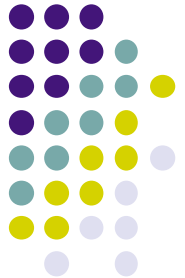
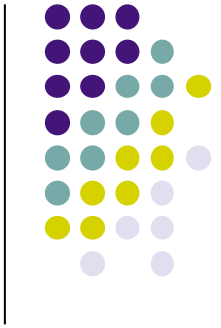
Polymorphism



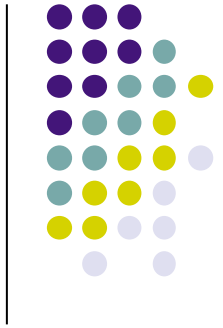
- In everyday language, an entity is polymorphic if it can assume different forms
  - The Greek god Proteus is polymorphic; he is a shape-shifter able to assume many forms
- In computing, an operation is polymorphic if it works correctly for arguments of different types
  - For example, an object message is polymorphic if many different objects will accept it
- This ability is needed in order to properly apportion responsibility over different parts of a program
  - A single responsibility should not be spread out; it should rather be concentrated in one place if possible

The responsibility principle



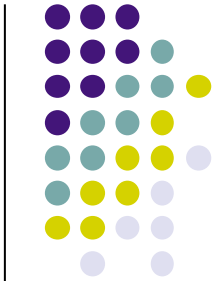
- Polymorphism allows to isolate responsibilities to the parts of the program that are concerned with them
  - A responsibility should be concentrated in one part of the program
- Example: a patient goes to see a medical doctor
  - The patient does not have to be a doctor!
  - The patient tells the doctor: "cure me"
  - The doctor understands this message and does the right thing (either cures the patient, or sends the patient to another doctor; we assume that eventually the right doctor is found!)
- The message "cure me" is polymorphic: it works with all medical specialties
  - All doctors understand the message "cure me"
  - The ability to cure a specific illness is concentrated in the doctor whose specialty covers that illness; we assume there is a mechanism to find the right doctor (for example, the generalist directs you to a specialist)

Implementing polymorphism



- All data abstractions we have seen can support polymorphism
  - Both objects and ADTs support it
  - But it is especially simple for objects
    - This is one reason for objects' enormous success
  - In this course, we will only talk about object polymorphism
    - The book also explains ADT polymorphism, if you are curious
- The idea is simple: we define the interface that the program needs
  - Then the program can accept all abstractions with that interface

Example: drawing of geometric figures

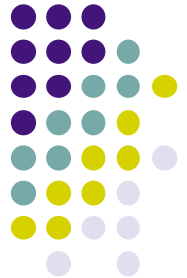


```
class Figure
    …
end
class Circle
    attr x y r
    meth draw … end
    …
end
class Line
    attr x1 y1 x2 y2
    meth draw … end
    …
end
```

```
class CompoundFigure
    attr figlist
    meth draw
        for F in @figlist do
            {F draw}
        end
    end
    …
end
```

This definition of draw in CompoundFigure works for all possible figures: circles, lines, and other CompoundFigures!

Correctness of a polymorphic program



- When is a polymorphic program correct?
  - To be correct, each abstraction that the program accepts needs to satisfy certain properties (namely, those needed by the program)
  - For each abstraction, we need to verify that its specification has those properties
- For the figure drawing example, each draw method must correctly draw the object's figure
- For the doctor example, all doctors must cure the patient for their specialty
  - And for patients with another illness, the doctor must send the patient to a doctor better able to cure the illness (no cycles to avoid infinite loops!)