Class 04

Java String Basics

In Java, a **String** is a not a primitive data type rather it is a reference data type that represents a sequence of characters. It's widely used to store and manipulate text. Here's a brief explanation for beginners:

1. **Text Storage**: A **String** can store letters, numbers, symbols, and even empty spaces. For example, **"Hello, World!"** is a **String** that contains a greeting.

2. **Immutable**: Once a **String** is created, it cannot be changed. If you want to modify it, a new **String** object is created. For example, if you concatenate two strings like **"Hello" + "World"**, it creates a new **String** with the value **"HelloWorld"**, leaving the original strings unchanged.

3. **Creating Strings**: You can create a **String** in Java by enclosing text in double quotes. For example, **String greeting = "Hello";**.

4. **String Concatenation**: You can combine strings using the **+** operator. For example, **String message = "Hello" + "World";** creates a new string **"HelloWorld"**.

5. **Length**: You can find the length (number of characters) of a **String** using the **length()** method. For example, **int len = greeting.length();** sets **len** to 5 for the string **"Hello"**.

6. **Accessing Characters**: You can access individual characters in a **String** using indexing. For example, **char firstChar = greeting.charAt(0);** gets the first character **'H'**.

7. **Comparing Strings**: To compare two strings, you should use the **equals()** method. For example, **boolean isEqual = greeting.equals("Hello");**.

8. **String Manipulation**: Java provides many built-in methods for manipulating strings, like converting to uppercase, lowercase, or replacing substrings.

   *****String Manipulation will be discussed more in detail when we learn more about Collection Classes and Object**

Remember that strings are essential for working with text in Java, and they have many useful methods for various operations.

Type Conversion/Casting in Java

In Java, type conversion and casting are fundamental concepts when working with different data types. They allow you to convert a value from one data type to another. Let's break down these concepts:

In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
  byte -> short -> char -> int -> long -> float -> double

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  double -> float -> long -> int -> char -> short -> byte

1. **Type Conversion**:

Type conversion, also known as type casting, is the process of converting a value from one data type to another implicitly or explicitly. There are two types of type conversion:

- **Implicit Type Conversion (Widening)**: This happens automatically by the compiler when it is safe to do so. For example, converting an **int** to a **double**:

java code:

int myInt = 5;

double myDouble = myInt; // Implicit conversion from int to double

- **Explicit Type Conversion (Narrowing)**: This requires a manual cast and is done when there's a risk of losing data. For example, converting a **double** to an **int**:

java code:

double myDouble = 3.14;

int myInt = (int) myDouble; // Explicit conversion from double to int

When performing explicit type conversion, you use parentheses to indicate the target data type, followed by the value you want to convert.

2. **Casting**:

Casting is the act of explicitly converting a value from one data type to another, usually in cases where an implicit conversion wouldn't be safe. Casting is done using parentheses followed by the target data type.

- **Casting for Numbers**:

java code

double myDouble = 3.14;

int myInt = (int) myDouble; // Casting double to int (loses decimal part)

As we know Class/Object are also a kind of user defined and reference data type in Java. Which is different from primitive data types. Java allows you to casting to objects once it is compatible to between two different class types.

- **Casting for Objects**:

When working with objects, casting is used to treat an object of one class as an object of another class, but this is only possible if there is an inheritance relationship between the two classes.

java code

// We will learn more about it once we learn Object

Keep in mind that casting may result in runtime errors if the conversion is not possible or safe.

Remember, type conversion and casting should be used with care to avoid data loss or unexpected behavior. Always ensure that the conversion is necessary and safe for your specific use case.

Statement and Expressions in Java

In Java, statements and expressions serve different purposes in a program. Here are the key differences between them:

Definition:

- **Statement**: A statement in Java is a complete unit of code that performs a specific action. Statements are used to control the flow of a program, declare variables, and perform various tasks. Examples of statements include variable declarations, assignments, loops, conditionals, and method calls.
- **Expression**: An expression in Java is a combination of variables, operators, and method invocations that produces a single value. Expressions are used to compute values or perform calculations. Examples of expressions include mathematical calculations like **5 + 3**, method calls that return values, or even simple variables like **x**.

Purpose:

- **Statement**: Statements are used for straight forward actions. They typically perform actions, such as changing the state of variables, controlling program flow, or performing I/O operations. Statements may or may not return a value.
- **Expression**: Expressions are used to compute values. They always return a value, and their primary purpose is to produce a result that can be used in assignments, method calls, or other expressions.

Examples:

- **Statement Examples**:
  - Variable declaration: **int x;**
  - Assignment statement: **x = 10;**
  - Conditional statement: **if (x > 5) {
    /* code */ }**
  - Loop statement: **for (int i = 0; i < 5; i++) { /*
    code */ }**

- **Expression Examples**:
  - Arithmetic expression: **int result = 5 + 3;**
  - Method call that returns a value: **int length = str.length();**
  - Boolean expression: **boolean isTrue = (x > 5) && (y < 10);**
  - Value of a variable: **int value = x;**

Result/Value that we get from Statement/Expression:

- **Statement**: Statements may or may not produce a value. For example, a method call statement may not return a value, whereas an assignment statement changes the value of a variable.
- **Expression**: Expressions always produce a value. The result of an expression can be assigned to a variable or used directly in another expression.

In brief we can say statements are used to perform actions and control program flow,

Assignment statement: x = 10;

while expressions are used to compute values.

int length = str.length();

Understanding the distinction between statements and expressions is important when writing and reading Java code, as it helps ensure that code is both correct and readable.