AARHUS UNIVERSITET

Software Engineering and Architecture

Roles, Responsibility, Behavior, Protocol

Programming Models

- "The way we think about programs"…
  - A program is a sequence of *instructions* operating on *data*
    - *Procedural thinking* (How CPUs actually operate)
      - Python, C, Pascal, Assembler
  - A program is a sequence of *pure functions*, taking input and producing output
    - *Functional thinking* (Mathematical computer science likes that a lot)
      - F#, Scala, Haskel, ML
  - A program is organized as *interacting objects*, encapsulating both data and operations
    - *Object-oriented thinking* (Closer to how humans think)
      - *C#, Java, C++*

AARHUS UNIVERSITET

Object-Orientation

- Object-Orientation (OO) is about objects…
- *But what is an object?*
- It turns out – that there are several ways of thinking…
- ***Language centric* perspective:**
  - Object = Data + Actions
- ***Model centric* perspective:**
  - Object = Model element in domain
- ***Responsibility centric* perspective:**
  - Object = Responsible for providing service in community of interacting objects

Competing or Complementing?

- These three models/ideas/perspectives
  - ***Language centric* perspective**
  - ***Model centric* perspective**
  - ***Responsibility centric* perspective**

- … are not "right or wrong" or competitors…
- Rather they are all valid and sort of complement each other…
- *However, as 'design and thinking tool' for developing complex software architectures, you need to master all!*

AARHUS UNIVERSITET

Language Perspective

- Language perspective
  - An object is a set of methods and variables grouped together
  - Yes, this is true!
    - The compiler treats it like that…

```
public class Foo {
  private int x;
  public static double y;

  public int double(int x) {
    return 2*x;
  }
}
```

- **But it does not help me to develop maintainable architectures and programs ☹**
  - No guidance on “what classes/what methods” to produce…

- WarStory…

AARHUS UNIVERSITET

Model Perspective

- Model centric focus
  - focus on concepts and relations in the **Domain**
    - generalization, association, composition
  - problem domain modeling
  - object = part of model

```
public class Account {
   int balance;
   public Account() { balance = 0; }
   public void withdraw( int amount ) {
      balance -= amount;
   }
}
```

Kristen Nygaard (1978)

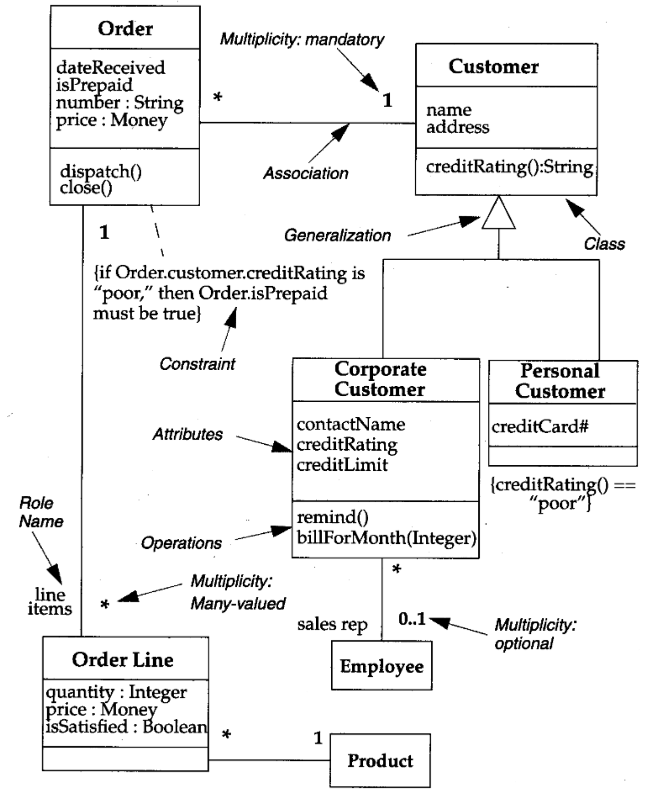
Ole-Johan Dahl (1978)

Strong Scandinavian Research Impact
Simula
1960-1990.
Alan Kay / Xerox PARC / **Smalltalk 80**
1980

Model Perspective

- Model centric focus
  - *A program execution is viewed as a physical model simulating the behavior of either a real or imaginary part of the world.*
  - *[Madsen, Møller-Pedersen, Nygaard 1993]*

- Talk to customer and identify “things” they talk in terms of. Then “model” these in the program: **Domain Modelling.**



Model Perspective

- **This perspective helps me greatly in my architecture and design of my program…**
  - “We want a card game played by two heroes”
  - *Better make a **Card** class and a **Hero** class*

- Design process is a *Who / What cycle*
  - **Who**: the objects comes **first**
  - **What**: the behavior comes **second**

