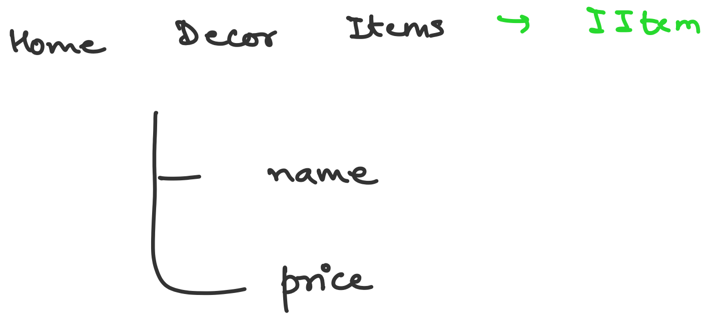
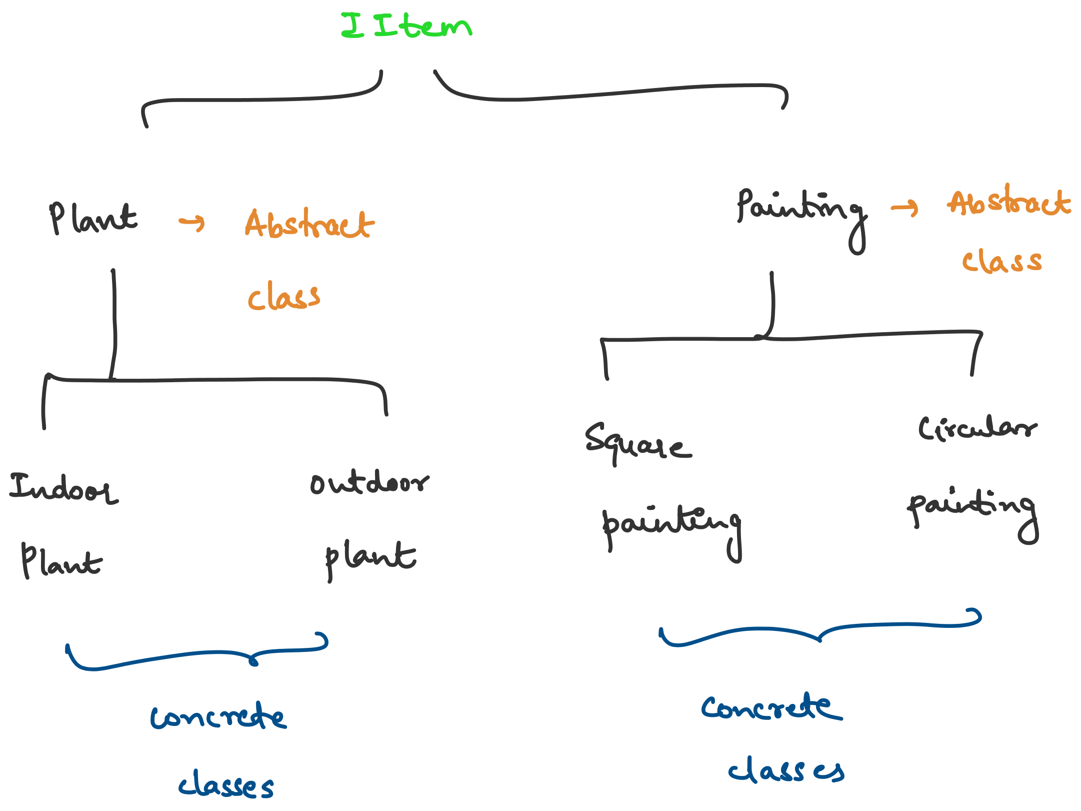
15__28-08-2022

Sunday, 28 August 2022 8:07 PM

Builder design pattern





Create order class

↓

will be used to place order

```
public class Order
{
    ==

    public static class Order Builder
    {
        ==
        ——→ create combos (optional step)

    }       1. Prepare indoor combo
                 ├─ indoor plant
                 └─ Square painting
}
            2. Prepare outdoor combo
                 ├─ outdoor plant
                 └─ Circular painting
```

Steps for implementation –

1. Create an interface.

$$\begin{tikzcd}
 & \text{name()} \\
\text{IItem} \arrow[ur, dash] \arrow[dr, dash] & \\
 & \text{price()}
\end{tikzcd}$$

