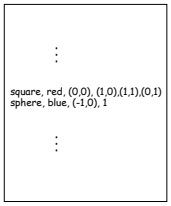
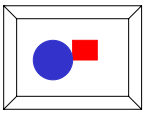
oo-intro

task





draw red square(v_1,v_2,v_3,v_4)

Shapes: square, circle

Colors: red, green, blue

rule

- draw red circles
- draw green squares
- draw blue squares

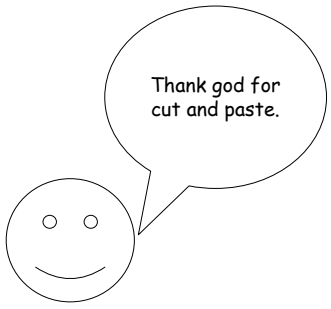
pseudo code

```
Open file
While not at end of file
   read shape, color
        if shape=square then read four vertices
                if color is blue then "draw blue square…"
                if color is green then "draw green square…"

        if shape=circle then read center and radius
                if color is red then then "draw red circle…"
Close file
```

whoops … I meant

- Shapes: square, circle, triangle
- Colors: red, blue, green, purple
- Rule:
  - Draw blue and purple squares
  - Draw red and green circles
  - Draw every triangle



hacker joe

pseudo code

```
Open file
While not at end of file
    read shape, color
        if shape=square then read four vertices
                if color is blue then "draw blue square…"
                if color is purple then "draw green square…"
        if shape=circle then read center and radius
                if color is red then "draw red circle…"
                if color is green then then "draw red circle…"
        if shape=triangle then read four vertices
                if color is blue then "draw blue square…"
                if color is purple then "draw green square…"
                if color is red then "draw red circle…"
                if color is green then then "draw red circle…"


Close file
```

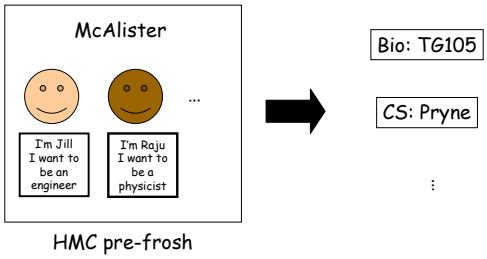
pseudo code

```
Open file
While not at end of file
    read shape, color
        if shape=square then read four vertices
                if color is blue then "draw blue square…"
                if color is purple then "draw green square…"
        if shape=circle then read center and radius
                if color is red then "draw red circle…"
                if color is green then then "draw red circle…"
        if shape=triangle then read four vertices
                if color is blue then "draw blue square…"
                if color is purple then "draw green square…"
                if color is red then "draw red circle…"
                if color is green then then "draw red circle…"


Close file
```

whoops … I meant

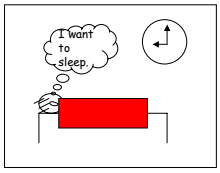
another task



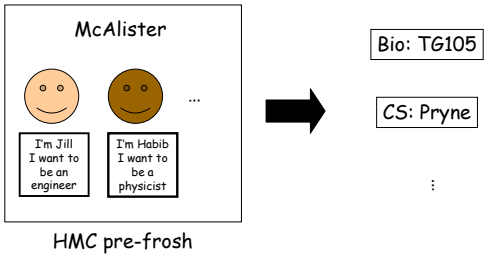
HMC pre-frosh

where are the CS pre-frosh?

somewhere in west



another task



HMC pre-frosh

procedure

- post classroom assignments
- post campus map
- tell students to
  - refer to the posted notices
  - go where they belong

whoops

CS has been moved to pepsi room

procedure

- post classroom assignments
- post campus map
- tell students to
  - refer to the posted notices
  - go where they belong

strategies

functional approach
- control is concrete, specific
- rigid, hard to change

object oriented approach
- control is abstract, general
- adaptable, easy to change

change is inevitable!

functional → OO

- modularization

modularize

```
Open file
While not at end of file
   read shape, color
        if shape is square then read vertices
                testAndDrawSquare(color, vertices)
        if shape is circle then read center and radius
                testAndDrawCircle(color, center, radius)
Close file
```

functional → OO

- modularization
- user-defined data types

user define data types

```
Open file
While not at end of file
   read shape, color
        if shape=square then read vertices and create theSquare
                testAndDrawSquare(theSquare)
        if shape=circle then read center, radius and create theCircle
                testAndDrawCircle(theCircle)
Close file
```

functional → OO

- modularization
- user-defined data types
- union (abstract data types)

abstract data types

```
Open file
While not at end of file
   theShape = readNextShape()
   testAndDrawShape(theShape)
Close file
```

functional → OO

- modularization
- user-defined data types
- union (abstract data types)
- encapsulation

encapsulation

```
Open file
While not at end of file
   theShape = readNextShape()
   theShape.testAndDraw()
Close file
```

cohesion & coupling

- cohesion: how closely the operations in a routine are related.
- coupling: the strength of a connection between two routines

strategies

functional approach
- control is concrete, specific
- rigid, hard to change
- low cohesion
- high coupling

object oriented approach
- control is abstract, general
- adaptable, easy to change
- high cohesion
- low coupling

principles of oo-something

- high cohesion
- low coupling

to be continued…

oo-something

- oo analysis: domain concepts
- oo design: software classes & their interfaces
- oo programming: implementation of classes

oo-something

- oo analysis
- oo design
- oo programming
- principles
- pattern

Dependency-Inversion Principle (Robert C. Martin)

- **Details should depend on abstractions; abstractions should not depend on details.**
- High-level modules should not depend on low-level modules; both should depend on abstractions.
- In other words, don't let low-level modules "call the shots" for high-level ones. The high-level ones are where policies should be set.
- Succinctly: ***Specify the interface first***, then implement.

Open/Closed Principle
(Robert C. Martin)

- Classes should be both "open" and "closed":
  - Open: means that the class can be extended through inheritance.
  - Closed: means that the functionality of a class, once set, should not be modified retroactively.
- In other words, add functionality by adding new code, not rewriting old code.

Liskov Substitution Principle (LSP)

As popularly stated:

A member of a derived class must also make sense when used as a member of the base class.

For example, if a method has an object of a class as an argument, the same method should be able to work with an object of a derived class.

As originally stated:

Let $\varphi(x)$ be a property provable about objects x of type T. Then $\varphi(y)$ should be true for objects y of type S, where S is a subtype of T.

B. Liskov and J. Wing, A behavioral notion of subtyping, ACM TOPLAS, **16**, 6 (Nov. 1994), 1811-1841.

(Prof. Barbara Liskov, M.I.T.)

LSP **Corollary**
(Robert C. Martin)

- Functions that use *pointers* or *references* to base classes must be able to use objects of derived classes **without differentiation** as to base vs. derived.

Law of Demeter (LoD)
colloquial version

- "Only talk to your immediate friends".
- [not meaning *friends* in the sense of C++ classes and methods]
- so named by Prof. Karl Lieberherr, Northeastern University:
- LoD home page:
  http://www.cs.neu.edu/home/lieber/LoD.html

LoD: More Specific Interpretation

- An object should only invoke methods of:
  - objects that are ***declared*** within it
  - objects that are ***parameters*** *of the method*
  - ***itself***
  - objects that it ***creates***
- mnemonic DPIC ("depict")

Precluded by LoD

- Do not extract object B from an object A and perform an operation on it.
- Instead, recast what you want to do as an operation on A. That operation may call operations on B.

Exercise

- Find an example of each principle in POP.
- Try to find a violation of each principle in POP.