7) Methods & Constructors:

a)
```
public void fnname (datatype varname)
{
                                        } Method
}
```

b) Passing arguments → $\underbrace{\text{customer}}_{\text{obj}}$.payBill(500,10);

c) Constructors

(i) → Syntax → public Customer() {
& no return type   Class variables to initialize.
}

~~As seen in d) point in last page~~

(ii) If no values are assigned then Java assigns null or 0 to that variable.

(iii) Can be more than one constructors based on number of parameters

If values assigned then:-

(iv) →
```
class Customer{
        public String customerId;
        public String CustomerName;
        public long contactNumber;
        public String address;

    Customer(String cId, String cName, long contact, String add)
        {
            customerId = cId;
            customerName = cName;
            contactNumber = contact;
            address = add;
        }
}
```

```
public class Tester {
        public static void main(String args[]){
                Customer customer1 = new Customer("C103",
                                                  "Jacob",
                                                  503L,
                                                  "Add")
                System.out.println(customer1.customerId)
```

★. this keyword is used when variables & their values have same values in constructor. [Eg:- this.custId = custId].

---

⑧ Memory Management.

a) ⇒ How java manages obj creation in memory?

Memory → Stack (local var & methods).
Memory → Heap (Instance Variables & obj)

⇒ Reference var in stack points at obj created in Heap.
Customer customerobj = new Customer(); → parameters

b) Java automaticly deallocates the memory if obj does not have any reference.

Eg:- After obj created.

Customerobj = new Customer(New parameters)
or
null
or
if obj creation is local & method finishes execution.

[then the mem is deallocated]

---

9 Encapsulation & Abstraction

a) Encapsulation → Restricting certain parts of code from directly
(data Hiding). accessing sensitive data.
& access restriction

2 ways to be implem. together. → Using private (access modifier).
→ Using Getter & Setter.

b) Eg of same is given in tryouts.

c) Abstraction → In ATM, only UI & money imp for user, inside (or) Internal code (or) working is not.

Hence internal working is abstracted from user.

```
account.withdraw(2000);  }  Internal working of
account.balance();       }  method withdraw not
                            shown
```

d) Access Modifiers.

→ Used to facilitate encapsulation.

→ Public, Private

Protected → Inside same package and to subclasses in different packages.

Default → If no access modifier specified.
(inside same package.)