IntProg and the BlueJ system is rooted in this paradigm…

- ***Define the classes, next define their methods…***

AARHUS UNIVERSITET

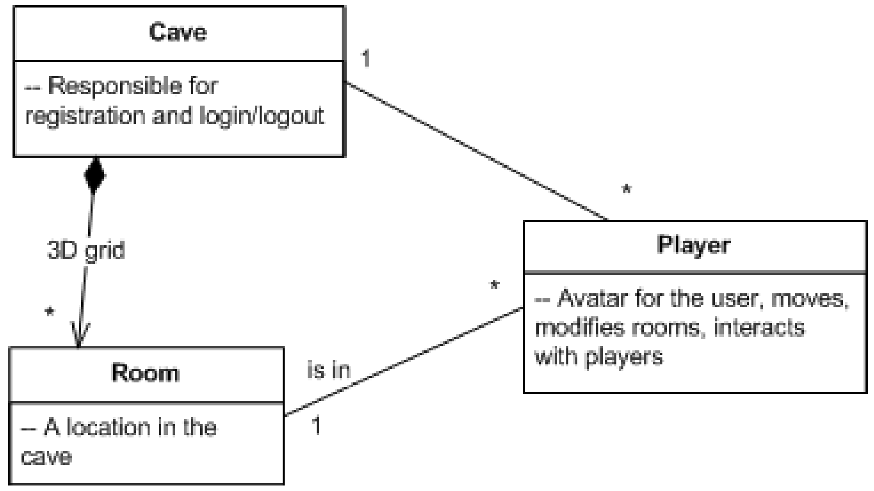
Critique

- I developed using this paradigm for 10 years…
- **And it caused me great trouble. I always ended up in**
  - **The Blob / God class**

- The issue is that 'domain' (= core business concepts) only covers a fraction of all the objects we need for a large IT system!
  - Design patterns do not appear in the domain. UI does not appear in the domain. Databases, networks, fault tolerance, security, performance optimizations, testing, etc. *does not appear in the domain…*

Example: SkyCave

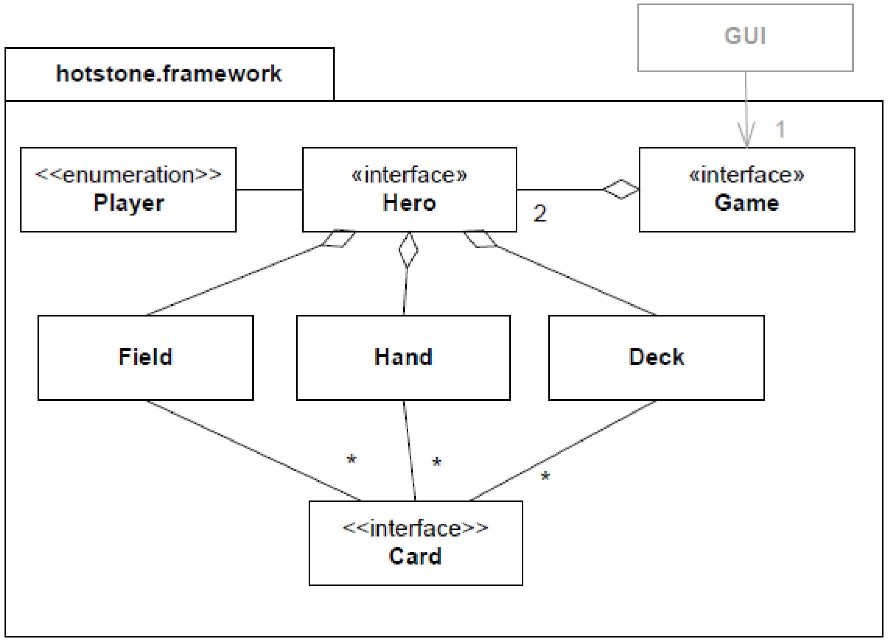
- From my *Microservice and DevOps* course
  - Domain model:
    - **Three** Concepts
  - Implementation model:
    - **94** classes
    - Patterns, dep. injection, network, databases, caching, availability, performance, …


Cave
-- Responsible for registration and login/logout
1
*
3D grid
*
Player
-- Avatar for the user, moves, modifies rooms, interacts with players
*
Room
-- A location in the cave
is in
1

Example: HotStone

- If strictly *Model* based
  - A) Identify landscape of concepts
  - B) Distribute behavior over this landscape
- … then I would only have
  - 3 to 5 classes

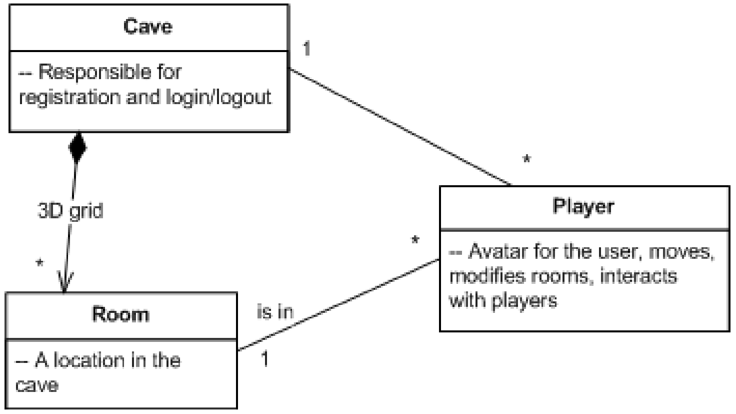
- My solution code have over 100 interfaces/classes!
  - Strategies, dep inject, distribution, GUI, caching, testing, name services, logging, database recordings, …



