# Object-Oriented Programming

- Object-oriented programming (OOP) is a ***methodology***

  = abstract data types (ADTs)

  + inheritance

  + dynamic binding

- OO programming languages (OOPLs) have features to ***support*** this methodology

| | | |
|---|---|---|
| Simula | 1967 | static inheritance and dynamic binding |
| CLU | 1980 | abstract datatypes |
| Smalltalk | 1980 | |
| ZetaLisp | 1984 | (along with other OO Lisps) |
| C++ | 1985 | |
| Objective-C | 1988 | |
| Oberon | 1988 | |
| SELF | 1989 | |
| Modula-3 | 1990 | |
| Oberon-2 | 1991 | |

and many more...

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

# C++ vs Modula-3 (A View)

| feature | C++ | Modula-3 |
|---|---|---|
| **safe** | no | yes |
| **efficient** | yes | yes |
| **garbage collection** | any day now | yes |
| **static typechecking** | mostly | yes |
| **enforced interfaces** | yes | yes |
| **concurrency** | no | yes |
| **widely available** | yes | no |
| **everyone knows it** | so they claim | no |
| **software tools** | yes | some |
| **good for a summer job** | probably | no |

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

# A Brief History of Modula-3

| | | |
|---|---|---|
| Algol-60 | 1960 | procedures, parameters, arrays, BEGIN-END, WHILE-DO, FOR-DO |
| Simula-67 | 1967 | Algol-60 plus pointers, records, inheritance |
| Pascal | 1971 | Simula-67 cleaned up minus inheritance |
| Modula | 1978 | Pascal plus abstract data types |
| Modula-2 | 1980 | Modula cleaned up plus interfaces |
| Modula-2+ | 1985 | Modula-2 plus concurrency and exceptions |
| Oberon | 1988 | Modula-2 stripped down plus inheritance |
| Modula-3 | 1990 | Module-2+ cleaned up plus objects, inheritance, and dynamic binding (influenced by Oberon) |
| Oberon-2 | 1991 | Oberon plus methods |

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

# Abstract Data Types

- ADTs support *data abstraction*
  ***hides*** representation details
  reduces *coupling*
  promotes *reuse*
- Procedures help make code representation independent, e.g., interface `line.h`

```
#ifndef LINE_INCLUDED
#define LINE_INCLUDED
#define MAX_LINE_SIZE 256
typedef struct Line_T *Line_T;
extern Line_T Line_New( char *text );
extern char *Line_Text( Line_T line );
extern void Line_Insert( Line_T current, Line_T new );
extern void Line_Delete( Line_T line );
extern Line_T Line_Split( Line_T line, int col );
       ...
```

- ADTs are a programming ***methodology***:
  applicable in any language, e.g., C
  easy to *preach*, hard to *do*
- ADT languages have features to ***support*** this methodology

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

# Inheritance

- Example: shapes in a graphics system
- Interface: `shape.h`

```
#ifndef SHAPE_INCLUDED
#define SHAPE_INCLUDED

#include "point.h"

typedef enum { circle, square, triangle, ... } Kind_T;

typedef struct Shape_T {
    Kind_T tag;
    Point_T center;
    union {
        float radius;  /* circle */
        float side;    /* square */
        ...
    } u;
} *Shape_T;

extern Point_T Shape_where(Shape_T s);
extern void Shape_move(Shape_T s, Point_T to);
extern void Shape_draw(Shape_T s);

#endif
```

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

# Inheritance, cont'd

- Implementation: **`shape.c`**

```
#include "assert.h"
#include "shape.h"

#define T Shape_T

Point_T Shape_where(T s) {
    return s->center;
}

void Shape_move(T s, Point_T to) {
    s->center = to;
    Shape_draw(s);
}

void Shape_draw(T s) {
    switch (s->tag) {
    case circle: ...
    case square: ...
    case triangle: ...
    ...
    default: assert(0);
    }
}
```

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

# Problems with ADTs

- Problems

  **`Shape_draw`** must know about ***all*** shapes

  new shapes require inspecting/modifying ***all*** functions

- ADTs don't distinguish between ***general*** and ***specific*** properties

  e.g., between all shapes and just circles

- OOPLs support this distinction

  ADT is a ***class***

  a class ***inherits*** the fields and functions of its ***superclass***

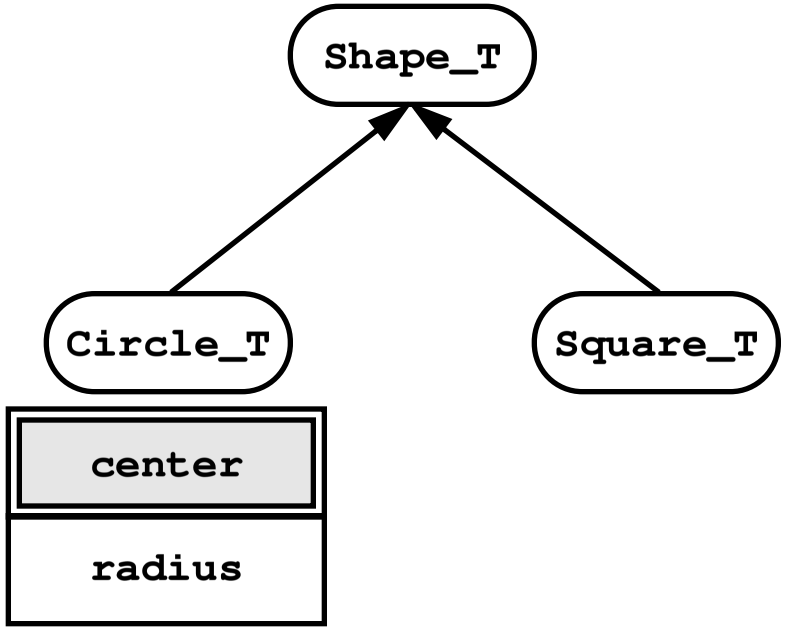
  ***objects*** are ***instances*** of a class

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

# Using Inheritance

- Revised interface: **shape.h**

```
#ifndef SHAPE_INCLUDED
#define SHAPE_INCLUDED

#include "point.h"

typedef struct Shape_T {
    Point_T center;
} *Shape_T;

typedef struct Circle_T {
    struct Shape_T super;
    float radius;
} *Circle_T;

typedef struct Square_T {
    struct Shape_T super;
    float side;
} *Square_T;

...
#endif
```




Note: a **Circle_T** is also a **Shape_T**, etc. **Circle_T** is a ***subtype*** of **Shape_T**, and **Shape_T** is a ***supertype*** of **Circle_T**; ditto for **Square_T**

- What about **Shape_draw**, etc?

Copyright ©1995 D. Hanson, K. Li & J.P. Singh