

JAVA VARIABLES

(Non Primitive Data Types In Depth)

⇒ Reference Data Types / Non-Primitive Data Types

* There are mainly 4 type of reference data types:

- Class
- String
- Interface
- Array

* What is reference ?

Let's understand with an example of class

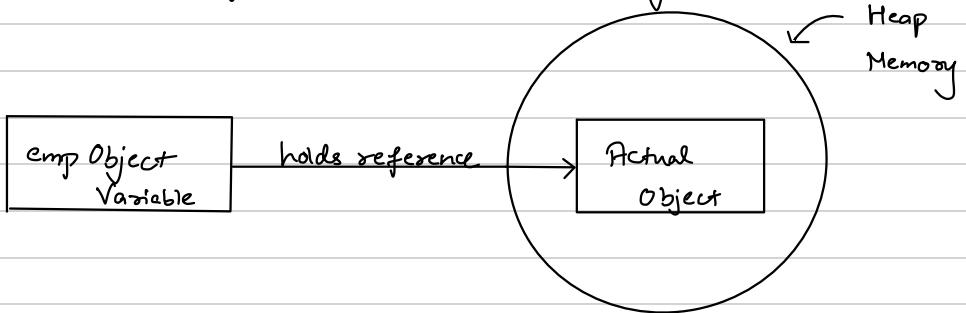
So let's create a class

```
public class Employee {  
    int empId;  
  
    public int getEmpId() {  
        return empId;  
    }  
  
    public void setEmpId(int empId) {  
        this.empId = empId;  
    }  
}
```

Now to create an object of class Employee

```
public class Student {  
    public static void main(String args[]){  
        Employee empObject = new Employee();  
    }  
}
```

new keyword allocates a memory block object & the variable's name holds a reference to actual memory.



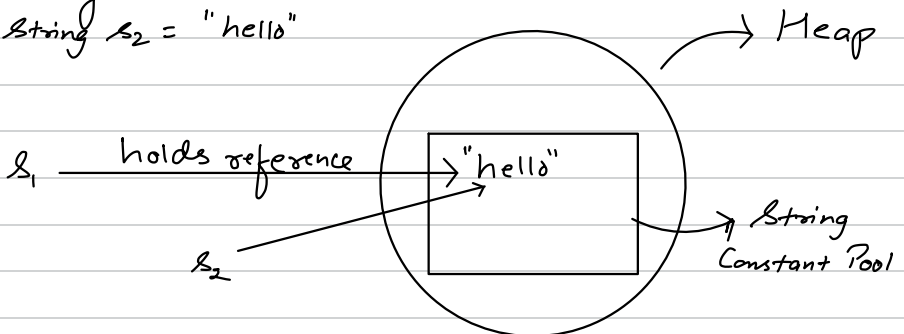
* In Java, everything is pass by value. So with the help of reference variables we're achieving the functionality of pointers in C++.

* String

- Strings are immutable in Java.
- It contains String literal.

Inside heap, there is a fixed memory space k/a String Constant Pool. So the String variable holds a reference of corresponding String literal in String Constant Pool.

Eg:- String s₁ = "hello".
String s₂ = "hello"



Since strings are immutable, 2 same strings holds reference to same string literal in String Constant Pool.

So both s_1 & s_2 holds reference to same string literal "hello".

This applies for normal strings. If we create using new keyword, it'll be stored as normal object in heap.

Eg:- `String s3 = new String("hello")`

* Interface

To understand better, let's create an interface & implement it.

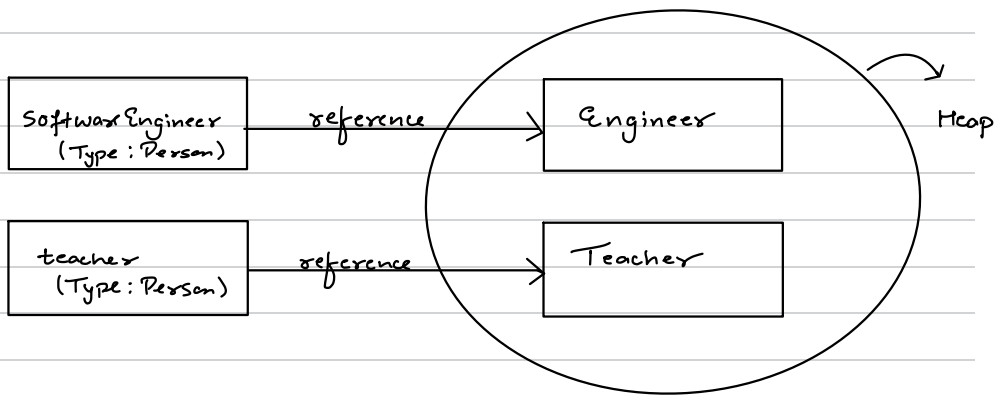
```
public interface Person {  
    public String profession();  
}
```

```
public class Teacher implements Person {  
    @Override  
    public String profession() {  
        return "teaching";  
    }  
}
```

```
public class Engineer implements Person {  
    @Override  
    public String profession() {  
        return "software engineer";  
    }  
}
```

Now let's create a few objects in a class:-

```
public class Student {  
    public static void main (String args []) {  
        Person softwareEngineer = new Engineer();  
        Person teacher = new Teacher();  
        Teacher teacher1 = new Teacher();  
        Engineer softwareEngineer1 = new Engineer();  
    }  
}
```



So here the variables SoftwareEngineer & teacher holds a reference of Engineer & Teacher type objects in the heap memory.

So we can store the objects of a child in a parent one or we can store the objects in same class itself but we cannot create an object of interface.

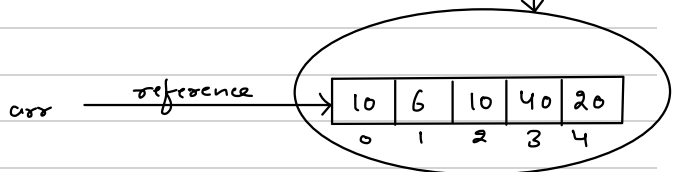
ie `Person person = new Person();`

This is wrong.

⇒ Array

- Sequence of memory storing same data type.

For eg:- `int [] arr = new int [5];` capacity of array
 or `int arr[] = new int [5];`



`arr[0] = 10, arr[1] = 6, arr[2] = 10, arr[3] = 40, arr[4] = 20`

* Arrays can be assigned in multiple ways & can be of multiple types like ID, AD etc.

⇒ Wrapper Class



For each of the primitive data types, we have corresponding reference types that are known as Wrapper Classes

int	————→	Integer
char	————→	Character
short	————→	Short
byte	————→	Byte
long	————→	Long
float	————→	Float
double	————→	Double
boolean	————→	Boolean

* Why the need for wrapper classes?

→ We get the advantages of passing by reference. For example if we've declared a wrapper type of int i.e Integer type, we can change it later on & it'll change in memory as well because in wrapper we're storing reference.

In primitive this won't be possible as they're stored in stack & not heap.

- The collections works on objects only i.e on reference data types, so we need wrapper class to use collections.

* Autoboxing

- To convert a primitive data type to its wrapper.

Eg:- `int a = 10;`

`Integer al = a;`

↖ primitive to its wrapper

* Unboxing :-

- To convert a wrapper class to primitive

Eg: `Integer n = 20;`

`int x1 = n;`

↖ wrapper to primitive

⇒ Constant Variable

- We cannot change the value of a constant variable. This is usually created using final keyword.

Eg:- `Static final VAR = 10;`

↙
It means only
one copy exist

↘
It means the value of
VAR can't be changed.