AARHUS UNIVERSITET

Critique

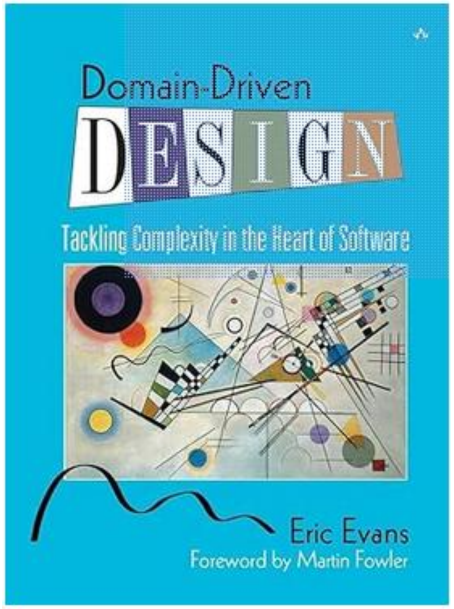
- Design process is a *Who / What cycle*
  - **Who**: the objects comes **first**
  - **What**: the behavior comes **second**
- … will make me end up with **few classes with zillions of methods covering all kinds of aspects ☹**
  - That is: **The Blob**


Cave
-- Responsible for registration and login/logout
1
*
3D grid
*
Player
-- Avatar for the user, moves, modifies rooms, interacts with players
*
is in
1
Room
-- A location in the cave

AARHUS UNIVERSITET

Not a Wrong Thinking per se…

- There is a lot of merits to Domain Modelling
  - Idea of Bounded Contexts is a prevailing way of organizing *microservices*
- The point is, if you *only* create objects/ classes from these domain concept, they will be overcrowded by too many responsibilities… Blobbing…

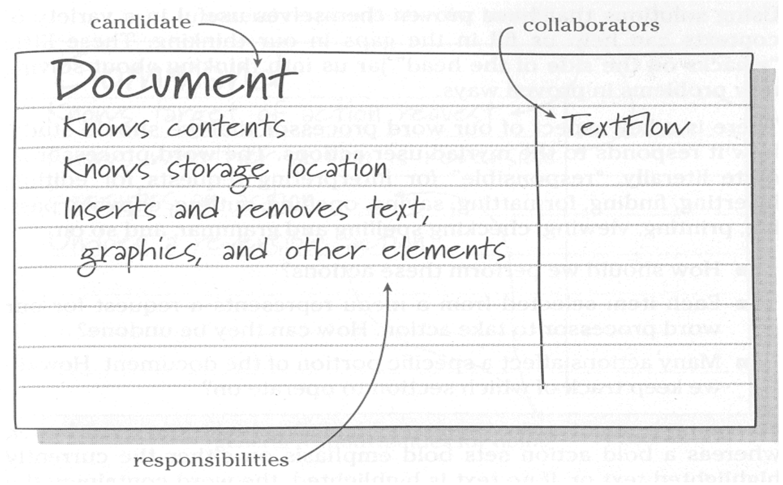

Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

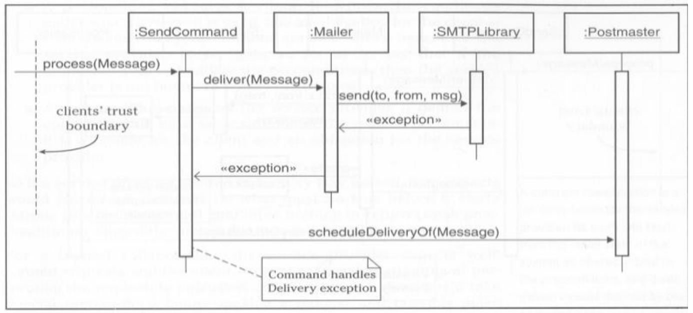
AARHUS UNIVERSITET

Responsibility-centric

Focus

- Responsibility centric focus
  - Role, responsibility, and collaboration
  - Object = provider of service in community
  - Leads to strong *behavioral* focus
  - CRC cards (Kent Beck, Rebecca Wirfs-Brock)





AARHUS UNIVERSITET

Another Definition

- Another definition:
  - *An object-oriented program is structured as a* ***community*** *of* ***interacting agents*** *called objects. Each object has a* ***role*** *to play. Each object* ***provides a service*** *or performs an action that is used by other members of the community.*
  - *Budd 2002*

- Shifting focus
  - away from “model of real world”
  - towards “community”, “interaction”, and “service”

AARHUS UNIVERSITET

Service

- Budd's definition is more skewed towards the functionality of the system.

  - **At the end of the day, software pays the bill by providing *functionality* that the users need, not by being a nice model of the world!**

- Services are what developers get paid to create!

What/Who

- Timothy Budd:
  - *"Why begin the design process with an analysis of behavior? The simple answer is that the behavior of a system is usually understood long before any other aspects."*

- *What / Who cycle*
  - **What:** identify behavior / responsibility ⇨ roles
  - **Who:** identify objects that may play the roles
    - or even invent objects to serve roles only
      - Larman "Pure fabrication";

Implications

- Responsibility perspective:
  - A) Analyze behavior (what?)
  - B) Assign objects (who?)
- Guidelines:
  - A) Behavior abstracted ⇨ landscape of *responsibilities*
  - B) Implement responsibilities in objects
- Analysis
  - Resemble human organizations – often roles are invented
  - Still need to define the objects ☺
    - That is, the person(s) to fill the role

AARHUS UNIVERSITET

The Central Concepts

A strong mind-set for
designing flexible software

“Theory of Compositional Designs”

AARHUS UNIVERSITET

How people organize work!