2. Create abstract classes implementing the item interface providing default functionalities.

```
abstract Plant implements IItem
        {
              provide abstract method for price
        }
```

```
abstract Painting implements IItem
    {
          provide abstract method for price


    }
```

3. Create concrete classes extending plant and painting abstract classes.

```
class Indoor plant extends Plant

{
            provide price

            provide name

}


public class Outdoor Plant extends Plant

{

        provide price and name

}


public class square Painting extends Painting

{

            provide name and price

}
```

```
public class Circular Painting extends Painting
{
    provide name and price
}
```

5. Create order class
   ↳ This class will be using the items defined above.
   (Like in previous example inside car class we used ac, brake, etc)

```
public class Order
{
    Build list of items
    Provide API for adding items
    Provide API for getting cost
```

Provide API [illegible] [illegible] [illegible]

Provide API to show the items

Define order builder class

{

prepare indoor gift combo

prepare outdoor gift combo

}

}

6. Create main method to build the order.

Prototype design pattern

↓

means copy | clone

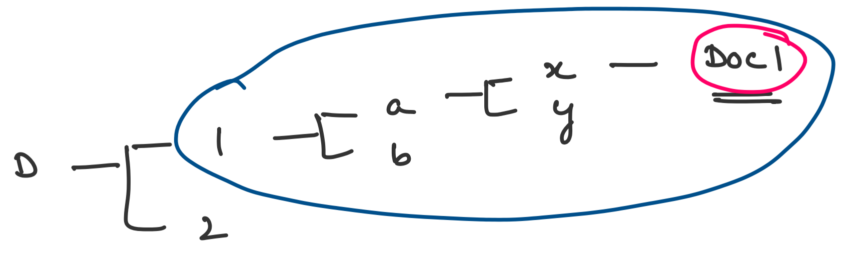
Object → Very heavy object

↓

Instead of creating the same object from scratch, go and copy it.

↓

Helps us in saving our time and resources.



According to Gang of four –

Prototype design pattern specify the kind of objects to create using a prototypical instance and create new objects by copying this prototype.

To simplify, instead of creating objects from the scratch every time, you can make copies of an original instance and modify it as required.

creational design patterns –

↳ singleton pattern – 1 object

↳ Factory pattern & abstract factory – Many objects & simple objects

↳ Builder design pattern – complex object

Mandatory Attributes

Optional attributes

↳ prototype design pattern — copy of object

Prototype is unique among the other creational patterns as it does not require a class but only an end object.

Implementation guidelines —

we need to choose the prototype design pattern when—

1. Creating an object is an expensive op^n and it would be more efficient to copy an object.

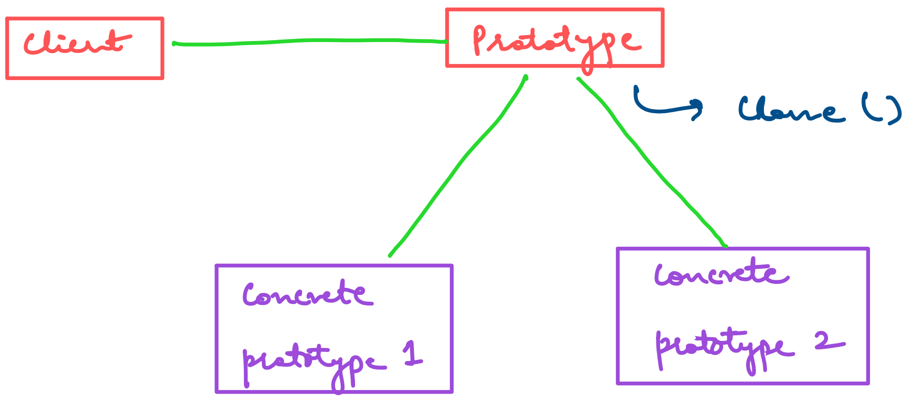
2. we need objects that are similar to

[illegible]

existing objects.

