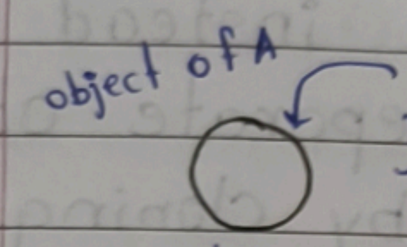
# Lecture : 36 - Prototype Design Pattern

## # Introduction

- Simple and solves specific type of problem
- We have used it before without knowing it is a design pattern.

- Problem solved by prototype




- Now, we have created one object of class A.

- And we need multiple object of same class ~~but they are~~ but there is too much similarity btw these mach created objects.

```
A *a1 = new A();
A *a2 = new A();
```
↳ allocation of memory in heap

- And we know object creations are most expensive task as it may include -
  1. DB connection establish
  2. complex calculation
  3. load big and complex file.

- These problem is fixed by prototyping design pattern
  ↳ Let's see how with an example

# Example : NPC - Non player character

1. Let's take example of NPC, so NPC are non player character in a game which are in background of player doing certain & specific task (ex : GTA person's)

2. So, whenever we start game all these NPC objects need to be created so instead of making all these characters separate objects we can reuse created objects by cloning or inshort using prototype design pattern.

# What is and How does prototype solve problem?

① Create one object of class suppose obj and we need one more object then instead of creating it again we can do is copy of them.

② Inshort, copy the existing object to reduce or save time in complex operation.

③ NPC *n1 = new NPC;
NPC *n2 = n1 ⟶ clone();

| NPC |
| --- |
| //expensive oper<br>clone(){..}; |

④ How does it clone?
↳ We know about copy constructor

Basically there are 3 constructors - default (no parameters), paratmetrized (having parameters) and

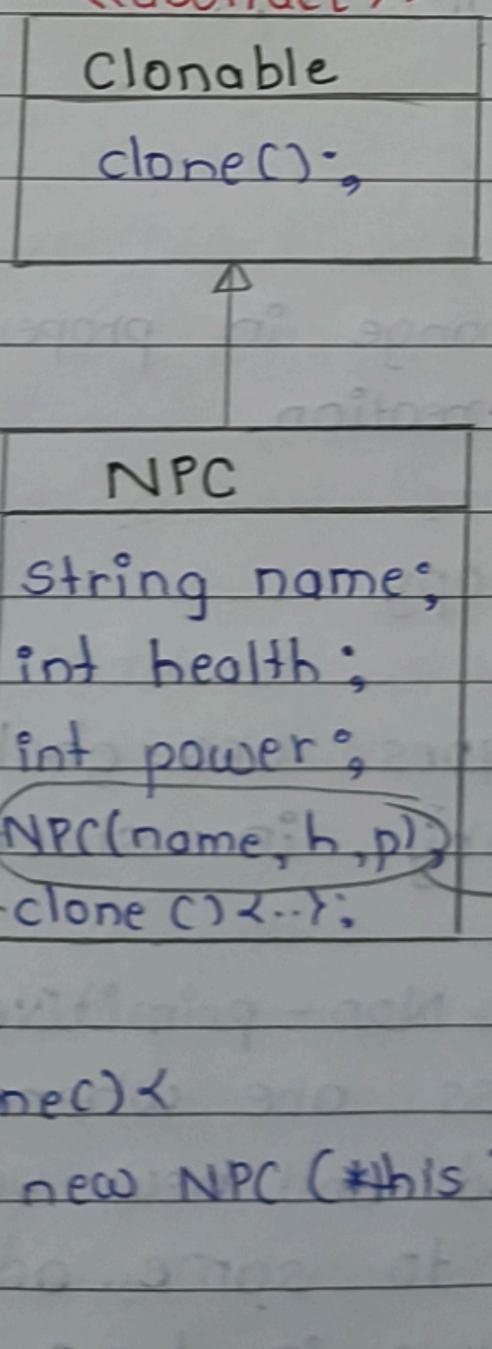
- Prototype Pattern is used only where there are only little changes btw multiple objects

copy constructor

- Copy constructor -
  ↳ Here we write how our ~~of~~ ~~class~~ object should be copied (details in notes)

⑤ Now, we copied all properties of n1 to n2 but we do not need same properties so we can update it using getters and setters of that class (example is in code)