- The central concepts:
  - **Behavior:** *What actually is being done*
    - ”Henrik sits Sunday morning and writes these slides”
  - **Responsibility:** *Being accountable for answering request*
    - ”Henrik is responsible for teaching responsibility-centric design”
  - **Role:** *A function/part performed in particular process*
    - ”Henrik is the course teacher of SWEA”
  - **Protocol:** *Convention detailing the expected sequence of interactions by a set of roles*
    - ”Teacher: ‘Welcome’ => Students: stops talking and starts listening”
    - Student asks question; teacher is expected to answer

AARHUS UNIVERSITET

It is all Roles and Protocol

- Any complex human organization relies completely on each person understanding roles and protocols
  - If I get hospitalized, I understand the roles of patient, nurses, and physicians
  - CEOs, managers, software developers, architects, testers, sales people, …
  - Hardship of marriage: finding the proper roles and protocols ☺
    - Role models?

AARHUS UNIVERSITET

Roles decouples

- The primary point of roles:
  - ***It provides a higher abstraction than that of the individual person***

- I know my responsibilities and the protocol once I am assigned a known role
  - Teacher role defines what my responsibilities are
- I can collaborate efficiently with others once I know their roles
  - Student role defines what I can expect them to do

AARHUS UNIVERSITET

Many-to-many relation

- Big company
  - One person is manager, one software architect, two lead developers, and ten software developers
- Small company
  - Same person is manager, software architect, lead and software programmer ☺
- That is: **One individual may serve many roles**
- *Henrik: Teacher, researcher, tax payer, company owner, tourist, father, husband, …*

Interface Segregation Principle

AARHUS UNIVERSITET

Many-to-many relation

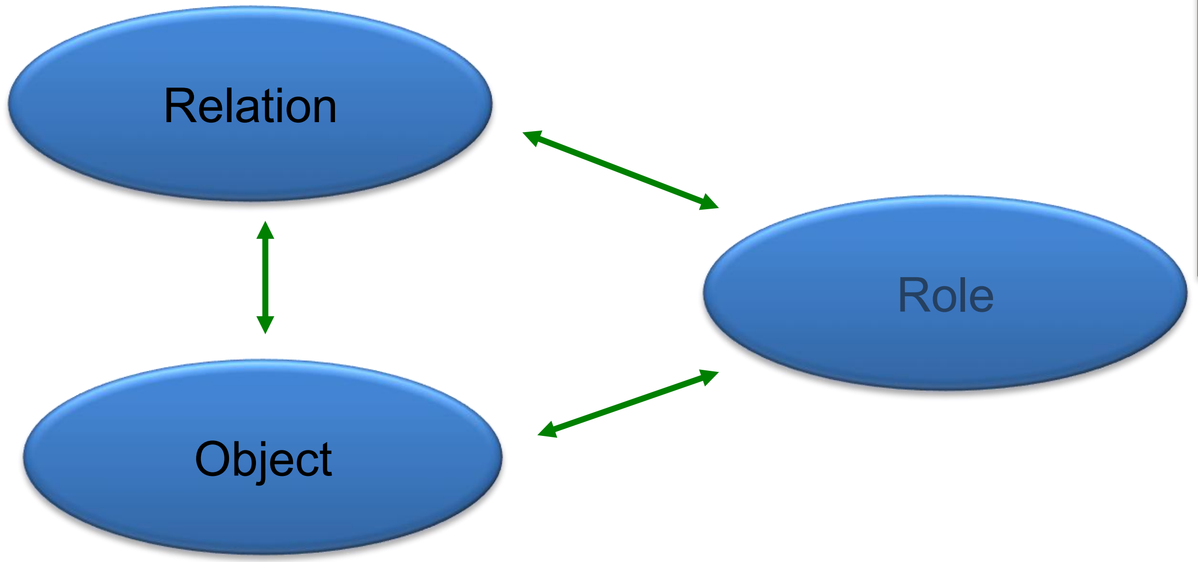
- Hospital
  - Nurses attend the patients
  - And different persons serve the role during shifts

- That is: **One role may be served by many persons**

Substitution Principle

Role concept

- The role concept allows us to use *either* approach (who/what or what/who) because “what” can be expressed as roles.



Role makes service a *first-class citizen* of our design vocabulary

Roles may be invented

- Roles may be invented by need.

- A pre-school kindergarten invented a ***Flyer*** role whose responsibility it was to ‘catch’ all interruptions to make the daily work more fluent for the ‘non-flyer’ pedagogues.

AARHUS UNIVERSITET

Enough Academic B…….

What should I do when designing???