3. We need to hide the complexity of creating the new instances from the client.

4. When we want our systems to be independent of how its products are created, produced and represented.



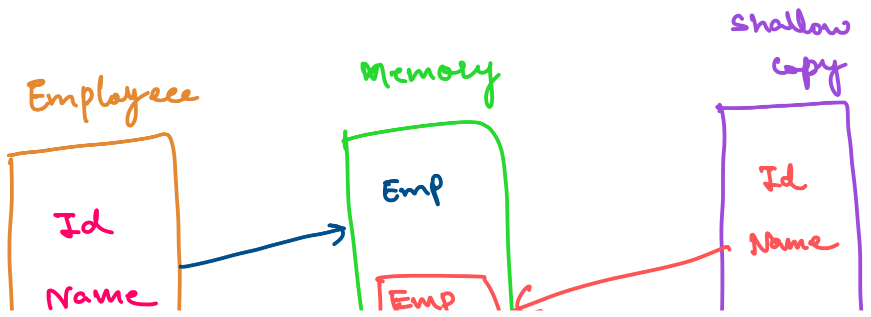
Copying mechanisms

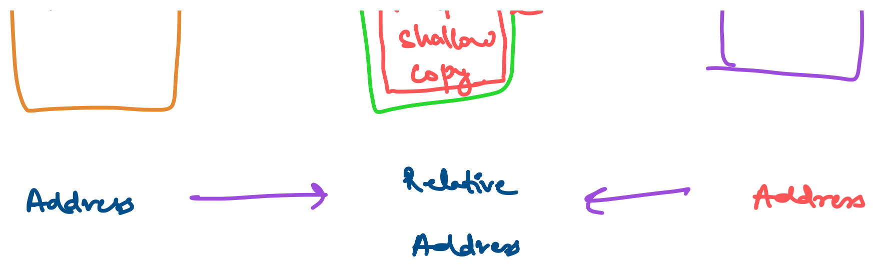
The idea of using copy is to create a new object of the same type without knowing the exact type of the object we are invoking.

Shallow copy

↓

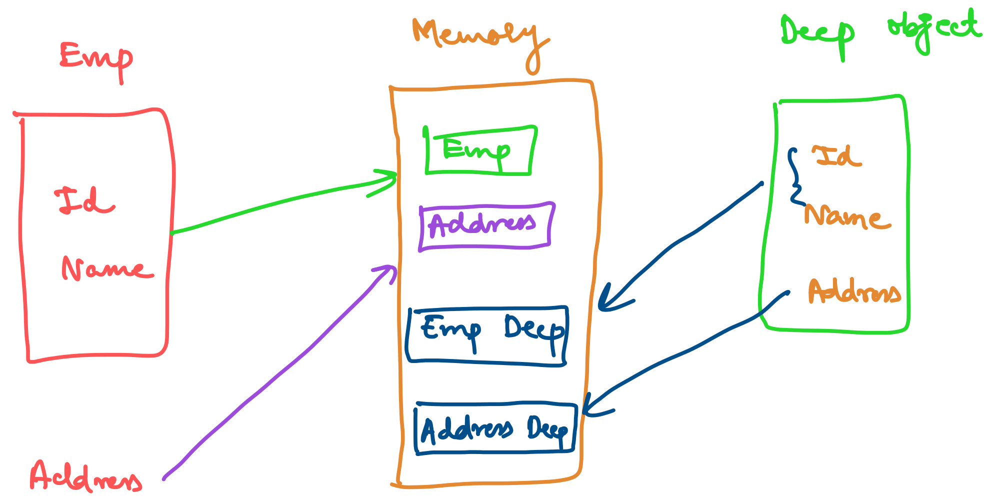
copies an object's value type fields into the target object and the object's reference types are copied as references into the target object.





Deep copy

Deep copy objects copy an object's value and reference types into a complete new copy of the target objects.



Prototype design pattern performs cloning of an existing object instead of creating a new one and this can also be customized as per the requirements.

Advantages of prototype pattern

1. It hides the complexities of creating objects.
2. It lets you add or remove objects at runtime.
3. It reduces the need of sub-classing.
4. The client can get new objects without knowing which type of object it will be.

IClone able interface

[illegible]

It provides us the customized implementation

that creates copy of an existing object.

$$\begin{tikzcd}
\text{ICloneable Inteeface} \arrow[r] & \text{method} \arrow[d] \\
& \text{Clone ( )} \arrow[d] \\
& \text{Provides support for Memberwise Clone method.}
\end{tikzcd}$$