OOP-4

1- Deep copy vs Shallow copy / Copy Const

2 - OOP Hands-on | CoffeeMachine | Designing class

3 - MISC - super, final, friend (will attach references)

4 - Wrap up + Recap

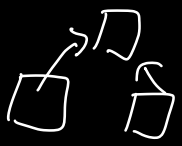
5 - Doubts and Further

Till now –

I
- Object, class, constructor, destructor ←
- Static, this
- Encapsulation, Abstraction

II
- Inheritance, constructor calls
  - modes, types
  - Diamond problem ←
- Abstract classes & interfaces

C++ multiple inheritance



III
- Overloading - Fn, operator ←
- Polymorphism
  - static → overloading.
  - Dynamic → overriding
- Virtual Functions

Misc
- Deep vs Shallow copy ⟸
- Problem discussion + questions

IV. Deep vs Shallow Copy

```cpp
class Bus {
private:
  int yearBuilt_;
  string name_;
  vector<string>* routeList_;

public:
  Bus(int yearBuilt, const string& name, vector<string>& routeList)
    : yearBuilt_(yearBuilt), name_(name), routeList_(&routeList) {}

  void setName(const string& name) { name_ = name; }
  void addStop(const string& stopName) { routeList_->push_back(stopName); }
  void removeStop(const string& stopName) {
    int id = -1;
    // find the id to remove
    for (int i = 0; i < routeList_->size(); ++i) {…
    if (id != -1) { routeList_->erase(routeList_->begin() + id); }
  }
  void busDetails() {
    cout << "The " << name_ << " bus was built in " << yearBuilt_ << " and its current\n";
    cout << "route is: ";
    for (int i = 0; i < routeList_->size(); ++i) {
      cout << (*routeList_)[i] << " ";
    }
    cout << endl;
  };
};
```

pointer to vector

Initializer list.

Copy constructor

```
int x = 4;
int y = x;
```

```cpp
int main() {
  vector<string>* route = new vector<string>();

  // Add one bus
  Bus local(2015, "LOCAL", *route);
  local.addStop("Main St.");
  local.addStop("First Ave.");
  local.addStop("Second Ave.");

  // Add another bus that skips First Ave.
  Bus express = local;
  express.setName("EXPRESS");
  express.removeStop("First Ave.");

  // Let's see what we've got
  local.busDetails();
  express.busDetails();
  return 0;
}
```

same ptr.

Objects will be at new locations in memory.

Copy constructor default

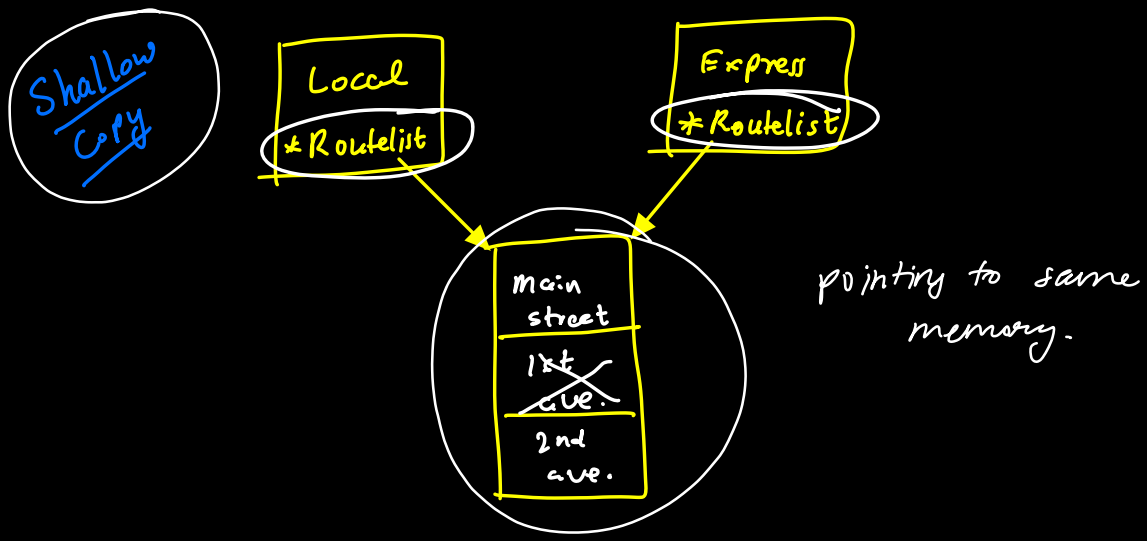
Deep ! Shallow

Bus(name): name_(name)

