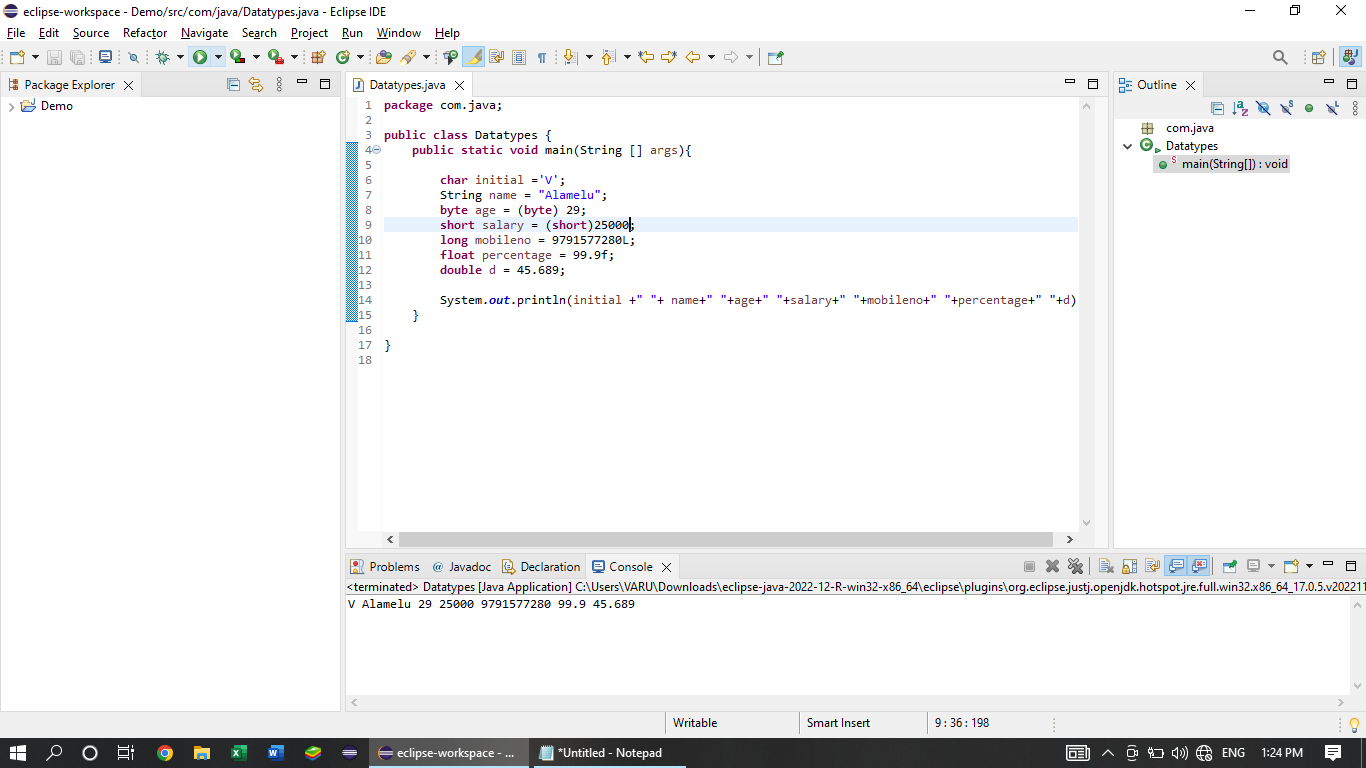
**DATA TYPES**

**Output:**



**Data Types:**

It defines the type of the data.

* Primitive
* Non primitive Data types:

**Primitive:**

We can store the single value for one variable.

Already used datatypes from basic programming languages like C, C++.

It is not a class. So, we cannot use any predefined methods.

