design principles intro

Processes

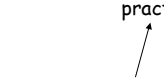
- Requirements ✓
- Design
- Implementation
- Testing

Design

practices, principles, patterns

Design

practices, principles, patterns



e.g. diagrammatic modeling

Design

practices, principles, patterns

e.g. "no forgery"

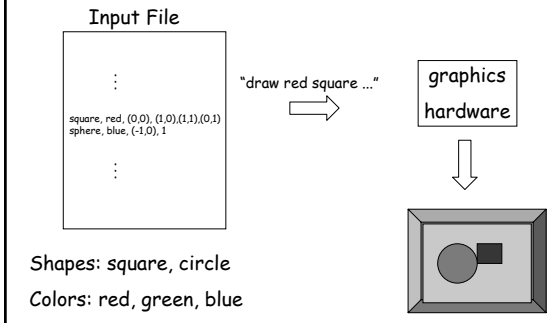
Design

practices, principles, patterns

e.g. singleton

Today: Intro to Design Principles

- background/history
- a few basic principles



Shapes: square, circle

Colors: red, green, blue

rule

- draw red circles
- draw green squares
- draw blue squares
- ignore everything else

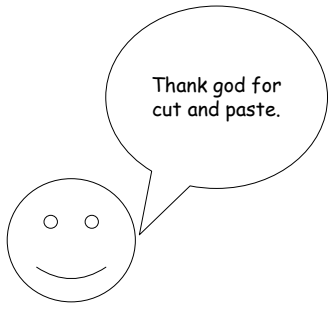
pseudo code

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

        else if shape is circle then read center and radius
                if color is red then then "draw red circle…"
        else skip this line of input
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
  - Ignore everything else



hacker joe

pseudo code

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


Close file
```

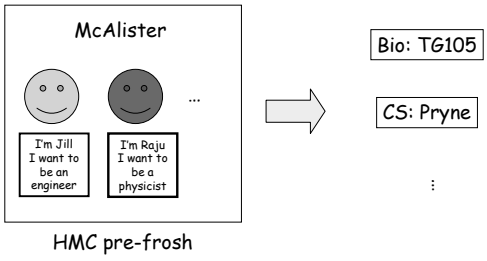
pseudo code

```
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…"
          else 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…"
          else 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…"
          else skip this line of input


Close file
```

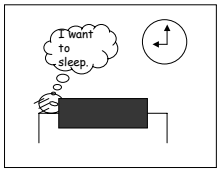
whoops … I meant

new task

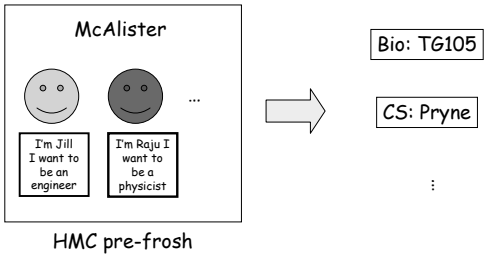


where are the CS pre-frosh?

somewhere in west



new task



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 ⟺ object oriented

NOTE: this has nothing to do with "functional languages"

strategies

functional
step by step procedure

⟺

object oriented
delegate responsibility

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!

History: 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
```

Today: Intro to Design Principles

- background/history
- a few basic principles

Dependency-Inversion Principle
(Robert C. Martin)

- Details should depend on abstractions; abstractions should not depend on details.
- Write class interfaces before you begin coding.
- High-level modules should not depend on low-level modules.

Open/Closed Principle
(Robert C. Martin)

Class design:

- Classes should be "open" in that they can be easily extended through inheritance.
- Classes should be "closed" in that their functionality (once set) should only be changed through inheritance.

Liskov Substitution Principle (LSP)

An instance of a derived class must also make sense when used as an instance of the base class.

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

Law of Demeter (LoD)

"DPIC"

"Only talk to your immediate friends".

An object should only communicate with
- objects that are ***declared*** within it
- objects that are ***parameters*** *of its methods*
- ***itself***
- objects that it ***creates***

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.