The Toolbox
工具箱

Vertical and horizontal contracts in large systems

Rolf-Helge Pfeiffer - Anders Kalhauge

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Fall 2016

The goal of the day

- You all understand the toolbox as a sound alternative, somewhere between the extreme formalization in Design by Contract, and no formalization in natural language contracts.
- You will master the central elements in the toolbox.
- Understand and can use UML artifacts to define contracts between development groups working at the same level.
  - Vertical contracts
  - Horizontal contracts

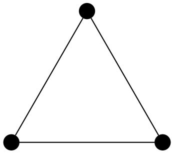
Agenda

- Presentation of the details in a design contract
    - Table of content (template)
    - Evaluation criteria
- Introduction to the toolbox as a practical example to contract based software development
    - focus is on vertical contracts: front-end $\longleftrightarrow$ back-end
- Presentation of the toolbox
    - we might skip some of the slides ☺

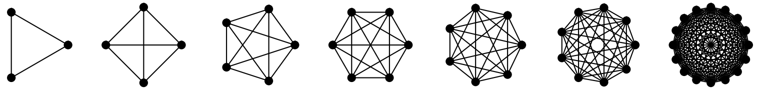
Complexity

IF MOUSETRAPS WERE DESIGNED LIKE THE INCOME TAX CODE...
JET FUEL
FORM A
IN
UNIT
ENTER
INTAKE FILTER
FIELD MOUSE EXEMPTION
DESIGNED BY CONGRESS
RATS → ← MICE
EAT CHEESE BY APRIL 15TH
UNIT
SLOT B
?
The Clarion-Ledger
MARSHALL RAMSEY

Complexity



Complexity



$n = 3$	$n = 4$	$n = 5$	$n = 6$	$n = 7$	$n = 9$	$n = 16$
$c = 3$	$c = 6$	$c = 10$	$c = 15$	$c = 21$	$c = 36$	$c = 120$

$$(n-1) + (n-2) + \ldots + 2 + 1 = \frac{(n-1)n}{2} = O(n^2)$$

Complexity
Philippos II of Macedon - 382–336 BC

$$\delta\iota\alpha\iota\rho\epsilon\iota\ \kappa\alpha\iota\ \beta\alpha\sigma\iota\lambda\epsilon\upsilon\epsilon$$

Complexity
Philippos II of Macedon - 382–336 BC

$\delta\iota\alpha\iota\rho\epsilon\iota\ \kappa\alpha\iota\ \beta\alpha\sigma\iota\lambda\epsilon\upsilon\epsilon$

- diaírei kaì basíleue

Complexity
Philippos II of Macedon - 382–336 BC

διαιρει και βασιλευε

- diaírei kaì basíleue
- divide et impera

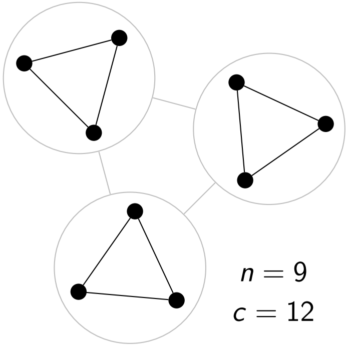
διαιρει και βασιλευε

- diaírei kaì basíleue
- divide et impera
- divide and rule

$$\delta\iota\alpha\iota\rho\epsilon\iota\ \kappa\alpha\iota\ \beta\alpha\sigma\iota\lambda\epsilon\upsilon\epsilon$$

