Intro to OOPS

LLD1

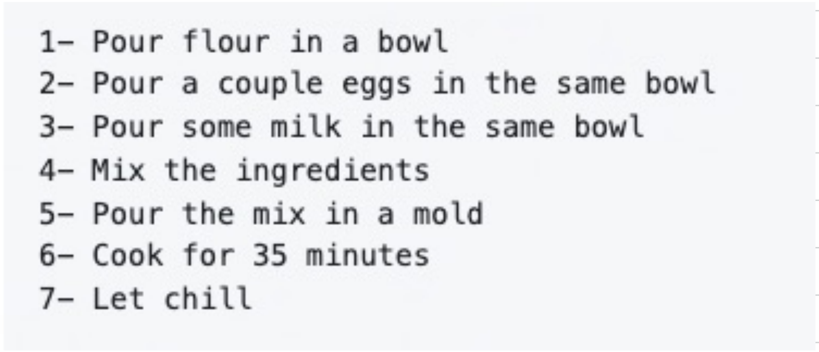
(4 classes
↓
OOPS)

OOPS
- ↳ 5 Programming Paradigms → style of writing code
  → not all language will support all Styles.
- ↳ Principles of OOPS
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
- ↳ Code Examples

1) Imperative (Java)
2) Procedural
3) OOPS (Java) LLD
4) Declarative (Java) LLD1
5) Reactive (Java)
   ↓ certain software applications
   asynchronous events
   Data streams,
   Observables, etc.

1 Imperative Programming

↳ each instruction step by step.



```
1- Pour flour in a bowl
2- Pour a couple eggs in the same bowl
3- Pour some milk in the same bowl
4- Mix the ingredients
5- Pour the mix in a mold
6- Cook for 35 minutes
7- Let chill
```

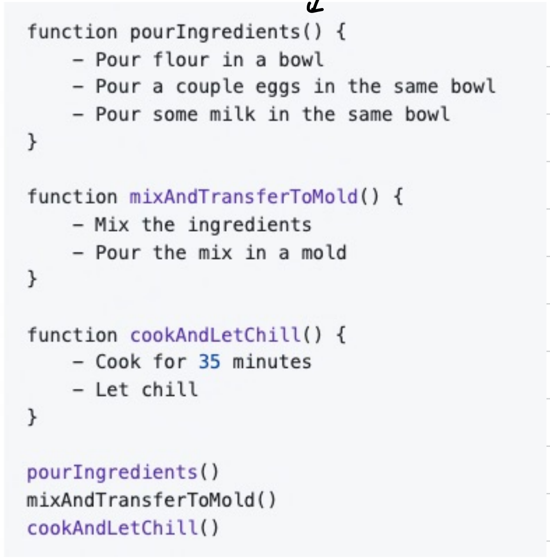
→ c + d

→ d + e

2 Procedural Programming (C, C++)

- create functions for specific tasks.
- reusable.
- readability

add(c,d)
add(d, e)

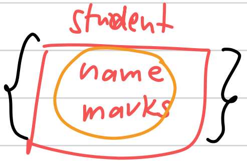


```
function pourIngredients() {
    - Pour flour in a bowl
    - Pour a couple eggs in the same bowl
    - Pour some milk in the same bowl
}

function mixAndTransferToMold() {
    - Mix the ingredients
    - Pour the mix in a mold
}

function cookAndLetChill() {
    - Cook for 35 minutes
    - Let chill
}

pourIngredients()
mixAndTransferToMold()
cookAndLetChill()
```

③ Object Oriented Programming (C++, Java, Python)

Pass a struct to a function.

```
void updateMarks(struct Student){

    Student.marks = 10;

}
```



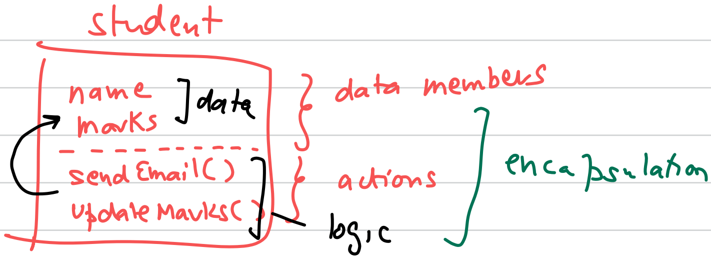
↳ Action is performed on the student

OOPs world

↑

```
Student.name = "Prateek"
Student updateMarks(100);
```

↳ student is performing the action.



