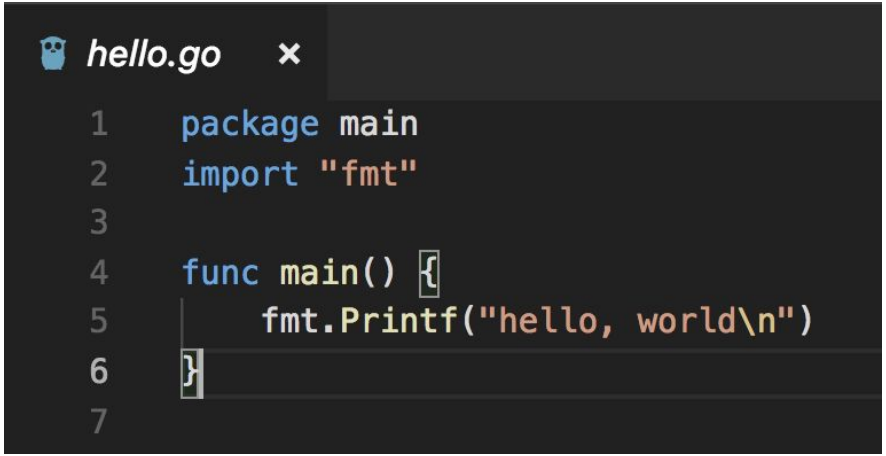
Stupid simple!

Vladimir Andrianov

Golang Estonia Meetup @ Mooncascade

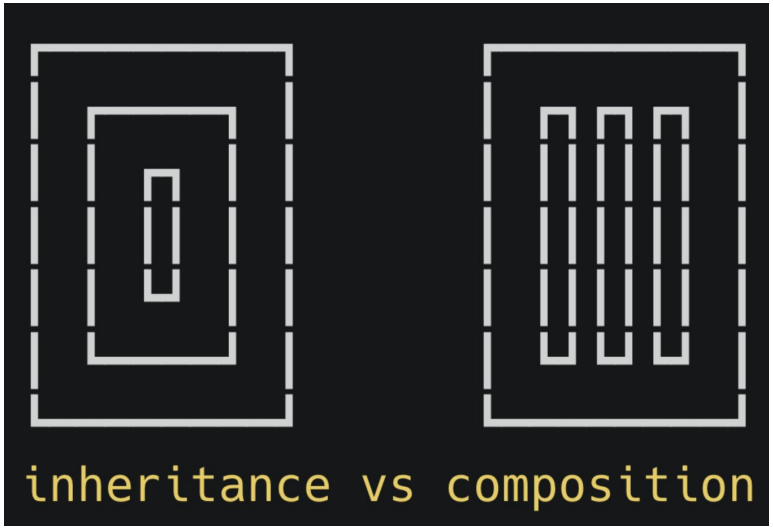
Simplicity in Go

- Easy to start
- Good to migrate
- Simple project layout
- Huge standard library
- Fast build



Back to the Past

- Composition over inheritance
- Strongly typed
- No cyclic imports
- Clean dependency graph



Code meant to be read by humans

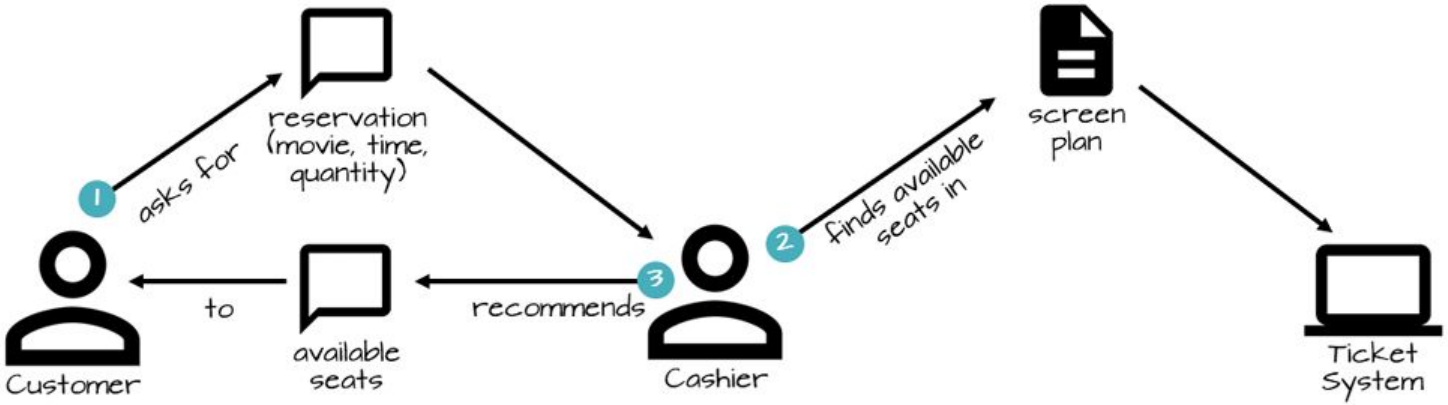
- Verbose and explicit
- Straightforward
- No magic
- Simple