- diaírei kaì basíleue
- divide et impera
- divide and rule
- del og hersk

Complexity

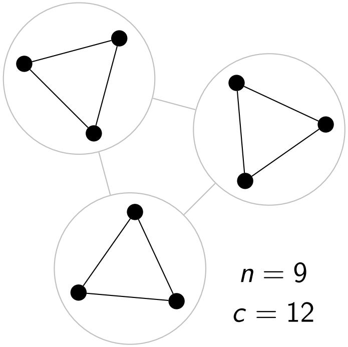


$n = 9$

$c = 12$

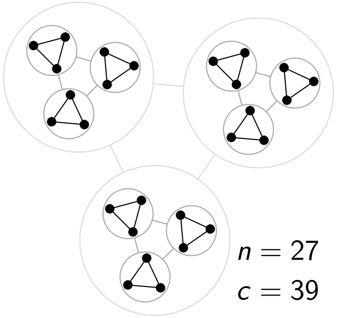
$$(\sqrt{n}+1)\frac{(\sqrt{n}-1)\sqrt{n}}{2} = \frac{(n-1)\sqrt{n}}{2} = O(n\sqrt{n})$$

Complexity



$n = 9$

$c = 12$



$n = 27$

$c = 39$

$O(n)$

Divide and conquor II

Overview

What's in the box?

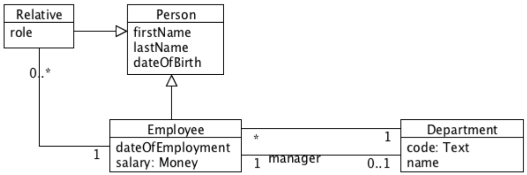
Overview
What's in the box?

- Logical data model
- Use case model
    - Use case diagram(s)
    - Use case descriptions
    - System sequence diagram
    - System operation contracts
- Communication model
    - System operation contracts
    - Transfer objects
        - Data Transfer Objects (DTOs)
        - Exception Transfer Objects (ETOs)
- Verification strategy

Logical data model
What is a logical data model?

- It models the system state.
- Expresses valid pre- and postcondition states.
- Expresses possible system state changes.

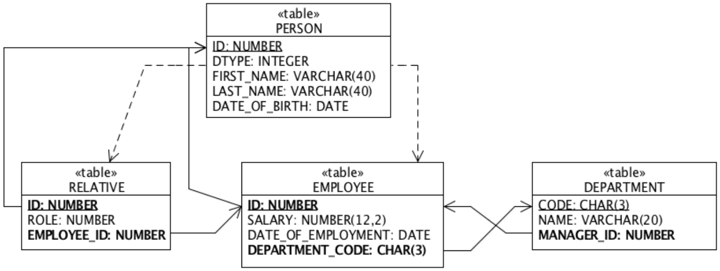
Logical data model
An example



- What should be persisted
- Only entities
- No implementation details
  - No ids unless they contain data (not necessarily wise)
  - Only abstract types

Logical data model

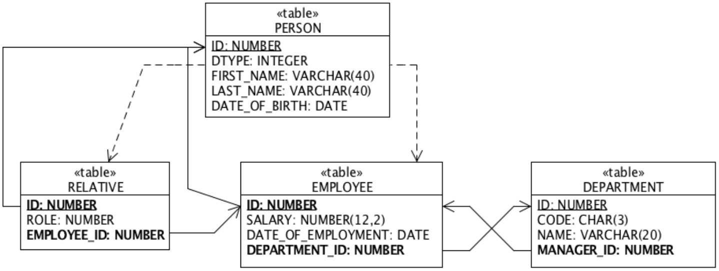
A relational implementation I



- Primary (<u>underlined</u>) and foreign (**boldfaced**) keys shown.
- Joined tables inheritance strategy, `DTYPE` discriminates between types.

