Lecture 12

Saturday, 15 February 2020 2:33 PM

OOPS. → object oriented programming.
↳ why OOPS?.

① Movie →
- duration
- director
- name
- language

② Hall
- location
- No of Audi
- Cafeteria.

③ Audi
- time slot
- Movie
- Seats.

④ time slot
- start time
- end time

Access Specifiers -> public, private, default, protected

classes and objects.

```
class Person {
    private String name;
    private int age;
}
```

```
public String getName(){
    return this.name;
}
public void setName(String myname){
    if(myname.equals("")){ return; }
    this.name = myname
}
```

getter & setters

↳ business checks.

```
default Person(){ }
Person(String name, int age){
    this.name = name;
    this.age = age;
}
```

```
public static void main(String[] args){
    Person p1 = new Person();
    Person p2 = new Person();
    Syso(p1.age); // 0 ×
    Syso(p1.name); // null ×
    Syso(p2.name); // null ×
    Syso(p2.age); // 0. ×
    p1.age = 20
    p1.name = "A".
    p1.setName("B"); //
    Syso(p1.getName()); // B'
    Person p3 = new Person("C", 20);
}
```

Constructor → function having same name as class name but it doesn't have any return type. even not void. to initialise instance variables.


```
class 1 {
}
```

```
class 2 { → object
    void main(){
        Class1 obj = new Class1();
    }
}
```

```
class Student {
    private String name;
    private int rollno;
    public static final String uniform_Color = "Blue";
    public static void getU_(){
        return uniform_Color;
    }
}
```

things which belongs to class

this ×

```
static {
}
```


```
Syso(St_.);
Student st = new Student();
Student s2 = s1;
```


```
Student.UNIT_COLOR = "GrRey" ×
Syso(s1.uniform_Color);
Syso(Student.uniform_Color);
St.uniform_Color = "Blue";
Syso(s2.unifom_);
```

Static can access static.

```
Syso(Student.getUnifo());
```
↳ object.

```
class Hello {
    public static void main(String[] args){
    }
}
```

object ×.

Hello.main

Hello.java

Hello.

```
public void Hello(){
    Syso("name");
}
```
→ object.

1. Encapsulation → [→ classes instance variable, → functions]
2. Abstraction
3. inheritence
4. Polymorphism.

(3, 4: ×)

(Abstraction)

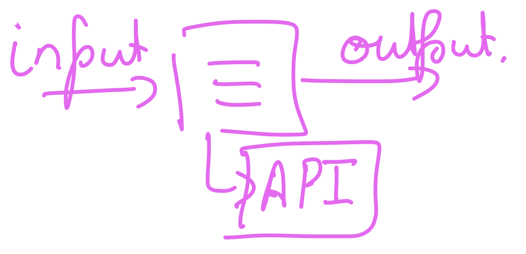
- Data hiding (Access Specifiers).
- (hiding the inner implementation details)

```
public class Car {

    public void MoveCar(String dir){
        if(dir=="left"){ MoveLeft() };
        elseif(dir=="right"){ MoveRight(); }
        else {
            MoveStraight();
        }

    private void MoveLeft(){
        =
    }
    private void MoveRight(){
        =
    }
    private void MoveStraight(){
        =
    }
}
```

```
Car c = new Car()
c.moveCar("left");
```



Data Structures. → why?

Implement:

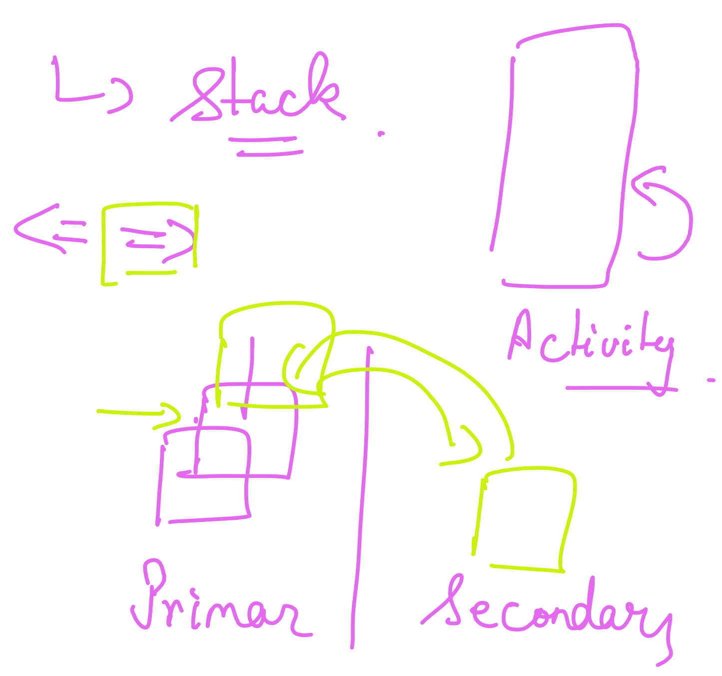
1. Stack
2. Queue
3. L.L
4. Trees
5. Hashtables
6. Heaps
7. tries
8. Graphs

(1–3: Linear; 4: Non-linea.)

Functions.