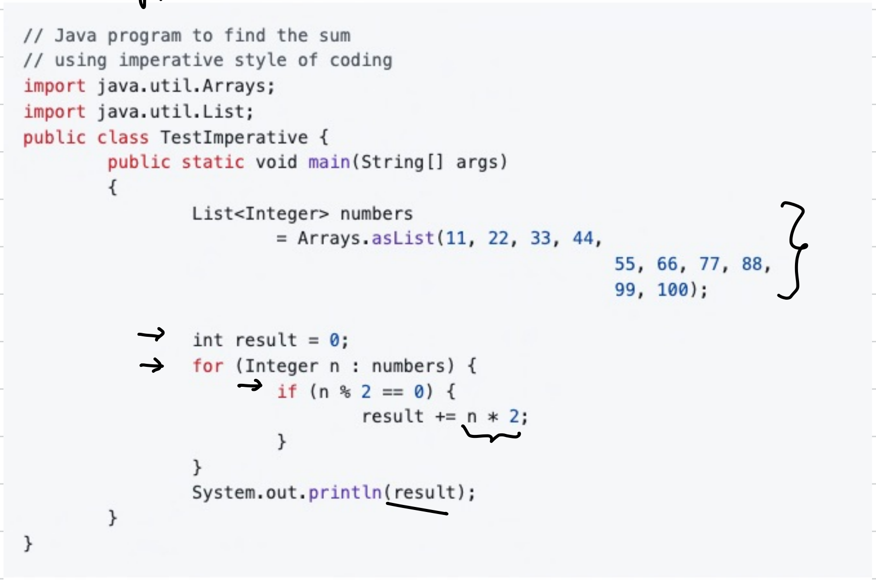
④ "Functional" Programming / Declarative Programming

$$\begin{tikzcd} & \text{Java} \arrow[dl] \arrow[d] \arrow[dr] & \\ \text{OOP} & \text{Declarative Prog} & \text{Reactive Prog.} \end{tikzcd}$$

↳ you "declare" what to do.
but not how to do.

↳ Java 8 - Streams, Lambdas

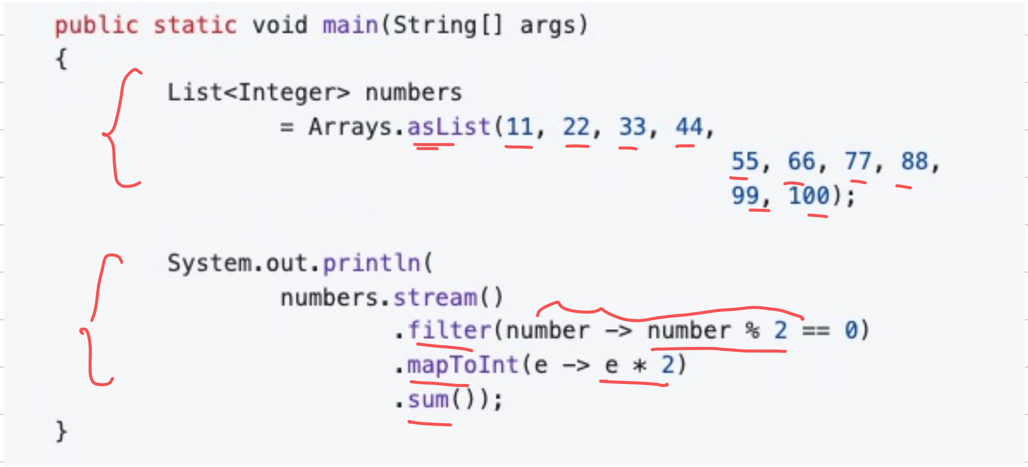
Imperative Style



```java
// Java program to find the sum
// using imperative style of coding
import java.util.Arrays;
import java.util.List;
public class TestImperative {
        public static void main(String[] args)
        {
                List<Integer> numbers
                        = Arrays.asList(11, 22, 33, 44,
                                                        55, 66, 77, 88,
                                                        99, 100);

                int result = 0;
                for (Integer n : numbers) {
                        if (n % 2 == 0) {
                                result += n * 2;
                        }
                }
                System.out.println(result);
        }
}
```