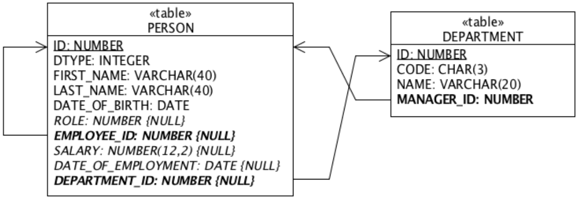
Logical data model

A relational implementation II



- Same as I, with no data bearing primary keys ☺

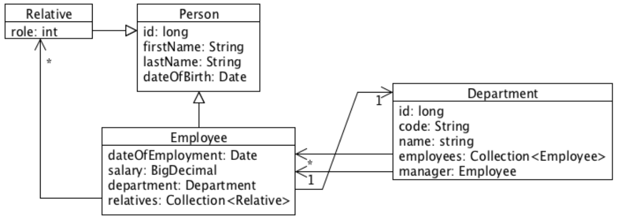
Logical data model
A relational implementation III



- Single table inheritance strategy
- “Irrelevant fields are nulled

Logical data model

An implementation with objects



- No associations, only references
- Id's to support "Object Relational Mapping"

User Stories

A user story describes functionality that will be valuable to either a user or purchaser of a system or software.

- User stories are written by users.
- User stories focuses on details.
- User stories cannot be guaranteed to be complete.
- User stories does not define system boundaries.

Use Cases

A use case is a generalised description of a set of interactions between the system and one or more actors, where an actor is either a user or another system.

- Use cases are written by developers.
- A use case model will describe the complete (sub)system.
- A use case defines the system boundary.
- Any use case scenario will bring the system from one consistent state to another.

Use Case Template

Alistair Cockburn has proposed a template for use cases, which can be found at:

`http://alistair.cockburn.us/Basic+use+case+template`

The template is used in the example later in this presentation.

Use Cases

Use cases can be at different detail levels:

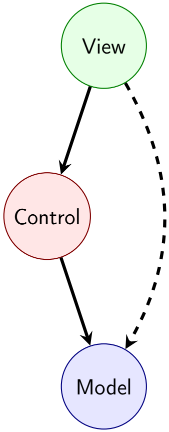
- Brief use case
  The use case is merely more than the title of an action
- Casual use case
  Typically casual use cases are described in more detail
- Fully dressed use case
  Includes all entries from the template

Actors

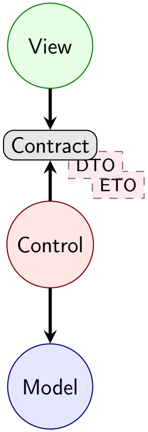
Actors are outside the system. For this reason, it is important to describe their responsibilities in detail. Actor resposibilities are not and should not be covered by the system.
Actors are not concrete people or systems, rather abstract roles of these users or systems.

Communication model

System operation contracts - MVC pattern reviewed



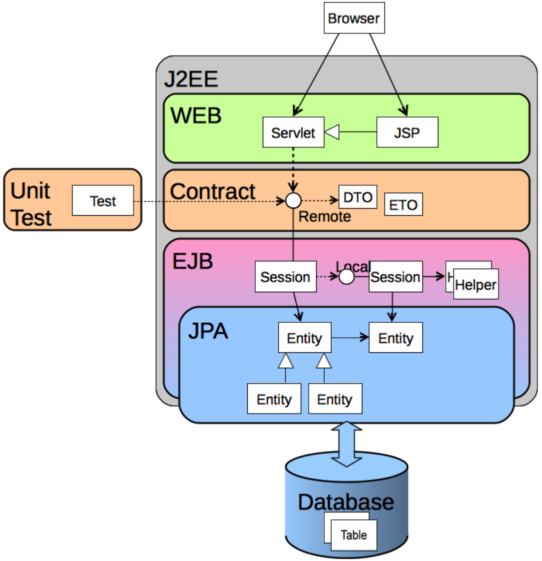
(*a*)



(*b*)