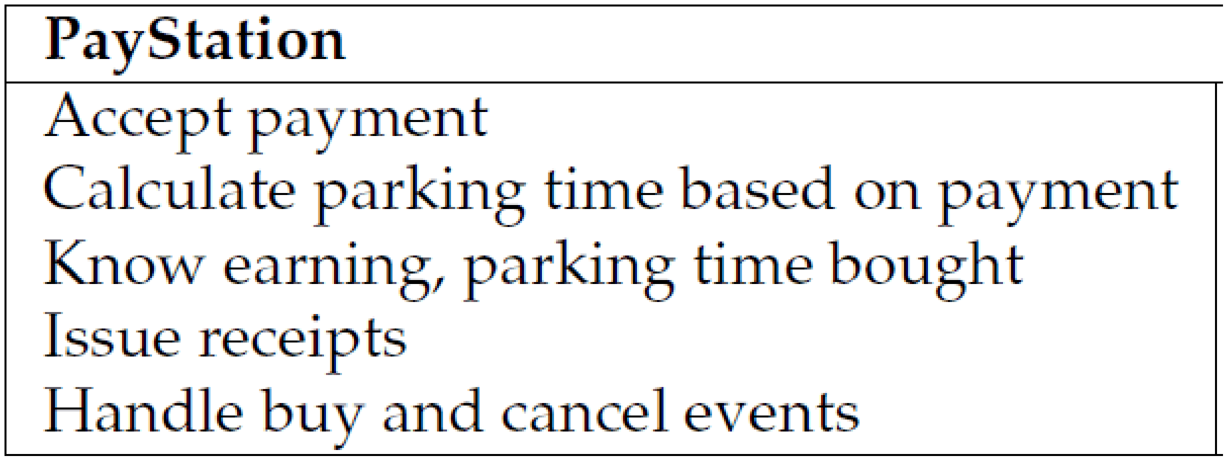
Software as Organizations

- The proposal:
  - Think software design in terms of
    - The **responsibilities** to be served
    - Group then into **cohesive roles**
    - And define their **protocols**, how are they going to collaborate

- That is:
    - **Design software as an Organization**

Super simple example

- The Pay station



PayStation

Accept payment
Calculate parking time based on payment
Know earning, parking time bought
Issue receipts
Handle buy and cancel events

- Now, one responsibility has been put into another role: the RateStrategy.
  - And different objects may play that role…

Another Example

- **HotStone**
  - Game (= manager/coordinator)
    - Role: Is responsible for overall game mechanics
      - Card handling, hand, battlefield, attacks, turn taking, rules enforcer…
    - Collaborates with lots of other roles
  - Hero, Card (= specialists)
    - Role: Primary *state holders* + simple, local, *state changes*
      - Owner, health, mana, …
  - WinnerStrategy (= super-specialist ☺)
    - Role: Is responsible for calculating who has won
      - Access information from other roles to do the calculation
  - DeckBuildingStrategy
    - Role: Is responsible for creating a deck
  - ect.

AARHUS UNIVERSITET

Yet Another Example

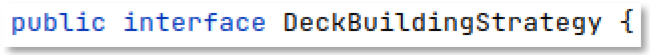
- **SkyCave**
  - Massive multiplayer on-line exploration experience
- (Some of the many) Roles:
  - Cave, Player, Room
    - Domain abstractions
      - *Player* with name may move in *rooms* in *cave*, and create new rooms to share with other players
  - Broker
    - Responsible for remote method calls (actually 6 roles!)
  - CaveStorage
    - Responsible for persisting rooms and players
  - SubscriptionService
    - Responsible for authenticating player login

MicroService paradigm!

AARHUS UNIVERSITET

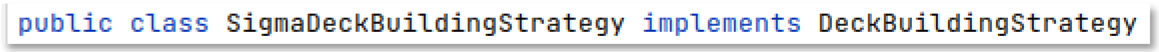
Programming Mechanics

- Use **interface** to define a **role**



```
public interface DeckBuildingStrategy {
```

  - Methods embody the **responsibilities**
  - (the **protocol** must be understood in the design)
    - Still lack programming constructs to describe these ☹
- Classes *implementing* an interface allow objects to be instantiated *to serve the roles*

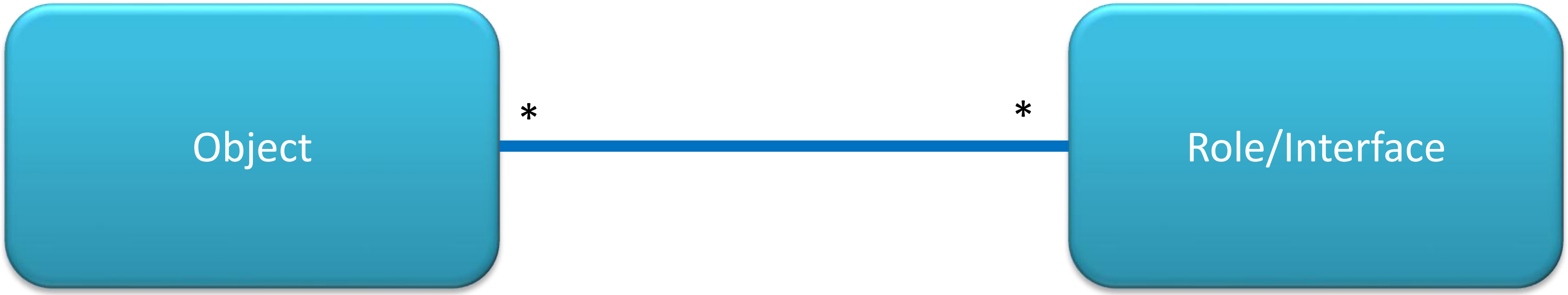


```
public class SigmaDeckBuildingStrategy implements DeckBuildingStrategy
```