name_ = name

N: New
U: Update

I — Copy Constructor : constructing new object



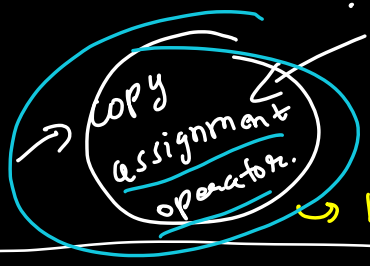
(Q) But, we didn't even define a copy constructor!

⇒ Default copy constructor

Bus Express = local; → copy constructor.

Bus Express; → New. Does this work?

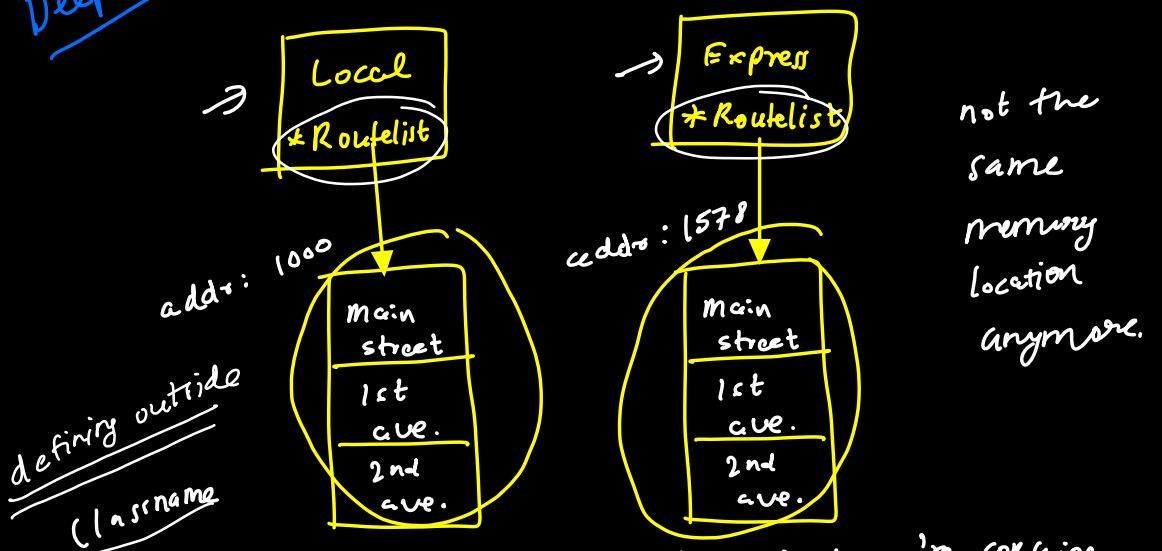
express = local; → Update



X copy constructor

int x = 3;
int y = 4;
y = x;
F

Deep Copy




```
45  Bus::Bus(const Bus& src)
46      : yearBuilt_(src.yearBuilt_),
47        name_(src.name_),
48        routeList_(new vector<string>(*src.routeList_)) {}
49
```

deep copy. (src->routelist_)

routelist_(src. routelist_) → Shallow

Q) What about copy assignment?

→ overload

Q) Should copy constructor always create deep copies?

NO

maybe situations where we need only shallow/ mix of them!

→ Store bus owners details (name) string* name;

→ subset. same owner

After deep copy constructor implemented

I. `Bus express = local;` → Deep copy

II. `Bus express;`

`express = local;` ←

⇓

not deep copy

overload the copy assignment operator as well!

