Identifiers and Variables

In the previous notes, we decomposed the main( ) method program and learnt various variants of the main( ) method and understood some basic concepts. Now let's understand what are identifiers.

Well, from the word itself, Identifiers means something through which we can identify something uniquely. These identifiers are meant for identification of any element in the program uniquely such that they are differentiated from the other members of the program.

By definition, **Identifiers** mean some **names** which are used for **identification purposes** and are **unique** throughout the program. We can say that all the names that we actually define other than the common syntax of Java is an identifier.

Just like parents name their children with some name, Identifiers also must have **legit** names. There are several rules which specify what names of identifiers are beneficiary to use and what actually Java does allow.

For example :

```
double variable = 10.32d;    // This means that the 'variable' is an identi
```

Let's make your brain work out. Try answering the given question.

```java
public class Demo {
  public static void main(String[] args){
      int hell = 10;
  }
}
```

Predict the number of identifiers that exist in the given above program.

Well if you guessed that it is *4*, then you are wrong. As mentioned above that an *Identifier can also be a class name.* But I guess, you did overlook **String** class. Hence, it makes it to count for *5 identifiers.* There is a factor to note that all **variables** are **identifiers** but all *identifiers are not variable.*

Rules for naming an Identifier: -

1. The first and foremost rule is that all identifiers are **Alphanumeric**. All Identifiers can only **Alpha + Numeric**. This can be remembered in this way that ***First comes Alpha* then comes *Numeric*.** So it makes it quite clear for a lot of rules for identifiers that they cannot contain any digit in the beginning.

2. Special Digits like **Underscore and $** is only allowed as the beginning of the identifier. They have the right of being used anywhere in the program. If you are thinking why the Java People included $ as our friends from *PHP* are used to use *$.*

3. Well, an identifier must be a *legit word.*

4. An identifier **can be of any length** but it *doesn't* mean that we can go on writing a long name. Max to max, you can use *15 characters at most.*

5. Java is **case sensitive**. So if I write "KRISH" and "krish", it differs a lot.

6. An identifier **can not be a Keyword or a Reserved word** in Java. Neither can it have any **Whitespace.**

> 🛠️ What is the difference between Identifier and Variable?

Test Yourself on Identifiers.

1. How many identifiers are present in the below given program.

   Eg: String is an **identifier**. Hence *count it only once.*

```
public class Hello{
    public static void main(String[] args){
        String melody = "Why is Melody so Chocolatey";
        String answer = "Eat Melody! And that itself will give yo
u the answer.";
    }
}
```

2. **Predict** the **first line** where the **error may originate(if any)** and **explain the reason**. : -

```
    public class Demo1{
        public static void main(String[] args){
            int 3_ide = 20.0;    //1
            double d1 = 22_00_3__.323;    //2
            String str = "@Auth Krish";    //3
        }
    }
```

3. ***Can we have a deprecated reserved word as an identifier? State the reason for the same.***

4. Which is/are a correct identifier/s?

   ☐ ___dent

   ☐ _:__dent

   ☐ dent___

   ☐ _$

   ☐ _transient

   ☐ goto

5. Explain about ***Identifiers in your own words.***

Variables

Variables are nothing but some containers which store some values. These are indeed identifiers. For example : A swimming pool or an overhead tank is a container. Just like the container boxes in your house which store the spices, variables also store **something.** This **something** refers to a *value*.

> 🛠 This is must that the **value** you want to store in a **variable** must be of the **same type.** This means if the type of data a variable can store is *int*, then it can store only values of *Integer* type.

```
int integerType = 20.34; // Syntax Error : Possible Loosy Conversion from
```

Java defines the **Primitive Datatypes** in 4 parts: -

1. Integral Types

2. Floating Types

3. Boolean Type

4. Characters

Also, Java defines 3 types of Variables namely :

1. Instance Variables

2. Class Variables

3. Local Variables

Understanding more about Datatypes.

Datatypes are basically the type of Data a variable can hold. As in the previous chapter, we used the *int* type to declare a variable that stores a value **10**.

Water is a Datatype since it defines what type of data the container or variable would store.

For example, **water** in a swimming pool is a *datatype*. You might imagine how. Well it is a basic sense that a swimming pool can store only water. Hence *water* becomes a *datatype* and *swimming pool* becomes a variable.

Swimming Pool stores something. That is why, it is a variable.

In short, a Datatype is something which defines what data is going to be stored in a particular container(variables). Now let's see some examples of using datatypes with variables. But before that, we need to know the segregation of datatypes in Java.

There are two divisions of Datatypes in Java: -

1. Primitive Datatypes : Datatypes that have a **fixed size** and **range** in the eyes of compilation unit of Java are called Primitive Datatypes. There are simply just *8* primitive datatypes in Java.

> 🛠 Note that all the 8 datatypes are reserved words in Java. Reserved Words in Java or Keywords are *special words* which convey some kind of meaning to the Compilation Unit.

Name of Primitive Datatypes

Aa Name	☰ Size	☰ Range
byte	1 byte or 8 bits	-128 to 127
short	2 bytes or 16 bits	-32768 to 32767
int	4 bytes or 32 bits	-2147483648 to 2147483647
long	8 bytes or 64 bits	-9_223_372_036_854_775_808 to 9,223,372,036,854,775,807
double	8 bytes or 64 bits	3.12 e-63 to 4.54e63 -1
float	4 bytes or 32 bits	-1.7e -32 to 3.14e 32 -1
boolean	1 byte reserved (1 bit used)	**true** or **false**
char	2 bytes or 16 bits	0 to 65,535

So as you can see that all these above datatypes have a specific size in *bytes* and have a specific range. This is why we call them Primitive Datatypes. In Java, its always necessary to specify the type of data a variable must store before its use.

For Example:

```
temp = 29;  // Would result in a Syntax Error. This is because the Compil
```

Hence we should specify :

```
int temp = 29; // So that the Compiler makes out that 'temp' can store on
```

Let's see how to use all the above 8 primitive datatypes.

```
public class Demo {
    public static void main(String[] args) {
        byte b = 10;
```

```
        short s = 20;
        int i = 632233;
        long l = 81381731l;
        float f = 23.123f;
        double d = 12323.12313112d;
        boolean bol = false;
        char ch = 'a';
    }
 }
```

2. Non-Primitive Datatypes : Datatypes which do not have a **fixed size or range**. Some examples are classes, interfaces, etc.

Now that we know how to use variables and datatypes, let's write a short program to find the sum of 2 numbers.

```
public class Sum {
  public static void main(String[] args){
    int a = 20;
    int b = 30;
    // Alternative : int a = 20, b = 30;
    int c = a+b;
    System.out.println(c);
  }
}
```

In the above program, we have two variables **a** and **b** having the values **20** and **30** respectively. Now we create another variable *c* in which we store the sum of a and b. To find the sum, we use the '+' operator. Operators are basically *symbols* which are used to perform a particular *mathematical or logical operation* on some values. Here, finding the sum is a mathematical operation. So using a simple Arithmetic '+' to find the sum is the only way as of now.

Then we just simply **print** the value stored in the variable c to the output window. And hence, we come to know how to print the sum of

two numbers on the output window.

But there can be one question :

c = a + b;

Here *a* and *b* are *operands* and '+' is an operator. Operands are *variables* on which *some operation* by an Operator is performed.

Now, the Test Time begins : -