Good principles

- **KISS - keep it simple, stupid/keep it stupid simple**
- DRY - Don't repeat yourself
- DDD - Domain driven design

Domain Driven Design

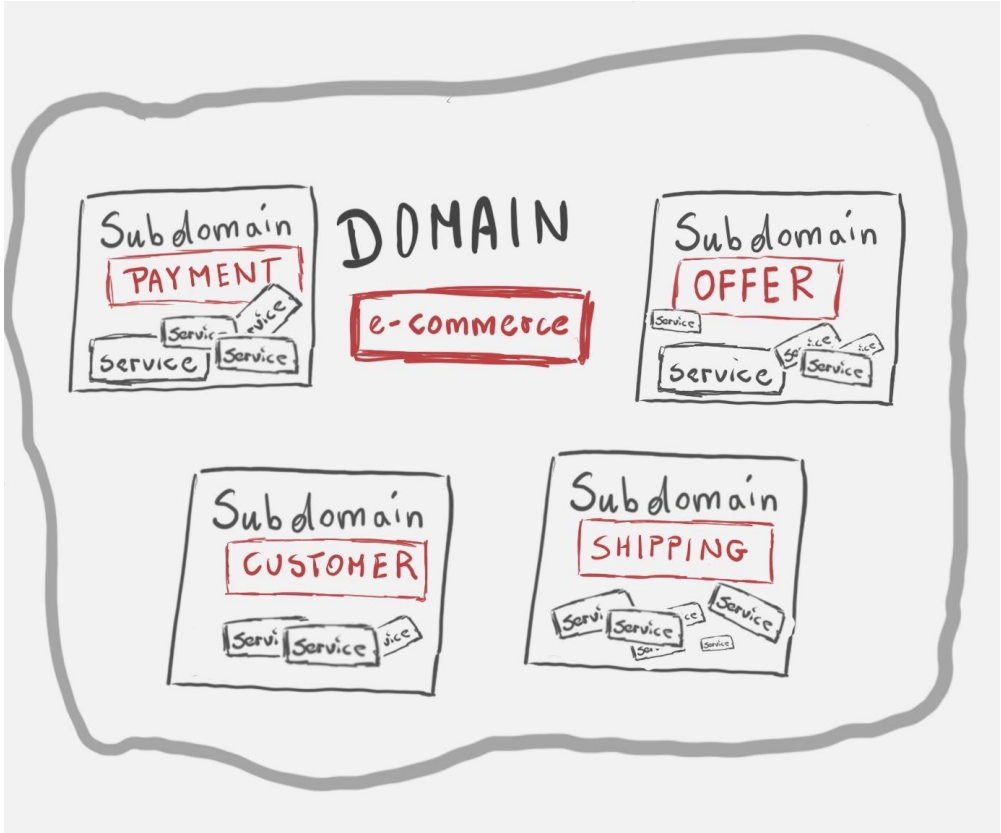
- What is DDD?
- Why does it matter?