=> Coffee Machine

multiple solutions

Object Oriented Design
- requirements
use cases

. Design a coffee machine which makes diff beverages

Requirements based on set of ingredients.

```
# Coffee Machine

- Design a coffee machine which makes different beverages based on set of ingredients.

- The initialization of the recipes for each drink should be hard-coded, although it should be relatively easy to
add new drinks.
- The machine should display the ingredient stock (+cost) and menu upon startup, and after every piece of valid user
input.

- Drink cost is determined by the combination of ingredients.
  - For example, Coffee is 3 units of coffee (75 cents per), 1 unit of sugar (25 cents per), 1 unit of cream (25
  cents per).

- Ingredients and Menu items should be printed in alphabetical order. If the drink is out of stock, it should print
accordingly.
- If the drink is in stock, it should print "Dispensing: ".

- To select a drink, the user should input a relevant number.
- If they submit "r" or "R" the ingredients should restock, and "q" or "Q" should quit.
- Blank lines should be ignored, and invalid input should print an invalid input message.
```

drink. → normal → black.

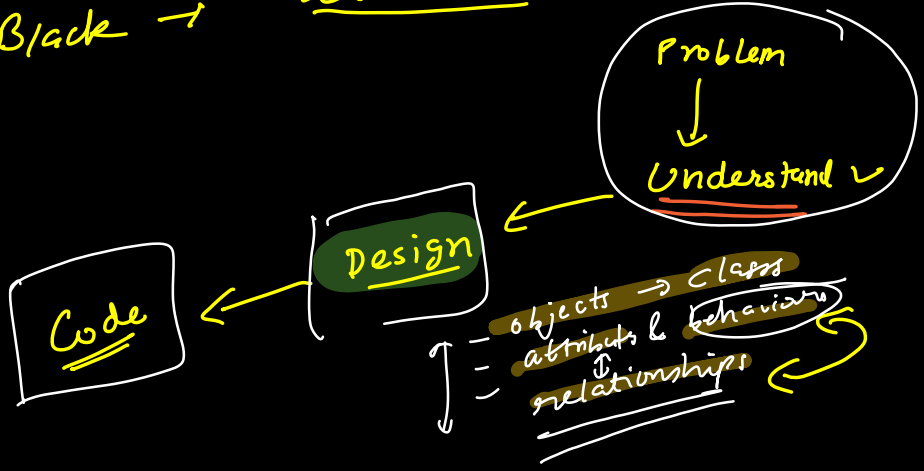
easy to add new drinks.

and ingredients.

full specs

→ Normal → Sugar, Milk, coffee → 10 Rs.

→ Black → water, coffee. → 10 Rs



Coffee Machine

- Design a coffee machine which makes different beverages based on set of ingredients.

- The initialization of the recipes for each drink should be hard-coded, although it should be relatively easy to add new drinks.

- The machine should display the ingredient stock (+cost) and menu upon startup, and after every piece of valid user input.

- Drink cost is determined by the combination of ingredients.
  - For example, Coffee is 3 units of coffee (75 cents per), 1 unit of sugar (25 cents per), 1 unit of cream (25 cents per).

- Ingredients and Menu items should be printed in alphabetical order. If the drink is out of stock, it should print accordingly.
- If the drink is in stock, it should print "Dispensing.".

- To select a drink, the user should input a relevant number.
- If they submit "r" or "R" the ingredients should restock, and "q" or "Q" should quit.
- Blank lines should be ignored, and invalid input should print an invalid input message.

n-m

abstraction - use case

qty. of ing.

adding

Step-1 Identify the objects → classes.