- (Simple roles with no need for variability – just use a class)
  - Typical example is ”records” = dump data containers
    - Java 17 directly has a ‘record’ type (at last…)

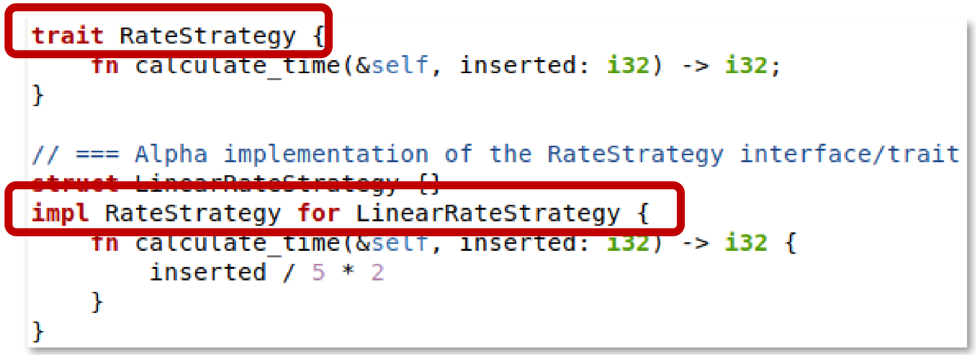
A many-to-many relation

- Interface – Object is a *many-to-many* relation
  - One role/interface implemented by many objects
    - Interface Game has been implemented in 70+ ways in your work
  - One object may implement many interfaces
    - StandardGame is both a ‘Game’ (UI uses this) and an ‘InternalMutableGame’ (Strategies uses this)



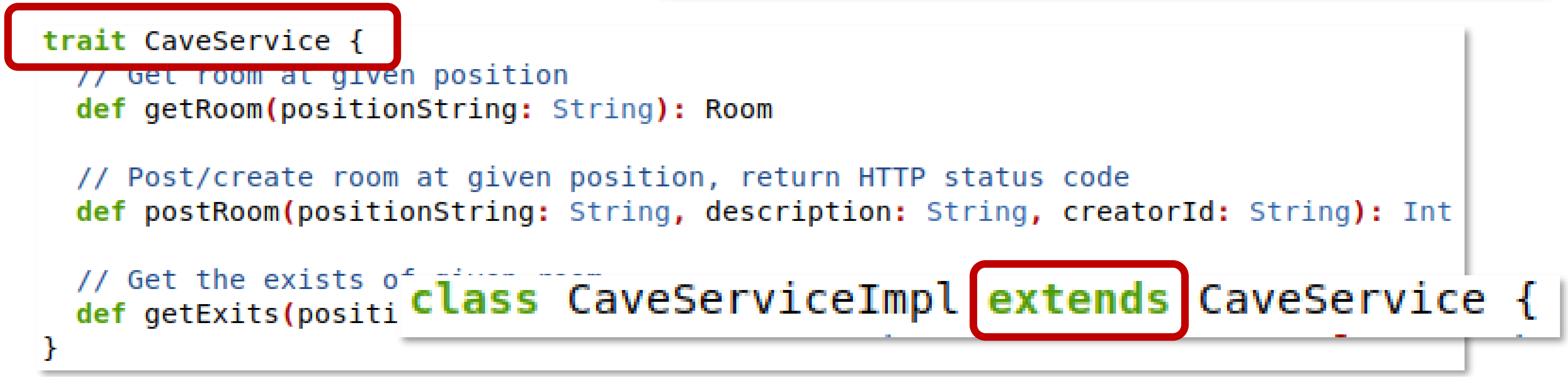
Language Support

- I find support for **interface** to define a **role** extremely important in a language!
- Rust supports **Traits**



```
trait RateStrategy {
    fn calculate_time(&self, inserted: i32) -> i32;
}

// === Alpha implementation of the RateStrategy interface/trait
struct LinearRateStrategy {}
impl RateStrategy for LinearRateStrategy {
    fn calculate_time(&self, inserted: i32) -> i32 {
        inserted / 5 * 2
    }
}
```

- Scala also has **Traits**



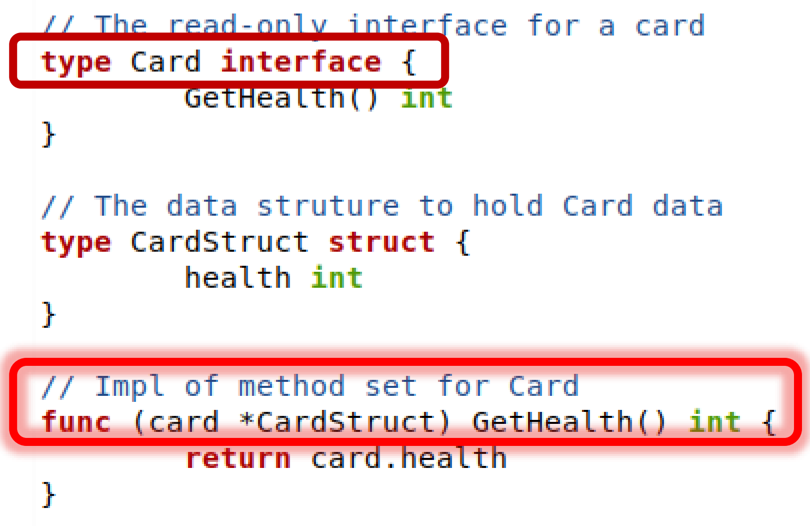
```
trait CaveService {
  // Get room at given position
  def getRoom(positionString: String): Room

  // Post/create room at given position, return HTTP status code
  def postRoom(positionString: String, description: String, creatorId: String): Int

  // Get the exists o
  def getExits(positi
}
```

```
class CaveServiceImpl extends CaveService {
```

