- Data Abstraction ← Modules
- Encapsulation/Hiding
- OOP

10/5

Data Abstraction & Structures

Data Type

- Name
- set of values
- set of operations

e.g. int x;

$-2^{31} .. 2^{31}-1$

ops: +, -, *, /, %

← representation is abstracted + hidden.

User-Defined Types

Shapes

Circle
radius
area()

Rectangle
width
height
area()

Data Abstraction

↓

User-Defined Types

↓

Abstract Data Types (ADTs)

requires encapsulation/hiding

↓

Objects

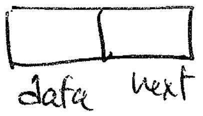
↓

OOP

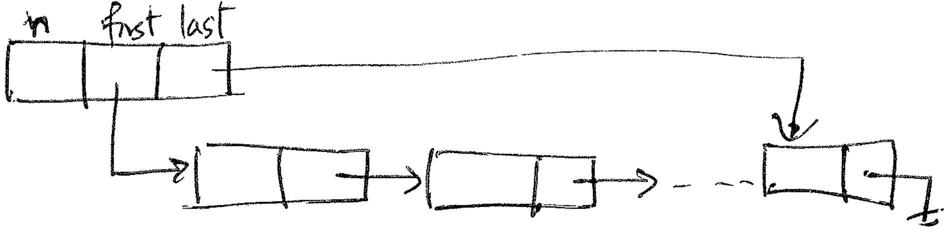
Data Structures

Linked List

Node



Linked List



operations

- insertAtFront
- insertAtEnd
- insert(i,d)
- length()
- empty()

- - -

Data Abstraction in C

circle.h

```
typedef struct {
      float radius;
 } circle;
 float areaOfCircle(c circle);
```

circle.c

```
#include "Circle.h"
#define PI 3.14159
float areaOfCircle(circle c){
    return PI*c.radius
            *c.radius;
 }
```

```
#include "circle.h"

circle c1 = {4.0};
float ac = areaOfCircle(c1);
```

rect

rect.h

```
typedef struct {
    float width;
    float height;
} rect;
float areaOfRect(rect r);
```

C provides good modularization

But does not provide a means for encapsulation/hiding

thus, we can legally write:

```
float ac = 3.1415 * c1.radius * c1.radius;
```

Lists in C

list.h

```
typedef struct {
    int data;        <-
    node* next;
} node;
typedef struct {
    int n;           <- int list!
    node* first;
    node* last;
} LinkedList;
```

int list! Not generic. Could make it generic by:

[illegible]

void* data;

ADT: Encapsulation/Information hiding enforced by language

e.g. MODULA-2

Interface

```
DEFINITION MODULE circle;
    TYPE Circle;
    PROCEDURE areaOfcircle(radius : FLOAT) : FLOAT;
END circle;
```

Implementation

```
IMPLEMENTATION MODULE circle;
    TYPE Circle = RECORD
                radius: ~~INTEGER~~;   (* FLOAT*)
            END;
    PROCEDURE areaOfCircle(radius: FLOAT): FLOAT;
    BEGIN

    END;
END Circle;
```

Use

```
  IMPORT circle;
or  FROM circle IMPORT Circle, areaOfCircle;
```

Java : ADT using classes

- Programmer decides which parts are public/private
- PL enforces visibility

Circle.java

```
public class Circle {
    private float radius;

    public Circle(float r){
        radius = r;
    }
    public area(){
        =
    } //area()
    public string tostring() {
        =
    } //tostring()
} //class Circle
```

terms
- Method dispatch

Use

```
public class MyProg {
    public static void main(___){
        Circle c1 = new Circle(4.0);
        float ac = c1.area();

Cannot do  c1.radius = 8.0;
        system.out.println(c1);
```

similarly

```
Rect r1 = new Rect(3.0,4.0);
~~float ar = area~~
float ar = r1.area();
```

polymorphism

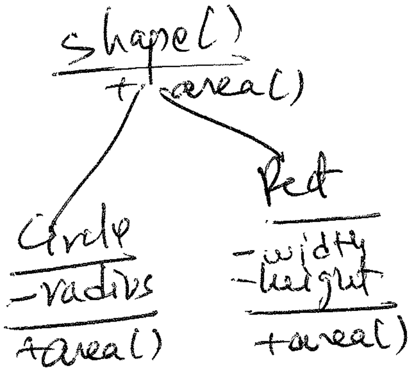
Object Oriented Programming (OOP)

= Objects + Encapsulation + Inheritance

```
public abstract class Shape {
    public float area();
} // shape

public class Circle ~~{~~ extends Shape {
    =
} // Circle

public class Rect extends Shape {
    =
} // Rect.
```



Linked List

```
public class Linkedlist<T> {
    private class Node {
        T data;
        Node next;
    } // Node
    int n;
    Node first, last;
    =
}
```

use

```
LinkedList L = new LinkedList<shape>();
L.insert(new Circle(4.0));
L.insert(new Rect(3.0, 6.0));
for ( shape s : L )
    System.out.println(s + ":" + s.area());
```

Iterator needs to be defined in LinkedList.

dynamic method dispatch

OOP in Python

```
class Circle:
    def __init__(self, r):
        self.radius = r
    def area(Self):
        return math.PI * self.radius * self.radius

    def __repr__():
        ==
```

def area(self):

Similarly define Rect.

Use

```
c1 = Circle(4.0)
r1 = Rect(3.0, 6.0)

ca = c1.area()
ra = r1.area()
```

```
c1.radius = 4.5
r1.radius = "Deepak"
```

This is OK in Python

i.e. Python does not support encapsulation or ADTs

Do Fraction from Lab#7, if time.

OR do dunder/magic methods

```
__eq__()
__add__()
etc.
```