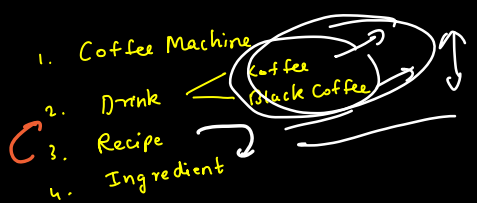
→ the before name

noun can be

Possibilities

1. Coffee Machine
2. Drink ← Coffee, Black Coffee
3. Recipe → Collection of ingredients.
4. Ingredient

Decision



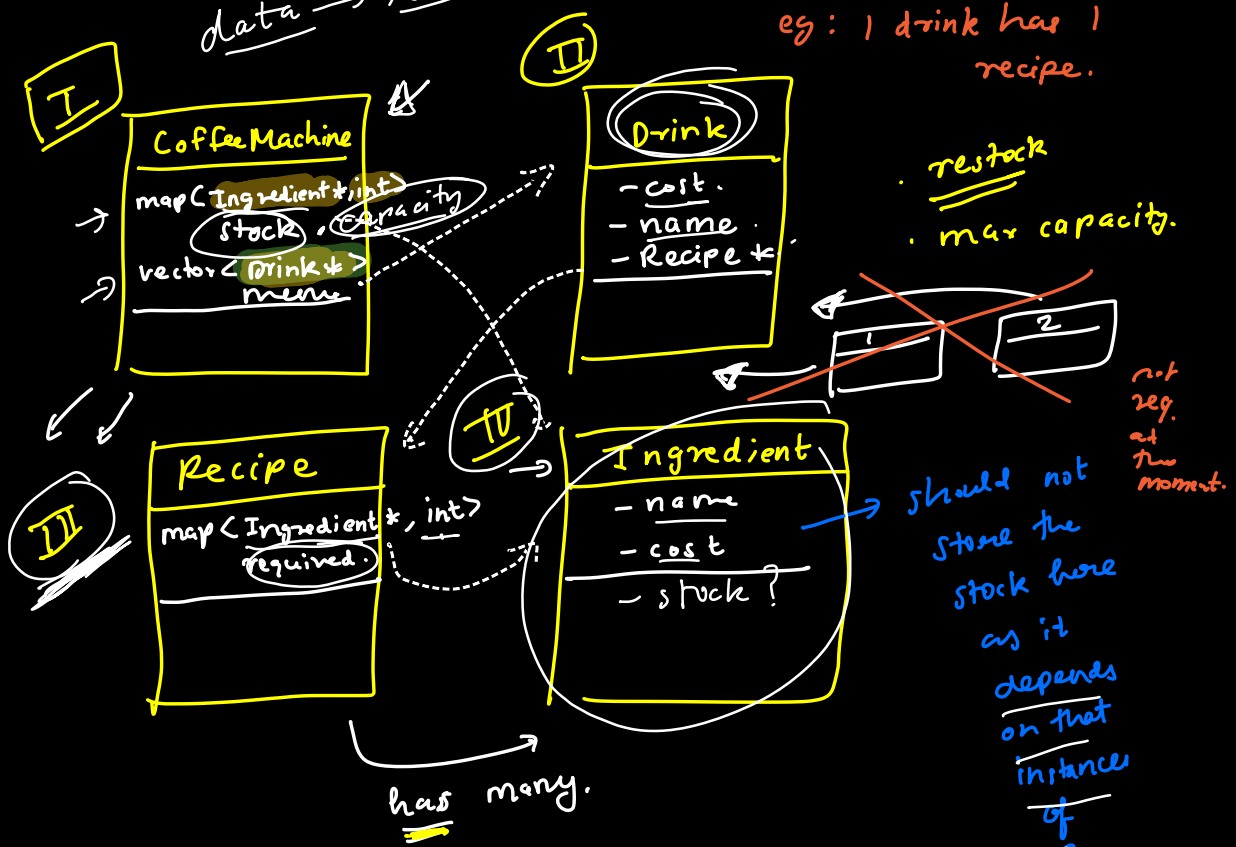
Step 2:

Figure out the attributes and relationships.

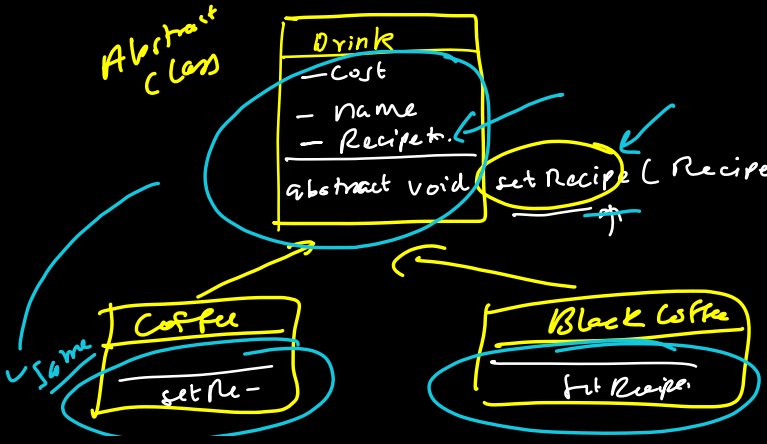
data

data → relationships

eg: 1 drink has 1 recipe.