AARHUS UNIVERSITET

Language Support

- Go has interface, but no way of expressing that a certain ‘object’ needs to implement it
  - *Duck typing*
  - *No way of expressing that CardStruct ‘implements Card’*



```go
// The read-only interface for a card
type Card interface {
        GetHealth() int
}

// The data struture to hold Card data
type CardStruct struct {
        health int
}

// Impl of method set for Card
func (card *CardStruct) GetHealth() int {
        return card.health
}
```

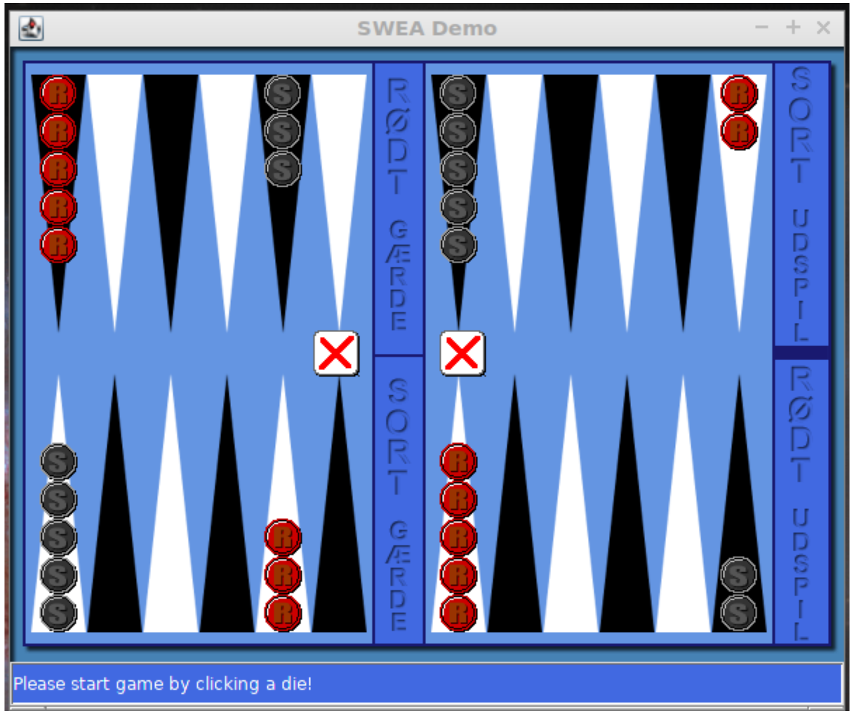
Morale:
Designing in Roles is a strong paradigm.
Some languages support it better…

An Example

Of identifying *roles* instead of *domain concepts*

HotGammon

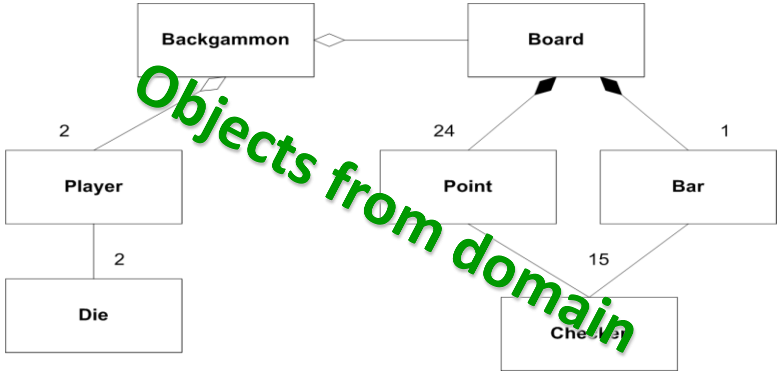
- Backgammon requirements:
  - Offer GUI for two players
  - Guaranty proper play
- Variants
  - *new rules* for which moves are legal
  - how many moves you can make per turn?
  - how the board is initially set up?



