1.Variables

Recap:

In the previous chapter, we learned about the control flow of code execution, focusing on the concepts of JDK, JRE, and JVM and their importance. We used JDK 17, which is a Long-Term Support (LTS) version. It's important to note that a new version of the JDK is released every six months, each bringing new features and improvements.

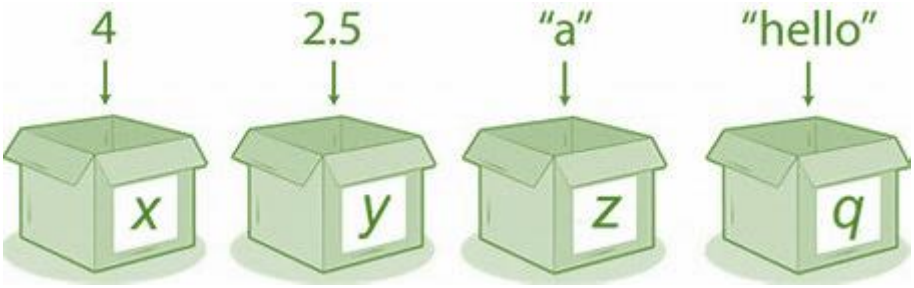
Variables:

Why do we build applications? Why do we code? These questions often arise, and the answer is simple: to solve real-world problems. For example, Amazon provides a convenient way to shop without visiting physical stores, and food delivery apps let us order meals without going to restaurants.

All these applications rely on large amounts of data to function effectively. This data is processed and stored in a persistent storage system called a database, where it can be retrieved and updated as needed. During data processing, we need to store data temporarily; this is where variables come in. Variables act as containers that temporarily hold data while we perform operations on it.

Data can come in various forms, such as text, whole numbers, integers, or decimals. To manage these different types of data, we use specific data types for our variables. For instance:

- An integer like 8 is represented by the int data type.
- Text data is represented using the String data type.



According to the image, as we can see, variables x, y, z, and q are the boxes that hold the data inside it.

Code Example:

Let's consider a simple example where we use two numbers and perform addition:

- Let num1 be the first variable with a value of 8, and num2 the second variable with a value of 5. Both are of the int data type.

- To store the result of their addition, we use another variable called result, which is also of int datatype.

```java
class hello{

    public static void main(String[] args) {
        int num1=8;
        int num2=5;
        int result=num1+num2;
        System.out.println(result);
        }
}
```

```
E:\Telusko>javac hello.java

E:\Telusko>java hello
13
```

Each time we modify our code, we must compile it before running the program. This step is crucial because it converts our code into bytecode, which the Java Virtual Machine (JVM) executes. Changes in the code affect the bytecode, which can alter the program's output.

Although we could directly use numbers in our operations, using variables allows us to store and manipulate data dynamically, making our code more flexible and maintainable.

Creating variables in Java

When we create variables in Java, there are a few key steps to follow:

1. **Specify the data type**:
    - The first step in creating a variable is to define its data type. The data type determines what kind of data the variable can hold (e.g., `int` for integers, `double` for decimals, `char` for characters).
2. **Give the variable a name**:
    - After specifying the data type, assign a suitable name to the variable. The name should be meaningful and follow Java's naming conventions (e.g., `camelCase` for variable names).
3. **Assign a value**:

You can assign a value to the variable at the time of creation by using the assignment operator `=`. This step is known as initialization. For example:

```java
int number = 10;
```

-

Alternatively, you can declare the variable without assigning a value. In this case, the variable will hold a default value, depending on its data type. For example:

```
int number;
```

-
    - Here, `number` will be initialized with a default value of `0`.

4. **Default Values**:
    - If you choose not to assign a value during variable declaration, Java automatically assigns a default value based on the data type:
        - `int`: 0
        - `double`: 0.0
        - `char`: '\u0000' (null character)
        - `boolean`: false
        - Objects (e.g., Strings): `null`

Steps to Remember While Coding:

1. **Use semicolons correctly:** Ensure each statement ends with a semicolon to avoid syntax errors.
2. **Correct Use of Curly Braces ({}):** Curly braces define code blocks. Missing braces can prevent your code from compiling successfully.
3. **Proper Indentation:** Indentation makes your code more readable and easier to understand for others. It also helps in visualizing the structure and flow of the code.