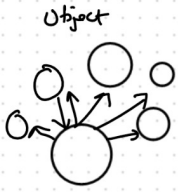
O.O.P

4 pillars

1. Encapsulation ✓
2. Abstraction ✓
3. Inheritance
4. Polymorphism

} better programs ⇑

pillars on paradigm of O.O.P

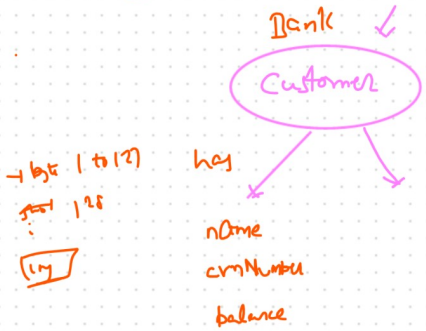


⟹ Communicate well each other ⇓ build better applications

Encapsulation: The process of binding the data & code together

We can achieve

1. code reusability ✓
2. abstraction ✓
3. Data Security ✓
4. Maintainability ✓
5. Read/write Protection (Getter & Setter) ✓




```
class Customer{
    String name;
    long crnNumber;
    double balance;
}

public class ProgWithoutEncap {

    public static void main(String[] args) {
        Customer x = new Customer();
        x.name = "Arvind";
        x.crnNumber = 75929271;
        x.balance = 500;
        System.out.printf("Name : %s, CRNNumber :
    }
}
```

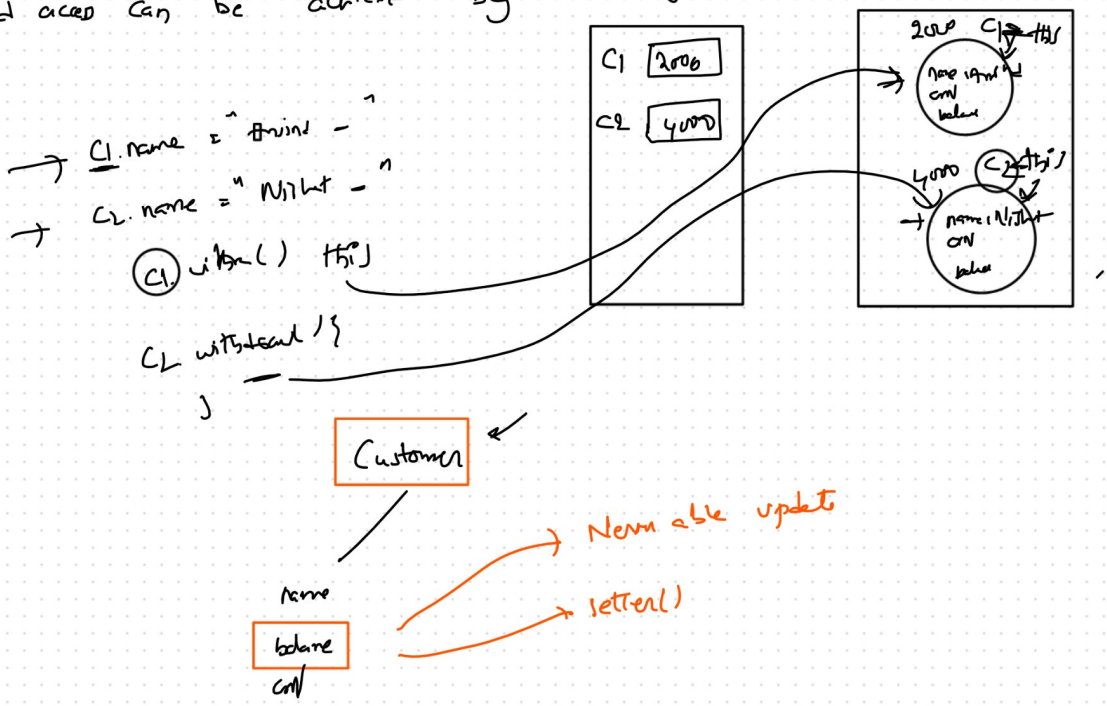
Notes:

1. In the above program, we are able to access the field directly, we can insert any values, which results in data security
2. In order to achieve data security, by the help of private keyword (General syntax)
   → `private String name;`
   ⇑ make field private
3. Private is a modifier in Java, which can be used for
   1. → variables
      → methods
      → inner class [with in file]
4. If we make any thing as private it can be only accessed with in the class (or) by the class

⑤ The process of making the fields as private & not giving direct access to the fields is called as "data hiding"

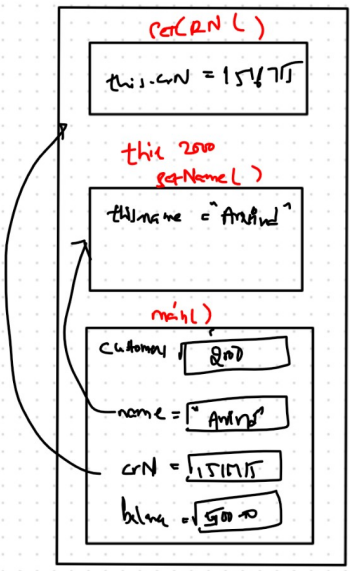
⑥ Since direct access is not available, we have to give some "controlled access" to the fields

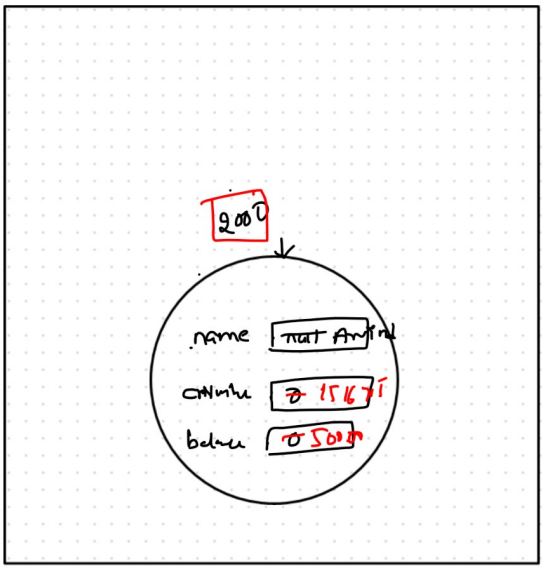
⑦ Controlled access can be achieved by the getter() and setter()



Memory Mapping

```
public static void main(String[] args) {
    Customer2 customer1 = new Customer2();
    customer1.setName("Arvind");
    customer1.setCRN(146612312);
    customer1.setBalance(500.00);
    customer1.setBalance(500000.00);
    System.out.printf("Name : %s, CRNNumber : %d
    customer1.getName(),
    customer1.getCRN(),
    customer1.getBalance());
}
```





Notes (importance of this keyword)

1) "this" will be applied to instance variable only not to local variables

2) The power of having the same name for the local variable & instance variable will arise a problem which is called "shadowing"

3) We can overcome this problem by using the keyword "this"

4) In 2nd program in setName(), the values for the fields are not assigned because the local variable & field's have the same name, so priority will be given to "local" variable by JVM instance var1 ⟹ fields

5) When setting balance if(amount > 0) is nothing but your abstraction, which is hiding the logic & functionality meant for updating balance

6) Maintenance is also done in version 2 when (amount > 0) condition applied

Ver1

5000 ← 1000

4000 ← -1000

balance + 100

Ver2

3000 ← -1000

throw error