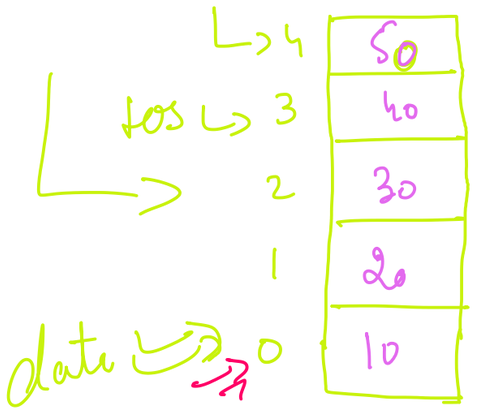
1. add
2. retrieve
3. remove
4. Size
5. isEmpty.
6. display.

→ Stack.



Q: stack.

class Stack{



stack.push(60).

LIFO

(tos == -1)

```
private int[] data;
private int tos;
Stack(){
    data = new int[5];
    tos = -1;
}
```

```
public void push(int item){
    if(this.size() == this.data.length){
        this.tos++;
        this.data[this.tos] = item;
    }
```

```
public int size(){
    return this.tos+1;
}
```

```
public int top(){
    int rv = ____
    return rv;
}
```

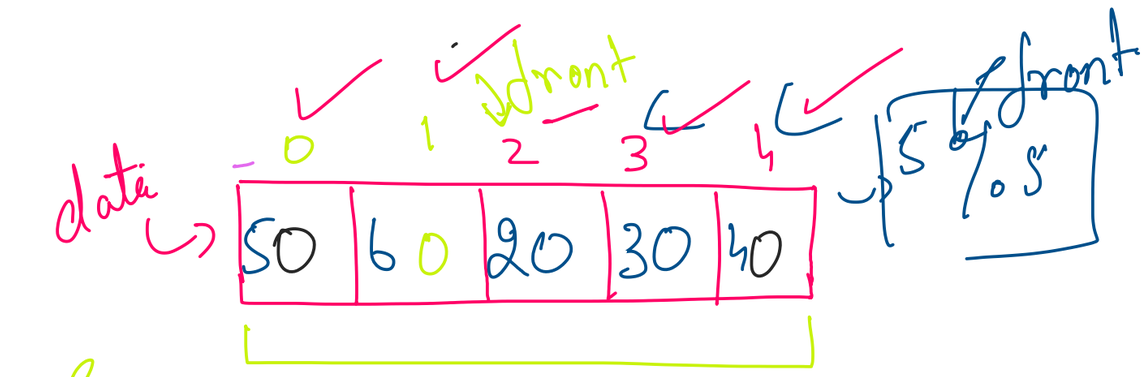
```
public int pop(){
    if(this.size == 0){ ____
    int rv = this.data[this.tos];
    data[tos] = 0;
    this.tos--;
    return rv;
}
```

```
isEmpty(){
    (size == 0).
}
```

(0+1)%5 ai (2+0)%5 => 2

20=> i=1<5 (2+1)%5 => 3

```
display(){
    for(int i = tos; i >= 0; i--){
        Syso(this.data[i]);
    }
```




```
class Queue{
    private int[] data;
    private int size;
    private int front;
    Queue(){
        data.
        size = 0
        front = 0;
    }
    isEmpty(){
        size() == 0;
    }
    void Enqueue(int item){
        if(  ){
            this.data[this.size
            this.sizeFt;
        }
```

size = 0

queue.enqueue(10);

size = 1

qu.en (20);

size = 2

—.enqueue(30);

—.—(40);

—.—(50);

—.enqueue(60). size = 0

—.dequeue(3); front = 0;

—.dequeue(); Size(){

size = 3

—.enqueue(60)

size = 4

—.enqueue(70)

size = 5

—.—(80).

size = 15

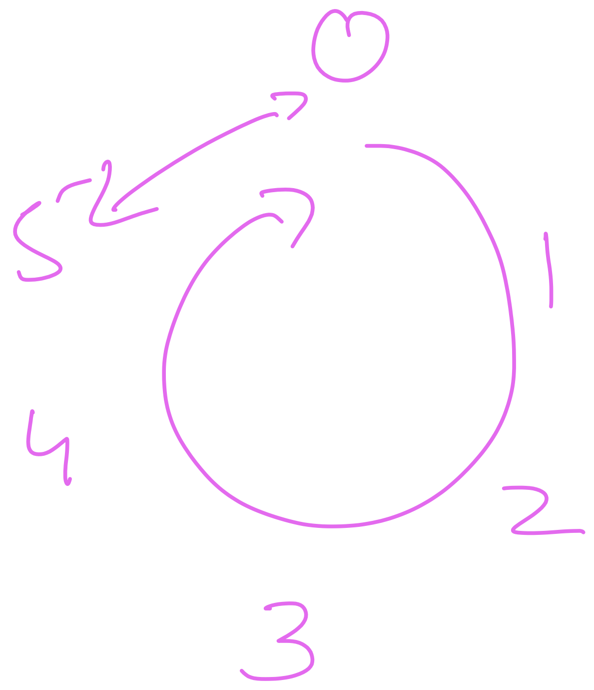
. dequeue()

____ ( )

size = 3

```
int dequeue() {
    rv = this.data[this
    this.data[this.f
    this.size--;
    this.front = (this.front + 1) %
    return rv;
```

```
front() {
    =
}
```



20 => 30 => 40 => 50 => 60

i = 0

```
display() {
    for (int i = 0; i < this.size
        int ai = (this.fro
    }
}
```

where? → processes
→ threads

p1 p2 p3