Objects, Object-Oriented Design, Methods
02-201 / 02-601
“Objects”

- Top-down design: start from the big problem and break it into smaller problems, writing a function for each of the smaller problems.

- Another useful way of thinking: describe the organization of your data and have that reflected in your program.
  - A contact management program will manipulate Contacts
  - A drawing program will manipulate a Canvas, and perhaps Lines, Colors, and Shapes
  - Facebook will manipulate Users, Posts, and Advertisements
  - Twitter will manipulate Tweets, Users, Advertisements

- These are the “nouns” of these programs.
These two ways of thinking complement each other
Objects + Operations

• Once you’ve decided on the “nouns”, you choose the “verbs” that apply to those nouns.

• Example:

  Your “noun” is a Tweet:

  ```go
  type Tweet struct {
      text string
      time uint64
      who *User
  }
  ```

  • Get Hashtags in Tweet
  • Get Direct Mentions in Tweet
  • Shorten URL in Tweet
  • Get URLs in Tweet
  • Get Short Version of Tweet

  Your “noun” is a User:

  ```go
  type User struct {
      name string
      followers []*User
      following []*User
      tweets []*Tweets
  }
  ```

  • Direct Message User
  • Add Follower
  • Remove Follower
  • Add Following User
  • Remove Following User
  • Get All Tweets from Followed Users
User structs are nodes
Pointers represent edges

type User struct {
    name string
    followers []*User
    following []*User
    tweets []*Tweets
}

Twitter Graph
Example 2: Contacts

Your “noun” is a Contact:

```go
type Contact struct {
    name string
    id    int
    salary float64
    friends []*Contact
    phone []int
}
```

Operations you will need to perform on a Contact:

- Get First Name
- Get Last Name
- Set First Name
- Set Last Name
- Get Formatted Phone Number
- Call
- Count Friends
- Add Friend
- Give Raise
Example 3: Spatial Games

```go
type Field struct {
    cells [][]Cell
}
```

- Count Cell Kinds in Neighborhood
- Read Field From File
- Evolve Single Step
- Save Field To File
- Draw Field
- Check Field is Valid
- Zero All Scores

```go
type Cell struct {
    kind  string
    score float64
    prevKind string
}
```

- Zero Score
- Set Kind
- Get Kind
- Get Previous Kind
- Get Cell Color
Example 4: Real World Example

- Too complex to be a good example for class, but I wanted to show that this kind of thinking is actually used:
Example 5: Canvas

definition of Canvas type

```go
type Canvas struct {
    gc *draw2d.ImageGraphicContext
    img image.Image
    width int
    height int
}
```

Operations you can perform on a Canvas:

- `MoveTo(c *Canvas, x, y float64)`
- `LineTo(c *Canvas, x, y float64)`
- `SetStrokeColor(c *Canvas, col color.Color)`
- `SetFillColor(c *Canvas, col color.Color)`
- `SetLineWidth(c *Canvas, w float64)`
- `Stroke(c *Canvas)`
- `FillStroke(c *Canvas)`
- `Fill(c *Canvas)`
- `ClearRect(c *Canvas, x1, y1, x2, y2 int)`
- `SaveToPNG(c *Canvas, filename string)`
- `Width(c *Canvas)`
- `Height(c *Canvas)`

These operations are logically related: they are the things you can do to a Canvas.

They are functions called “methods”.

They all take a *Canvas as their first parameter.

Go provides a special syntax for this situation (next slide).
Go’s Method Syntax

• Same as a function definition, with one addition:

```go
func (c *Canvas) SetStrokeColor(col color.Color) {
    c.gc.SetStrokeColor(col)
}
```

Move the logical “first” parameter to before the name of the function

Can use “c” just like any other parameter

• Now use “dot” syntax to call the method:

```go
var pic *Canvas = MakeCanvas()
pic.SetStrokeColorColor(blue)
```
What’s the Point?

• Logically groups operations with the data they operate on

• Supports the “noun” / “verb” way of designing programs directly

• Let’s you use the same function name for different object types:
  
  • (c *Canvas) Draw()

  • (b *Button) Draw()

are different functions.
Method Summary

• Methods are functions that are associated with a type.

• If you have a variable X, you can call any of its methods using:

  X.methodName(param1, param2)

  This works like a normal function call.

• This is “object-oriented programming”
Object-Oriented Design Summary

• Create types for the things your program will manipulate

• Write methods for each of those types that perform the operations on those things that you will need.

• Use those methods to solve the tasks you are aiming to solve.