**Eg. int a=10;**

**int b=20;**

**int a=10, b=20; (we can give more than one values for a different variable in a single line under one int data type)**

It has 8 types.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Type** | **Memory Size** | **Default Value** | **Wrapper Class** | **Limits** |
| byte | 1 byte | 0 | Byte | -127 to +128 |
| short | 2 bytes | 0 | Short | - 32768 to + 32767 |
| int | 4 bytes | 0 | Int | -2147483648 to +2147483647 |
| long | 8 bytes | 0 | Long | -9223372036854775808 to 9223372036854775807 |
| float | 4 bytes | 0.0 | Float | single precision 32bit IEEE 754 floating point |
| double | 8 bytes | 0.0 | Double | 64bit double precision |
| char | 2 bytes | - | Char | 0 to 65535 single bit Unicode |
| boolean | 1 bit | false | Boolean | simple flags that track true / false conditions |

**DEFAULT DATA TYPE IS INT AND DOUBLE (for floating)**

**\*** For long datatype the values should be mentioned with ‘l’ and ‘L’.

‘long’ used when we use more than 9 digits.

Eg. long ph = 2135468792L;

\* In float and double, we should mention ‘f’ or ‘F’ at last.

Eg. float a = 2.5f; or 2.5F;

**Wrapper Class:**

Converting a **datatype into class** objects.

It is used in collections. Because collections cannot use datatypes. So, we need this Wrapper class.

**Non – Primitive data types:**

It can store multiples value for single variable.

It allows predefined methods.

It is a class. So, we can use the methods.

* String
* Arrays

**CASTING**

Storing the smaller data type values into larger datatype values and vice versa.

**Types:**

**Widening** – Smaller to larger [ byte to long]

Eg. short s= 12345;

int a = s;

**Narrowing** – Larger to smaller [ int to byte]

Eg. int a = 12345;

Byte b= (byte) a;

**Encapsulation**

**Binding the code and data together into a single unit [wrapping up of data into a single unit]**

It can be achieved through **Access Specifiers.**

\* **Public** – The methods in this class can be accessed throughout the project.

Eg. public void methodname()

\* **Private** - The methods in this class can be accessed within the class.

Eg. private void methodname()

\* **Protected** - The methods in this class can be accessed within the package and if it **extends** then it can be accessed throughout the project.

Eg. protected void methodname()

\* **Default** - The methods in this class can be accessed withing the package. It need not the keyword(default) in the program.

Eg. void methodname() – here it considered as default class.

**INHERITANCE**

Process of accessing the Parent / Super class properties.

**“Extends”** Keyword is used to get the relationship between two classes.

**\* Single Inheritance**

Parent class methods

Sub class

**\* Multiple Inheritance**

Parent class 1 methods Parent class 2 methods Parent class 3 methods

Sub class

=> It is not possible in java class due to the priority issue and ambiguity (same name) problems.

=> It can be overcome by **INTERFACE**

**\* Multi level Inheritance**

Parent class (Grand Parent)

Sub class (Parent)

Sub class (child)

Sub class (Grand child) (it can access all the above methods)

\* **Hierarchical Inheritance**

Parent class

Sub class 1 Sub class 2 Sub class 3 Sub class 4

\***Hybrid Inheritance**

**=>** Combination of two inheritances.

=> There is not possibility in java class because of Multiple inheritance issue.

=> If we combine multiple inheritance with other inheritance there will be an issue.

=> We can overcome this problem by **Interface.**

If we want to access the default method to another or new package, we must **import** the previous package first.

Eg. **import** com.java.concepts.AccessSpecifiers;

Then we can inherit the properties of the previous package’s **class and methods**.

We can create Multiple objects in a single package(class) or various packages and class.

Because if we want to access the one package’s methods to another package we must use **‘Extends’** keyword, even though for a protected and default method we can’t use the methods. So, if we need to call the methods from different classes from different packages, we can create the new object for the class and we can get an output.

**POLYMORPHISM**

One method can be used in different ways.

We cannot create same method name in a single class.

It can be achieved by,

**=> Methods Overloading (or) Compile time Polymorphism (or) Static Binding**

**\*** Happens in a single class (Same class)

\* Method name should be same

\* But parameters should be different.

Here,

-> We can add the parameter.

Eg. public void std(int sno){

System.out.println(sno ) }

-> We can add increase the count of a parameter.

Eg. public void std(int sno, char initial){

System.out.println(sno, initial) }

-> We can add change the order of the parameter.