Communication model

System operation contracts - Layers in EJB



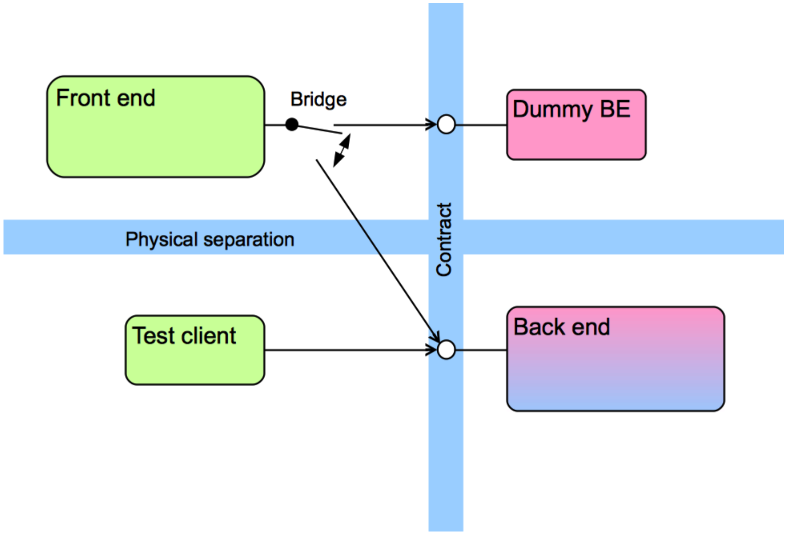
GUI design

Test cases

Use cases

Domain model

Communication model
Project setup with bridge



Communication model

System operation contracts - “Remote” interface

The interface is the code based operation contract.

- Use strong typing.
  - use DTOs instead of simple data types.
- Make inline documentation (JavaDoc)
  - have documentation close to code - easier to update.
  - generates written code contracts.
- Implement the interface with a Remote facade in the “backend”.
  - Changes to the backend code or to the interface will have less impact.
- Reference the interface from a Factory in the “frontend”.
  - Change of backend can be done with practically no code changes in “frontend”

Communication model
System operation contracts - Data Transfer Objects

Data transfer objects should be as abstract as possible when still being concrete. Use DTOs for request and return values.

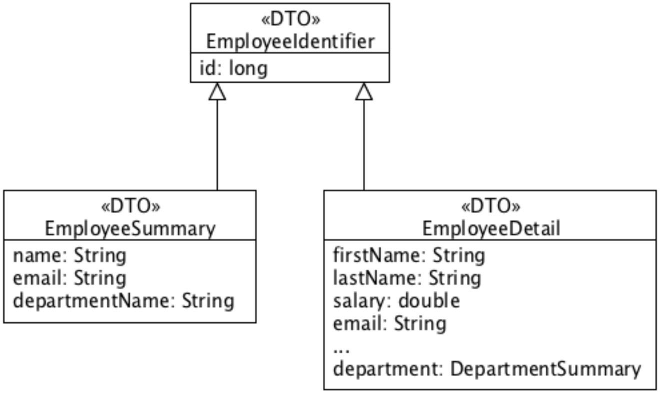
- Efficiency
  - Packing related data together
  - reducing calls - network calls are expensive to establish
  - reducing data - bandwidth is still an issue
- Encapsulation
  - by hiding irrelevant or secret data
  - **by hiding actual implementation**
- Serializable

Communication model
System operation contracts - Exception Transfer Objects

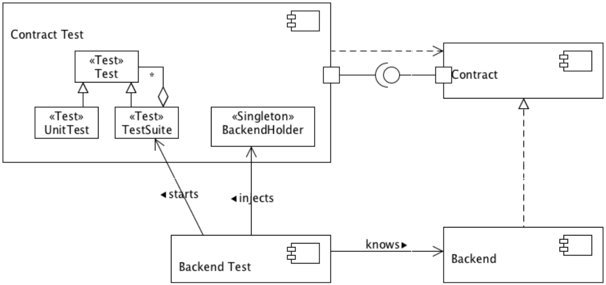
Exceptions are as valid, even less happy, return values from operations.

