# 18 Data Abstraction & Structures

Monday, November 30, 2020 9:10 AM

* Assignment #5 (Lang. Reports) due on Thursday, Dec. 3
* Language Presentations on Thursday: C++, Haskell, Javascript

**Data Type:** - Name
- set of values
- set of operations

`int x;` +, -, *, /, %

**User-defined types**

**Shapes**

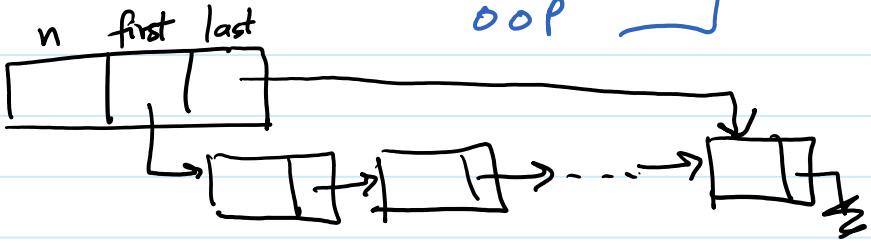
| circle | rectangle |
|---|---|
| radius | width |
| | height |
| area() | area() |

Data Abstraction → User-defined types → ADTs → objects → OOP

**Data Structures**

**Linked List**


**LinkedList**




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

# 18 Data Abstraction in C

Monday, November 30, 2020 9:11 AM

typedef int fahr;

circle.h

```
typedef struct {
    float radius;
} circle;

float areaOfCircle(c circle);
```

Interface

rect.h

```
typedef struct {
    float width;
    float height
} rect;

float areaOfRect(r rect);
```

header files

use

```
#include "circle.h"
#include "rect.h"

. . .
circle c1 = {4.0};
rect r1 = {3.0, 6.0};

float ac = areaOfCircle(c1);
float ar = areaOfRect(r1);
```

a1.radius
r1.width
r1.height

Implementation

circle.c
rect.c

Modularization is a convention!

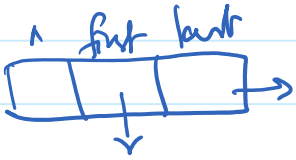
a1.radius = 0; ok

Can do data abstraction but works based on good programming habits!!

list.h

```
typedef struct {
    int data;
    node* next;
} node;

typedef struct {
    int n;
    node* first;
    node* last;
} LinkedList;
```


Linked list is not generic!

# 18 Abstract Data Types

ADTs

Monday, November 30, 2020 9:11 AM

Key Idea: Information hiding enforced by the programming language

Encapsulation

Pascal → Modula-2

SIMULA 67

Interface

circle.def

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

Implementation

circle.mod

```
IMPLEMENTATION MODULE circle;
   TYPE Circle = RECORD
                    radius : INTEGER;
                 END;

   PROCEDURE areaOfCircle(radius : FLOAT) : FLOAT;
   BEGIN
      . . .
   END;
END circle;
```

IMPORT circle;

FROM circle IMPORT .....

Good PL enforced encapsulation – ADT

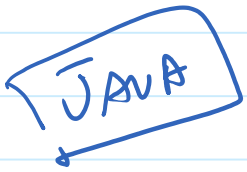
# 18 ADTs Using Classes

Monday, November 30, 2020 9:11 AM

Key idea: A class defines a type

Programmer gets to decide which parts of a class are public or private.
The programming language enforces the visibility.

Provides mechanism for true encapsulation/information hiding.




```
public class Circle {
    private float radius;

    public Circle(float r) {
        radius = r;
    }

    public float area() {
        . . .
    } // area()

    public String toString() {
        . . .
    } // toString()
} // class Circle
```

```
public class Rectangle {
    private float width;
    private float height;

    public Rectangle(float w, float h)) {
        width = w;
        height = h;
    }

    public float area() {
        . . .
    } // area()

    public String toString() {
        . . .
    } // toString()
} // class Rectangle
```

```
Circle c1 = new Circle(4.0);
Rectangle r1 = new Rectangle(3.0, 6.0);

float ca = c1.area();
float ra = r1.area();

c1.radius = 7.0;
r1.height = r1.width;
```

System.out.println(c1);

method dispatch
polymorphism

a+b

objects + polymorphism

Plus, Java has inheritance...

# 18 Object-Oriented Programming (OOP)

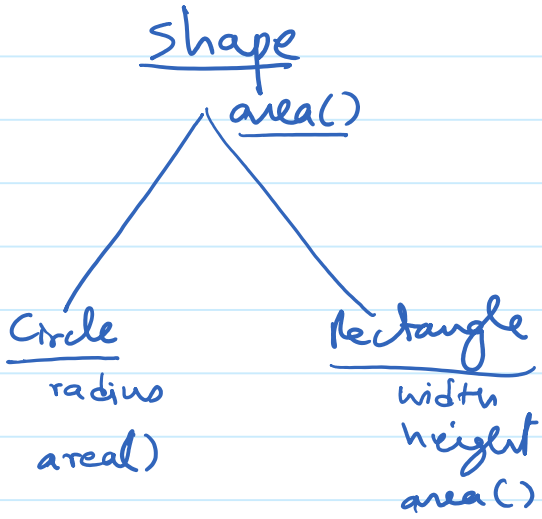
Monday, November 30, 2020 9:11 AM

OOP = Objects + Encapsulation + Inheritance

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

public class Circle extends
Shape {
    . . .
} // class Circle

Public class Rectangle extends
Shape {
    . . .
} // class Rectangle
```




```
public class LinkedList<T> {
    private class Node {
        T data;
        Node next;
    } // class Node

    int n;
    Node first;
    Node last;

    . . .
} // class LinkedList


LinkedList L = new LinkedList<Shape>();

L.insert(new Circle(4.0));
L.insert(new Rectangle(3.0, 6.0));

for (Shape s : LinkedList) {
    System.out.println(s + ":" + s.area());
}
```

Iterators

Multiple Inheritance


JAVA does not have. C++ does

Common lisp

Dynamic method dispatch

# 18 OOP in Python

Monday, November 30, 2020 9:11 AM

```
class Circle:
    def __init__(self, r):
        self.radius = r

    def area():
        . . .

    def __str__():
        . . .
```

constructors

```
class Rectangle:
    def __init__(self, w, h):
        self.width = w
        self.height = h

    def area():
        . . .

    def __str__():
        . . .
```

Print method

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

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

c1.radius = 45
r1.radius = "yuge!"
```

??

This is OK!!

FFI → SWIG

__init__(self, ____):

__str__(self):

Python does not have ENCAPSULATION or ~~ADTs~~ !!

# 18 OOP in Go?

Monday, November 30, 2020 9:12 AM

Go is not an OOP language

→ objects + methods
→ ADTs

Interface

types

methods

```
type shape interface {
    area() float64
}

type circle struct {
    radius float64
}

type rect struct {
    width int
    height int
}

func (c circle) area() float64 {
    . . .
}

func (r rect) area() float64 {
    . . .
}

func main() {
    c1 := circle{4.0}
    r1 := rect{3.0, 6.0}

    fmt.Println(c1.area())
    fmt.Println(r1.area())

    . . .
    c1.radius =
    r1.width =
```

c . . . .

r

?

ok!!

type fahr float64
type celsius float64

var c celsius = 100.0 ✓
var f fahr ✓

f = c ? ✗

f = tofahr(c)

shape s := circle(4.0)

BUT Go has packages

Illegal

```
package shapes

type Circle struct {
    radius float64
}

type Rect struct {
    width float64
    height float64
}
```

```
func (c Circle) Area() {
    |
}
```