# UML Diagram of NPC example



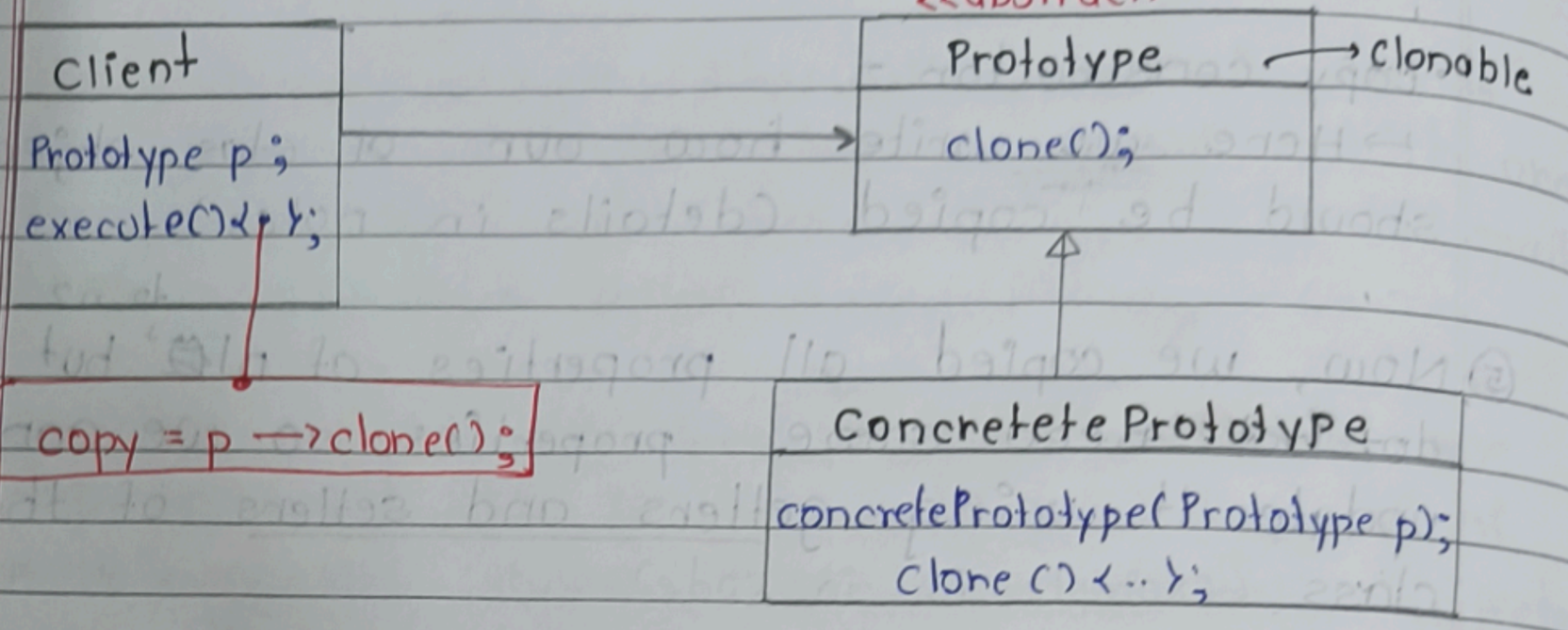

→ Whichever class inherits clonable class, that class/objects ~~are~~ can be cloned.

→ In Java, Clonable abstract class is known as marker interface as it marks the class which are clonable.

→ this has copy constructor which shows how it shld be copied or helping to copy. → reference of self

```
NPC(NPC *n){
    this.name = n.name
    this.health = n.health
    this.power = n.power
}
```

# Standard UML Diagram




# Standard Definition

If let you create new object by copying [cloning] another instance.

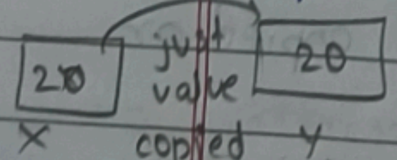
# Real-World Use Case

① Games NPC characters
② Multiple objects with less change in properties
③ Complex / Expensive object creation.

# Shallow Copy Vs Deep Copy

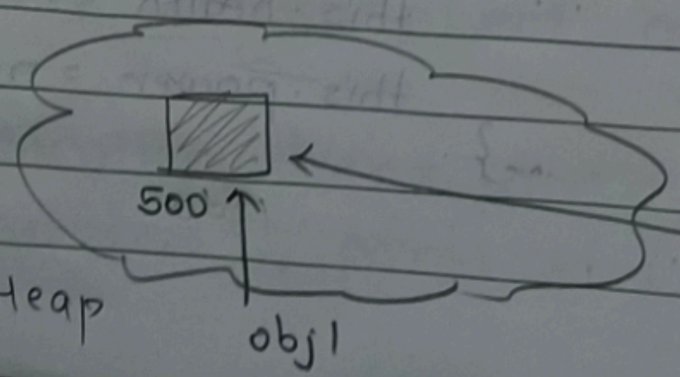
Deep

1. Shallow Copy : Primitive datatypes are shallow copied as it takes memory in stack




Shallow

2. Deep Copy : Non-primitive datatype like class are when copied both old and new variables points to same address as it stores in heap.




obj1 = obj2