- User friendly - return only relevant information.
  - Preconditions: What precondition was violated (unchecked).
  - Postconditions: What went wrong (checked).
- Encapsulation
  - by hiding actual implementation
  - **revealing errors and their precise cause, is pleasing hackers**
- Serializable - in Java Exceptions are already Serializable

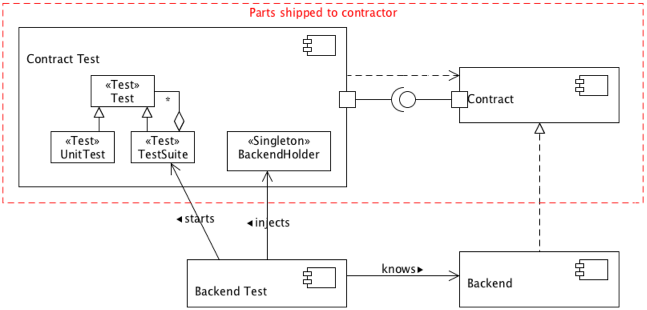
Communication model
System operation contracts - Data Transfer objects



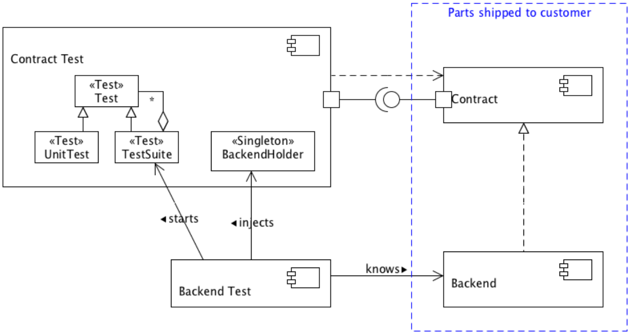
Verification



Verification



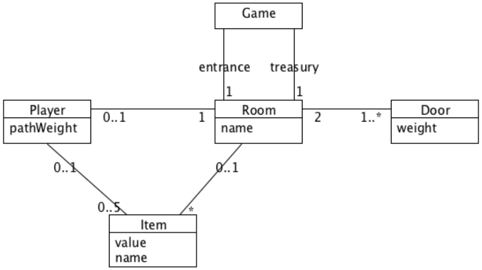
Verification



Dungeon game example
Requirements

We want a Dungeon game. The scenario of the game is a number of connected rooms or dungeons in a mountain. The player enters the mountain from the entrance room, and he/she should travel from dungeon to dungeon until he/she reaches the treasury room. All dungeon has doors that leads to at least one other dungeon. A dungeon can contain an unlimited number of items. Items have values. When a player is in the room he/she can see the items in the room, and he/she can see the doors leading from the room to other dungeons. The player can pick up and lay down items when he/she is in a room. But he/she can keep at most five items at a time. The quest is to reach the treasury room with the most expensive items through the shortest path. The game should run on a central server, and played throug a mobile phone connected to the server.

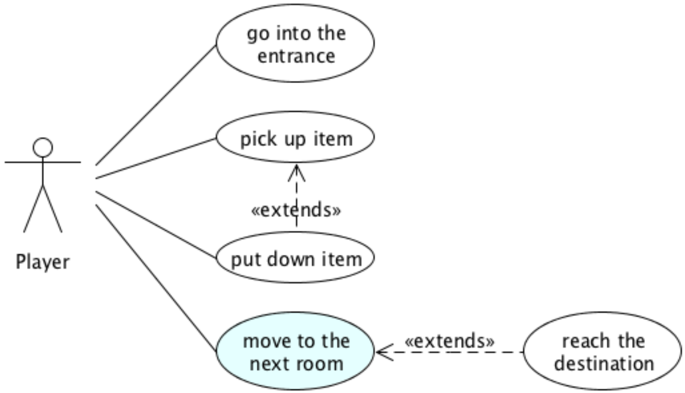
Dungeon game example
Logical data model



Again:

- Nouns from the requirements (glosary) are candidates
- What should be persisted
- Only entities
- No implementation details

Dungeon game example
Use Case Model - Use Case Diagram



Dungeon game example
Use Case Model - Detailed Use Case Description. . .