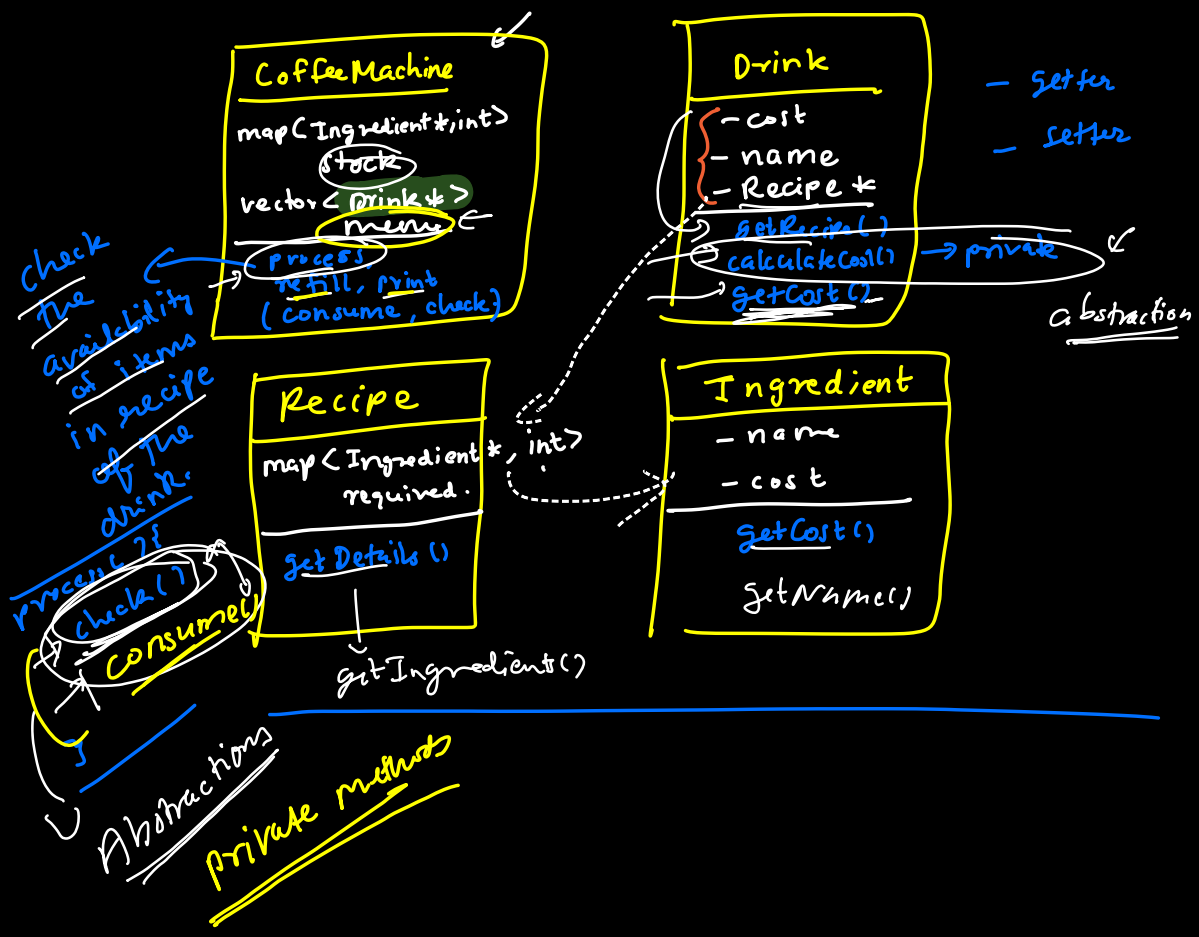


should not store the stock here as it depends on that instance of Coffee Machine.



does inheritance provide any additional benefits in this case?

Step 3: Figure out the behaviors/methods for the classes.



→ Hands-on Live Coding

→ Doubts

→ Announcement

* OS sessions start from Mon 8 pm.

M, W, F, Sunday