Domain Storytelling. Retrieved from: https://domainstorytelling.org/

Domain

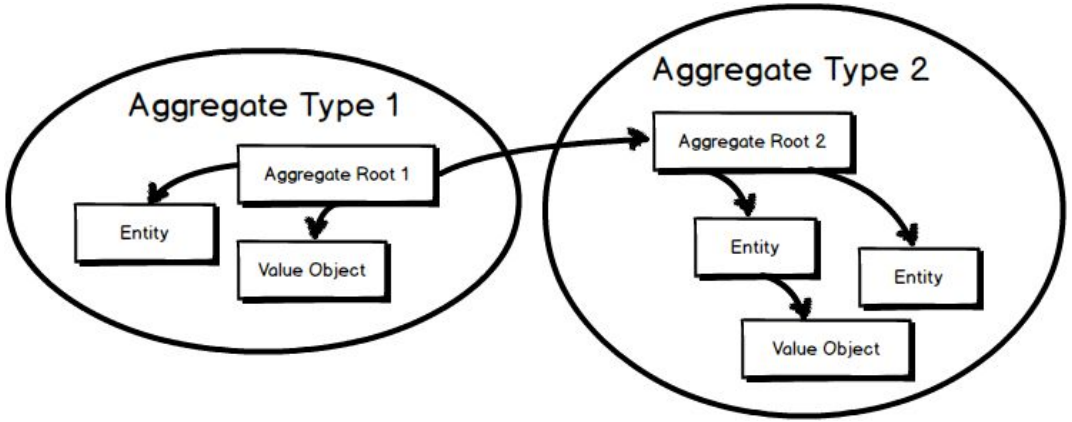
- What is a domain?
- What is a subdomain?
- How does it work?



Braintelligence. The nature of Domain Driven Design - what the heck is it? Retrieved from: http://www.braintelligence.pl/wpcontent/uploads/2018/06/Untitled_Artwork-8.jpg.

DDD concepts

- What is an use case?
- What is a command?
- What is a state?
- What is an aggregate?



DDD states in Go

- Guestbook invite model
- Different use cases
- Separate states
- No redundant data
- Code reused is not duplicated

Invite states

```
type Created struct {
    Id   event.AggregateId
    Name string
}

type Accepted struct {
    Id event.AggregateId
}

type Declined struct {
    Id event.AggregateId
}
```

DDD commands in Go

- Separate commands
- Explicit state control

```
case Accepted:
    invite.Accepted = true
case Declined:
    invite.Declined = true
case Created:
    invite.Name = ev.Name
```

```
type Aggregate struct {
    event.Aggregate
    Info
}

type Info struct {
    Name     string
    Accepted bool
    Declined bool
}
```

```
func (invite *Aggregate) Accept() error {
    if invite.Declined {
        return fmt.Errorf("%s already declined", invite.Name)
    }
    if invite.Accepted {
        return nil
    }
    invite.Apply(Accepted{invite.Id})
    return nil
}
```

HTTP API in Go

- Minimal code to start server
- Standard library

```go
func main() {
    http.HandleFunc("/createInvite", invitesHandler)
    fmt.Printf("Starting server for testing HTTP POST...\n")
    if err := http.ListenAndServe(":9805", nil); err != nil {
        log.Fatal(err)
    }
}
```

```
API server listening at: 127.0.0.1:9805
Starting server for testing HTTP POST...
2020/02/04 00:53:46 Invite 86897c61-9bb3-4b40-b73d-c0fc1e179689 has been created for Vlad
2020/02/04 00:53:53 Invite 8bfe48af-7729-4a14-baa0-e4a70d4d0322 has been created for Gophers
```

API endpoint in Go

- API & invites wired together

```go
func invitesHandler(w http.ResponseWriter, r *http.Request) {
    if r.Method == "POST" {
        body, err := ioutil.ReadAll(r.Body)
        if err != nil {
            http.Error(w, "Error reading request body", http.StatusBadRequest)
        }
        newInviteID, err := service.NewInvite(string(body))
        if err != nil {
            http.Error(w, "Error creating new invite", http.StatusInternalServerError)
        }
        log.Printf("Invite %v has been created for %v", newInviteID, string(body))

    } else {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
    }
}
```

BDD testing in Go

- What is BDD?
- What is GinkGo?
- Structured tests
- Specifications in tests

Self explaining tests

- Structure
- Reads like a story
- Well defined blocks
- Test patterns
- Domain overview

```
Describe("Accepting an invite", func() {
    When("the invite is accepted", func() {
        Specify("the invite is persisted in the database", func() {
        })
    })

    When("the invite is already accepted", func() {
        Specify("the invite is persisted in the database", func() {
        })
    })

    When("the invite does not exist", func() {
        Specify("an invite does not exist error is returned", func() {
        })
    })

    When("the invite is already declined", func() {
        Specify("an invite already declined error is returned", func() {
        })
    })
})
```

Simplicity benefits

- Easy to model & track
- Self-explaining code
- Well defined specifications
- Explicit and straightforward
- Quick jump in

Keep it simple!