find out the sum of double of even numbers from the list

Declarative Style   Streams + Lambda's (later classes)

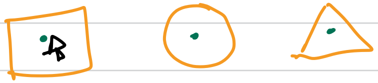


```java
public static void main(String[] args)
{
        List<Integer> numbers
                = Arrays.asList(11, 22, 33, 44,
                                                55, 66, 77, 88,
                                                99, 100);

        System.out.println(
                numbers.stream()
                        .filter(number -> number % 2 == 0)
                        .mapToInt(e -> e * 2)
                        .sum());
}
```

$\boxed{\text{OOPS}}$

Head first Java (Beginners Book.)

Problem Statement Once upon a time in a software shop, two programmers were given the same spec and told to "build it". The Project Manager forced the two coders - to compete. The problem statement is as follows: There will be shapes on GUI, a square, a circle and a triangle. When the user clicks the shape, it will rotate clockwise 360 degrees and play a .mp3 sound corresponding to that shape.



→ Amoeba Shape.
↓
modify rotate method

Procedural world

```
void rotate ( Shape s ){
    // Shape around center Rotating 360 clockwise
    if (s!= Amoeba)
        ≡
    else (
        ≡
    )
}
}
```

ugly.

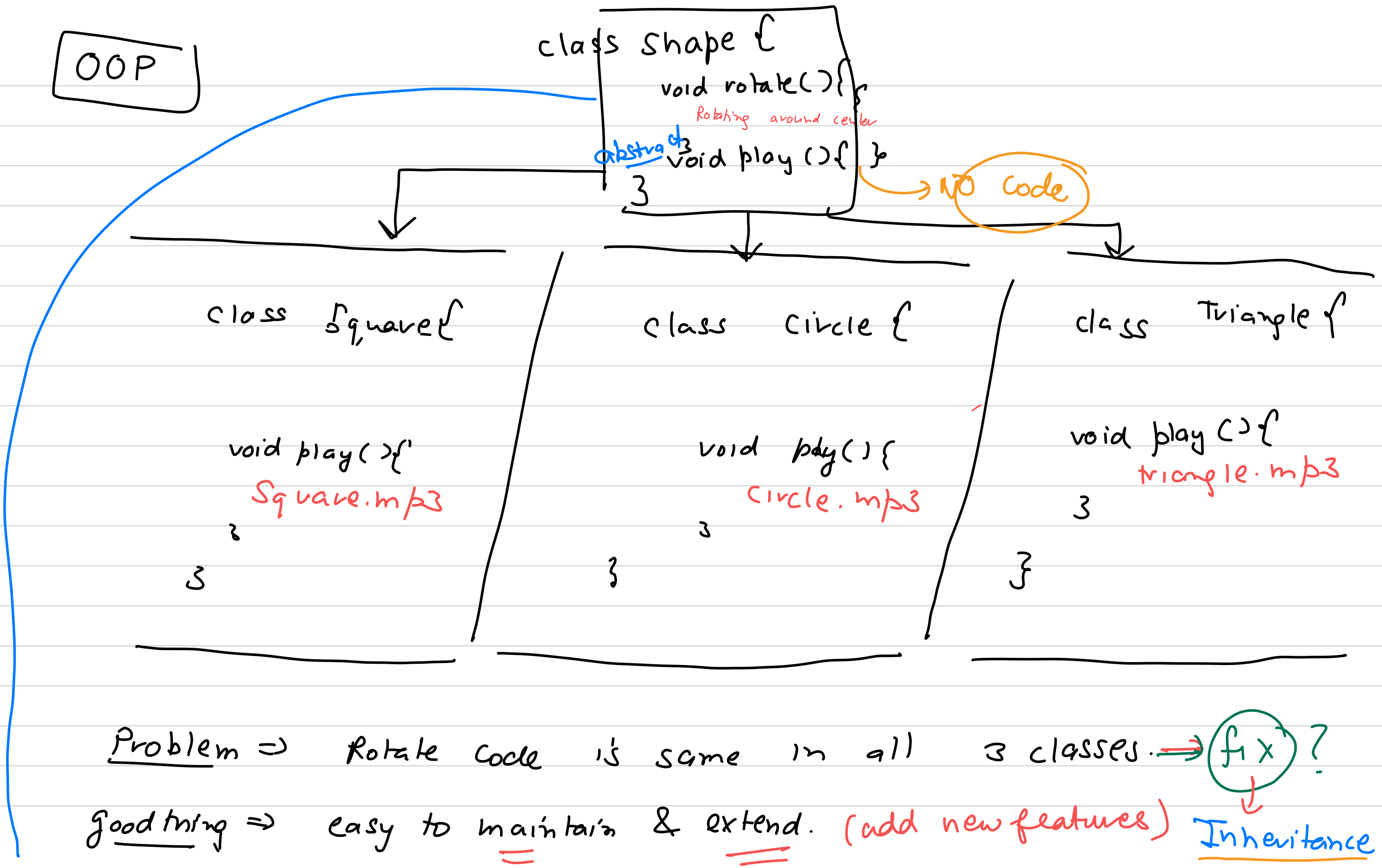
→ modify the existing code

```
void   playsound(Shape s){

    if(s == "Circle")
          play(circle.mp3);

    else if(s == "Triangle")
          play(Triangle.mp3);
    :
    :
    else if(s == "Amoeba");
}
```