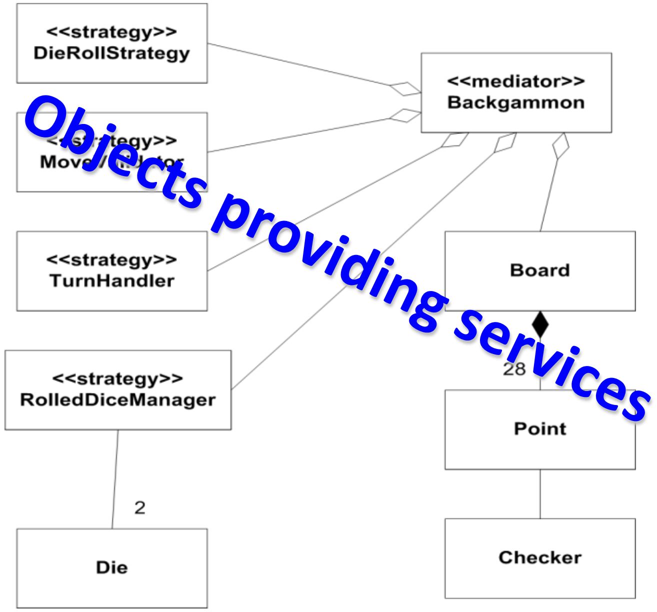
Please start game by clicking a die!

Same challenge – different designs

Model perspective:

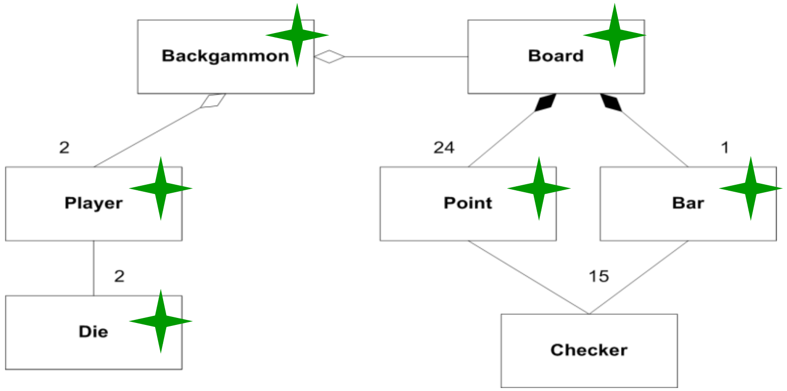


Responsibility perspective:

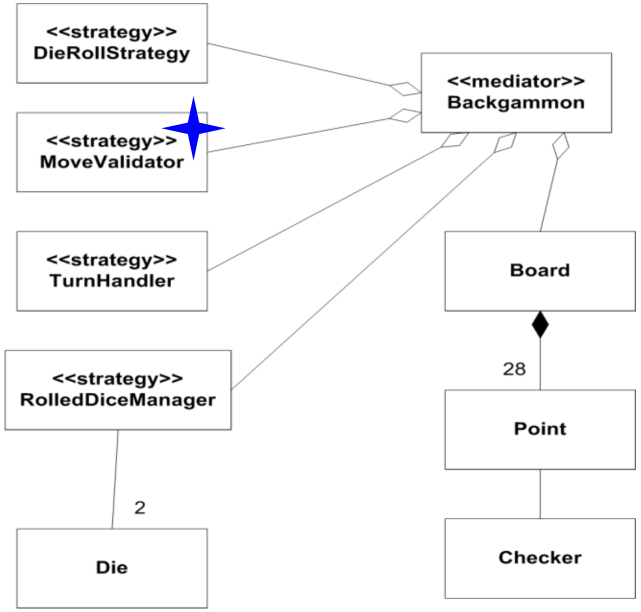

<<strategy>> DieRollStrategy
<<mediator>> Backgammon
<<strategy>> MoveValidator
<<strategy>> TurnHandler
Board
28
<<strategy>> RolledDiceManager
Point
2
Die
Checker
Objects providing services

Who is responsible for validating moves?

Model perspective:



Responsibility perspective:



<<strategy>> DieRollStrategy
<<mediator>> Backgammon
<<strategy>> MoveValidator
<<strategy>> TurnHandler
Board
<<strategy>> RolledDiceManager
28
Point
2
Die
Checker

What is the cost of altering *algorithm to compute if move is valid?*
How to change it at run-time?

AARHUS UNIVERSITET

Summary

AARHUS UNIVERSITET

Summary

- The central concepts:
  - **Behavior:** *What actually is being done*
    - ”Henrik sits Sunday morning and writes these slides”
  - **Responsibility:** *Being accountable for answering request*
    - ”Henrik is responsible for teaching responsibility-centric design”
  - **Role:** *A function/part performed in particular process*
    - ”Henrik is the course teacher”
  - **Protocol:** *Convention detailing the expected sequence of interactions by a set of roles*
    - ”Teacher: ‘Welcome’ => Students: stops talking and starts listening”

AARHUS UNIVERSITET

Perspectives

- Three different perspectives on OO
  - **Language**: Important because code is basically only understandable in this perspective
  - **Model**: Important because it gives us good inspiration for organizing the domain code
  - **Responsibility**: Important because it allows us to build highly flexible software with low coupling and high cohesion

- *They do not have to be in conflict – they build upon each other...*

Role Perspective

- Thinking in responsibilities grouped into roles is a strong design model
  - And it is not only relevant for Object-Oriented design thinking
- It works well in the imperative design world as well
  - As evident that Rust/Go and others have ‘interface’ constructs
- Regarding functional programming? Yes, why not
  - But I am no expert so…

AARHUS UNIVERSITET

Summary

- Design in terms of what roles and responsibilities there are in a system.

- Express these as **interfaces** with appropriate additional documentation.
  - Or ‘traits’ in some languages

- Implement the roles by concrete classes.

- Roles should *encapsulate points of variability*