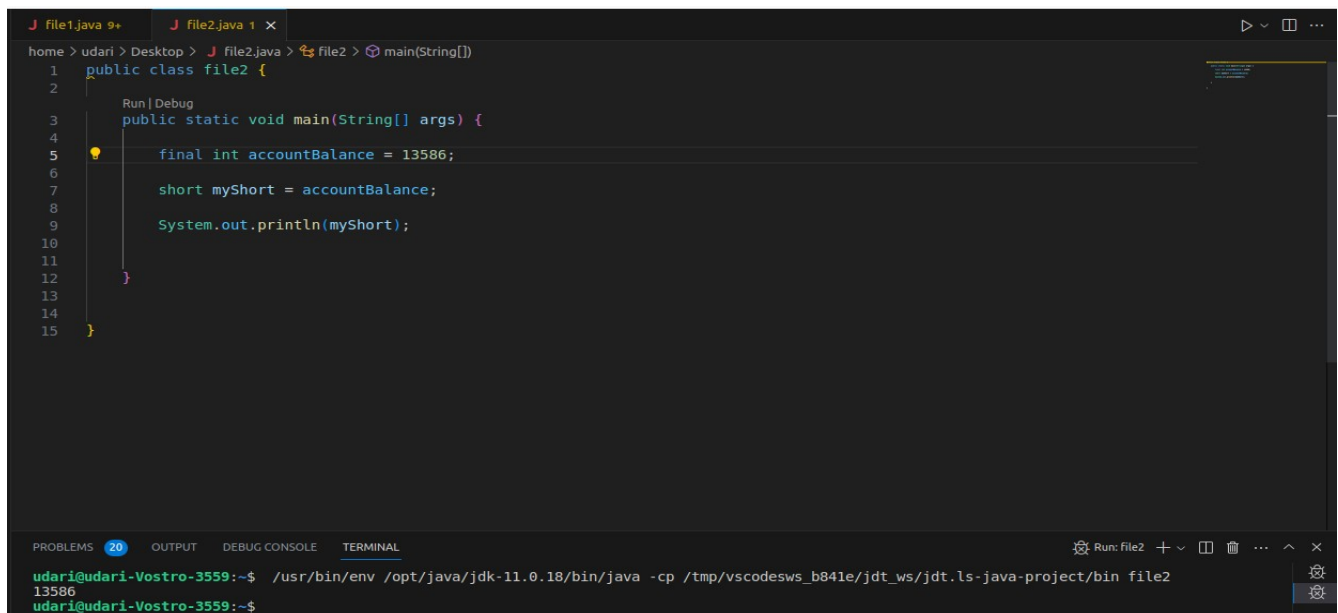
```
1 public class file2 {
2
3     public static void main(String[] args) {
4
5         int accountBalance = 14000;
6
7         int sum1 = accountBalance + 10000;
8
9         short myShort = sum1;
10
11         System.out.println(myShort);
12
13     }
14 }
15 }
```

PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL

at file2.main(file2.java:9)
udari@udari-Vostro-3559:~\$ /usr/bin/env /opt/java/jdk-11.0.18/bin/java -cp /tmp/vscodesws_b841e/jdt_ws/jdt.ls-java-project/bin file2
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Type mismatch: cannot convert from int to short
at file2.main(file2.java:9)
udari@udari-Vostro-3559:~\$

Compile-time constant

- The values which are fixed at the time of code compilation. To define a compile-time constant in java, we use 'final' keyword along with the data type.



```
1 public class file2 {
2
3     public static void main(String[] args) {
4
5         final int accountBalance = 13586;
6
7         short myShort = accountBalance;
8
9         System.out.println(myShort);
10
11     }
12 }
13 }
```

PROBLEMS 20 OUTPUT DEBUG CONSOLE TERMINAL

udari@udari-Vostro-3559:~\$ /usr/bin/env /opt/java/jdk-11.0.18/bin/java -cp /tmp/vscodesws_b841e/jdt_ws/jdt.ls-java-project/bin file2
13586
udari@udari-Vostro-3559:~\$

6.

The difference between Implicit Narrowing Primitive Conversion and Explicit Narrowing Primitive Conversion is how the type conversion is occurred between larger data type to a smaller data type and whether it requires explicit casting.

There must be met two conditions for an implicit narrowing primitive conversion to occur?

- * The assigned value must be a constant.
- * The destination type can fully accommodate the value of the source type.

7.

This type conversion is occurred due to implicit narrowing primitive conversion. Java allows certain type of conversions between numeric data types. While converting 'long' data type into a 'float' data type, the compiler truncates the least significant bits of the 'long' fit into the 'float' format.

8.

Int data type can be stored values from $2^{(-31)}$ to $2^{(31)}-1$. So, rather than using 'byte' or 'short' as the default data type for integers, 'int' data type is very useful. However, 'long' data type can store larger number of values, it needs to add a suffix explicitly by the programmer. Moreover, 'long' data type decrease the efficiency where as int would suffice.

The key difference between a float and double in Java is that a double can represent much larger numbers than a float. Therefore, 32 bit floating point with only ~7-8 decimal digits of precision was problematic for most applications that need to use floating point.

9.

Java allows implicit narrowing primitive conversion to smaller set of data types only because it can gain ease of use, type safety, and efficiency. If it allows that conversion to the larger set of data types like 'long', 'float' or 'double', it can lead to data loss, as values may be truncated when converting from a larger value to a smaller value.

10.

Widening and Narrowing Primitive Conversion - This conversion combines both widening and narrowing primitive conversions.

The reason for not classifying the 'short' to 'char' as Widening and Narrowing Primitive Conversion is the range.

Both 'short' and 'char' data types have same number of bits that is 16 for representing the values. However, 'char' is a unsigned integer that holds only positive integers including 0. ($0 - 2^{16}$). 'short' is a signed integer that holds both positive and negative integers including 0. ($2^{(-15)}$ to $2^{(15)}-1$). Therefore, while compilation occurs, all the 'short' type values can be stored as 'char' values implicitly without considering the sign bit of the 'short' value. Therefore, that conversion cannot be classified as Widening and Narrowing Primitive Conversion. It is a Widening Primitive Conversion.