→ Modify the already tested code.

which was tested.
↳ Bugs.

OOP



```
class Shape {
    void rotate(){ }
        Rotating around center
    abstract void play(){ }
}
```

```
class Square{

    void play(){
        Square.mp3
    }
}
```

```
class Circle {

    void play(){
        Circle.mp3
    }
}
```

```
class Triangle {

    void play(){
        triangle.mp3
    }
}
```

Problem ⇒ Rotate Code is same in all 3 classes. → fix ? → Inheritance

good thing ⇒ easy to maintain & extend. (add new features)

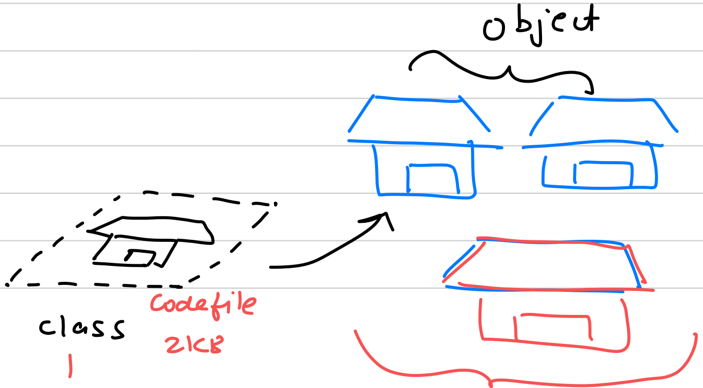
Method over-riding ✓

```
class Amoeba {
    void rotate() {
        logic
    }
    void play() {
    }
}
```

← we don't have to modify / test previously written code.

Classes & Objects.