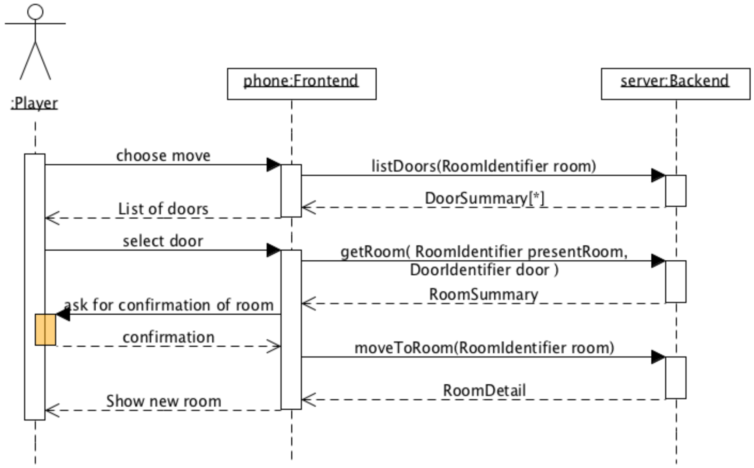
- **Name** Move to the next room
- **Scope** System under design (SuD)
- **Level** Goal: Move to the next room
- **Primary Actor** Player
- **Precondition** The player is in a room
- **Main succes scenario** . . .
- **Success guaratees** The player is in a new room
- **Extensions** Reach the destination if room is treasury room
- **Special Requirements** NONE

- **Name** Move to the next room

  . . .
- **Main succes scenario**
  1. Player chooses “move”
  2. System shows a list with all doors to other rooms
  3. Player selects the door he/she wants to move through
  4. System shows the room name, and asks the player to confirm
  5. Player confirms the selection of door
  6. System moves the player to the room behind the selected door

  . . .

Dungeon game example
Use Case Model - System Sequence Diagram

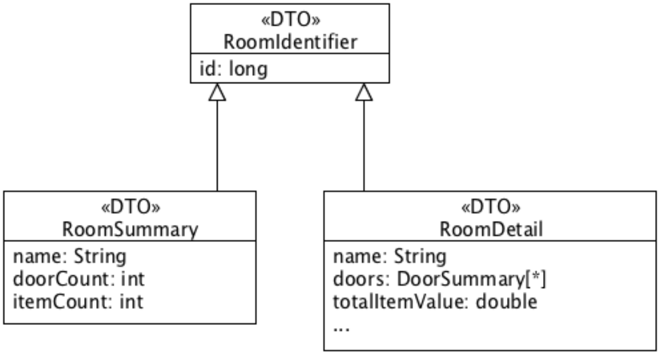


:Player
phone:Frontend
server:Backend
choose move
listDoors(RoomIdentifier room)
DoorSummary[*]
List of doors
select door
getRoom(RoomIdentifier presentRoom, DoorIdentifier door)
RoomSummary
ask for confirmation of room
confirmation
moveToRoom(RoomIdentifier room)
RoomDetail
Show new room

Dungeon game example
Communication model - "Remote" interface

```
@Remote
public interface DungeonManager {
  ...
  /**
   * List the doors leading from a given room.
   * @pre the room cannot be null and must exist.
   *   @throws NoSuchRoomException room doesn't exist.
   *   @param room the given room.
   * @post the doors in the given room is returned
   *   @return A collection of door summaries.
   */
  Collection<DoorSummary> listDoors(
      RoomIdentifier room
      );
  RoomSummary getRoom(
      RoomIdentifier room, DoorIdentifier door
      );
  RoomDetail moveToRoom(RoomIdentifier room);
  }
```