Eg. public void std(char initial, int sno){

System.out.println(sno, initial ) }

**=> Method Overriding**

\* Different class names

\* Same Method names

\* Parameters should be same

\* It should be extended with another class

\* While running child method is run override will be executed.

If we create an object for the parent class in child class, the methods in the parent class only executed.

Even though in overriding.

**ABSTRACTION**

Hiding the Implement part is called abstraction.

Eg.

Public void login() // method declaration (or) Signature Part

{ // Method definition (or) Implementation Part

Functions;

}

Types:

1. Abstract class or partial abstraction
2. Interface or full abstraction

**Abstract class or partial abstraction:**

Accept both abstract and non- abstract method

Abstract keyword is must

We cannot create object for abstract class

It should be extended to another class

All the unimplemented methods should be added in that child class

**Interface or full abstraction:**

Only abstract methods allowed

Abstract keyword is not necessary

We cannot create object for Interface

We should implement the interface to a class using ‘implements’

All the unimplemented methods should be added in child class

We can implement more than one interface at same time

We can achieve multiple inheritance.

**VARIABLES**

To Store a value, we can use variables.

Eg. int a = 10; // variable declaration with **initialization**

String s; // variable declaration without initialization

Int b;

a = 20; // new value **assigned**

**Types:**

\* Class variable or Instance variable

-> It can be declared inside the class outside the method.

-> It can be declared without initialization

-> Default value will be printed if it is not initialized.

-> Its lifecycle will be throughout the class.

-> It can be accessed inside the static method.

\* Local Variable

-> It can be declared inside the method

-> It should be initialized

-> Its lifecycle will be inside the method only.

-> If local variable and class variable has same name then the priority will be gone for local variable. Because local variable cannot go to another method.

// colour for local variable is grey..

Colour for class variable is blue.. //

\* Final Variable

- final keyword is must

- final variable value cannot be changed

- we can declare final keyword inside the outside the method

\* Final method

- It cannot be over rided

- It can be overloaded.

\* Final class

- It cannot be extended.

\* Static variable

- Static keyword is must

- It can be declared inside the class and outside the method

- It can be declared without initialization.

- default value will be printed if the variable is declared without initialization.

- It0s lifecycle will be throughout the class

- It can be accessed without objects.

- static keyword cannot be used inside the method.

Static Method:

- It can be called without object.

- static variables only allowed inside the static method.

**CONSTRUCTOR**

- Constructor name and class name should be same

- When an object is created the default constructor will be executed automatically

- Constructor doesn’t have any return type (void)

- It can be **overloaded and cannot be over rided.**

- This or super must be the first statement in the constructor.

**Two keywords:**

**-** This ---🡪 Current class reference (obj)

- Super --🡪 Parent class reference (obj)

**SCANNER**

-> It is a class

-> to get input at run time.

Scanner sc = new scanner(system.in);

system.out 🡪 Print output in console.

System.in 🡪 Get input from console.

In Primitive datatype

Sc.nextByte();

Sc.nextShort();

Like these except char datatype all the primitive datatypes represented as sc.nextDATATYPE is used in scanner class.

It accept the input in green colour

It will give the output in black colour.

If we want to check the output we have to check “terminated and class name in upper side”.

**String:**

Sc.next() /// we can use only one word

Sc.nextLine() /// we can use multiple words

If we want to use multiple lines We should call the sc.nextline() method first and sc.next() method should be at last.

**Char:**

Sc.next().charAt(index);

// index value always starts from 0 in java and selenium//

Sc.close(); // we can stop to get the input using this scanner method//

**CONTROL STATEMENTS**

\* Loop Statements

- for

- while

- do-while

Syntaxes:

**For loop:**

for(initialization,condition,iteration){

}

**While loop:**

Initialization

While(condition){

Syso();

Iteration;

}

**Do-while loop:**

Initialization;

Do{

Syso;

Iteration;

}while(condition);

‘=’ - value assigning

‘==’ - check whether both the values are same

‘!=’ - checks the value is that not equal to the given value

‘<=’ - checks the value is lesser than or equal to the given value

‘<’ - checks the value is lesser than to the given value

‘>=’ - checks the value is greater than or equal to the given value

‘>’ - checks the value is greater than to the given value