↓ template to create object



↓ piece of code that Compiler uses to create an

↓ RAM

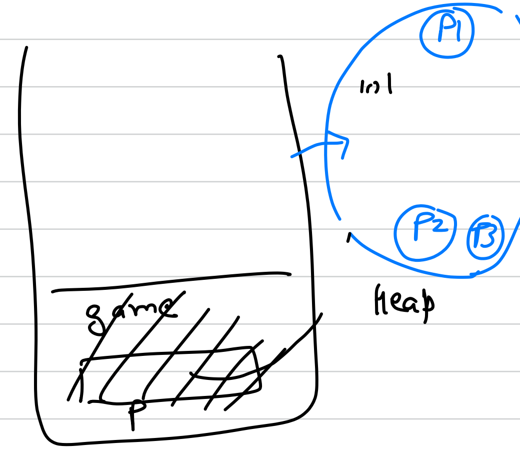
```
Class Player {

    String name,
        int age;
        int number,

    void  guess Number (){
          number = rand(),
    }

}
```

Data Members

→ Methods that operate on these data inside one class

Primitives & objects →

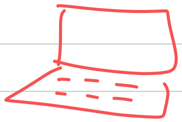
```
Create Players(int n) {
        Array arr;
        for( i=0; i< n, i++) {
            arr add(new Player());
        }
        return arr;
}

game () {

Players[] p = Create Players(3);

}
```



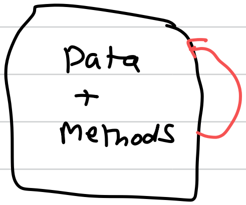
Pillars of Object Oriented Programming

① Abstraction

↳ hiding the complexities of implementing.



② Encapsulation



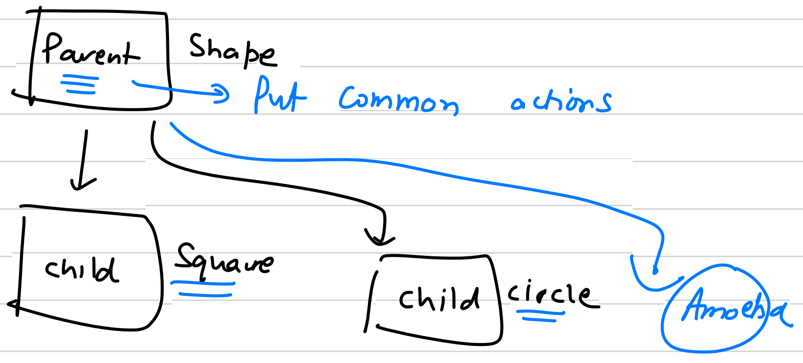
also ensure state of data is maintained correctly / impose certain constraint

10:35

③ Inheritance

↳ re-usability of code.

↳ maintainble & extensible



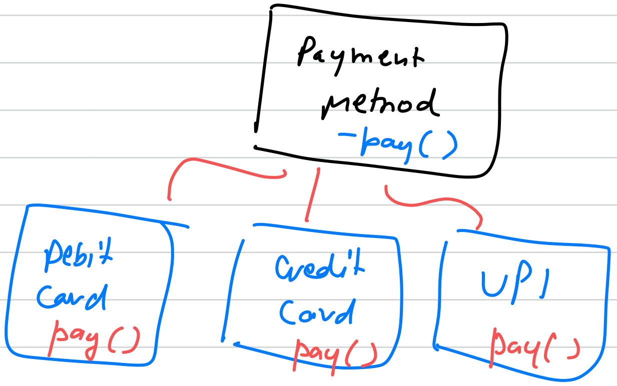
↳ used a lot in building software apps

④ Polymorphism

↳ is the ability of the object to take multiple forms.

```
make Payment ( Payment method  pm ){

    pm.pay(500);

}
```

```
makePayment( dc )
     "     ( cc )
     "     ( upi )
```



of OOPS

Advantanges:

✓

1. Re-usability : inheritance
2. Security - hiding details through encapsulation.
3. Maintain
4. Inheritance - easily Import existing functionality to create Something new

$$\begin{tikzcd} \boxed{\text{Buttons}} \arrow[d] \\ \boxed{\text{New Button}} \end{tikzcd}$$

Disadv :-

1) Think about entities before writing code.

2) Typically OOPS code can get larger.