Dungeon game example
Communication model - Data Transfer objects



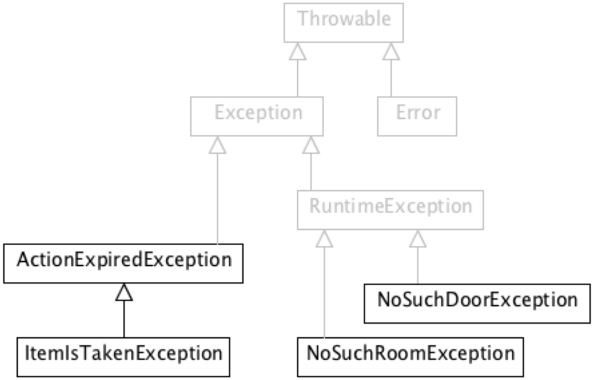
Dungeon game example
Communication model - Data Transfer objects

```
public class RoomIdentifier implements Serializable {
  private long id;

  public RoomIdentifier(long id) {
    this.id = id;
    }

  public long getId() { return id; }
  }
```

Dungeon game example
Communication model - Data Transfer objects

```
public class RoomSummary extends RoomIdentifier {
  private String name;
  private int doorCount;
  private int itemCount;

  public RoomSummary(
      long id, String name,
      int doorCount, int itemCount
      ) {
    super(id);
    this.name = name;
    this.doorCount = doorCount;
    this.itemCount = itemCount;
    }

  public long getName() { return name; }
  public long getDoorCount() { return doorCount; }
  public long getItemCount() { return itemCount; }
  }
```

Dungeon game example
Communication model - Exception Transfer objects



Dungeon game example
Communication model - Exception Transfer objects

```
public class ActionExpiredException
                extends Exception {

    public ActionExpiredException(String message) {
        super(message);
        }

    }
```

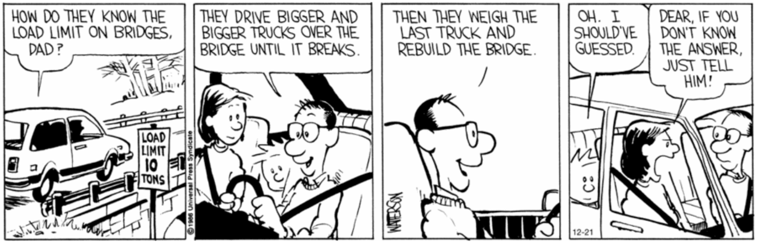
```
public class NoSuchRoomException
                extends RuntimeException {

    public NoSuchRoomException(String message) {
        super(message);
        }

    }
```

Contract Properties
The good contract

- Complete
  - Everything is in the contract
- Consistent
  - The are no conflicts in the contract
- Unambiguous
  - There is no room for interpretation
- Correct
  - It expresses our intentions

Contract Properties
Sound principles

- Traceability
  - One should be able to refind terms and descriptions in the actual implementation of the contract
- Encapsulation
  - Systems should not be accessible outside the scope of the contract, great risk of violating invariants.
- Testability
  - If a contract can't be verified, it has little value


HOW DO THEY KNOW THE LOAD LIMIT ON BRIDGES, DAD?
LOAD LIMIT 10 TONS
THEY DRIVE BIGGER AND BIGGER TRUCKS OVER THE BRIDGE UNTIL IT BREAKS.
THEN THEY WEIGH THE LAST TRUCK AND REBUILD THE BRIDGE.
OH. I SHOULD'VE GUESSED.
DEAR, IF YOU DON'T KNOW THE ANSWER, JUST TELL HIM!
© 1986 Universal Press Syndicate
WATTERSON
12-21

Toolbox - first delivery

- **Wednesday September 14th before 24:00** Create a toolbox contract for the Ferry case. Only important use cases in the contract, less important use cases in the use case diagram.
- **Friday September 16th before 08:00** Make a review of the contract you have received. Is it contract quality?