NÁVRHOVÉ VZORY

ÚVOD

JAVA ZOOLOGY

Java Standard Edition – Java SE

- Basic types, objects, classes, networking, security,
- Database access, XML parsing, user interfaces

Java Enterprise Edition – Java EE

- Large scale, multi-tier, scalable, reliable apps, components

BUILDING AN APP

How?

ABSTRACT DATA TYPE

Abstract Data Type

- Mathematical model for data types
- Stack `(push, top, pop)`
- `new Stack / create() –` instantiation

ABSTRACT DATA TYPE

```
typedef struct stack_Rep stack_Rep;    // type: stack instance representation
typedef stack_Rep* stack_T;            // type: handle to a stack instance
typedef void* stack_Item               // type: value stored in stack instance

stack_T stack_create(void);            // creates a new empty stack instance
void stack_push(stack_T s, stack_Item x); // adds an item at the top
stack_Item stack_pop(stack_T s);       // removes the top item  and returns it
bool stack_empty(stack_T s);           // checks whether stack is empty
```

BUILDING LARGE APP
ABSTRACT DATA TYPE

```
#include <stack.h>                   // includes the stack interface


stack_T s = stack_create();   // creates a new empty stack instance

int x = 17;

stack_push(s, &x);                   // adds the address of x at the top of the stack

void* y = stack_pop(s);   // removes the address of x from the stack and returns it

if(stack_empty(s)) { }               // does something if stack is empty
```

OBJECT-ORIENTED PROGRAMMING

"objects” may contain

- data in the form of fields, often known as attributes;
- code, in the form of procedures, often known as methods.
- A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this”).
- In OOP, computer programs are designed by making them out of objects that interact with one another.

OBJECT-ORIENTED PROGRAMMING

Base properties

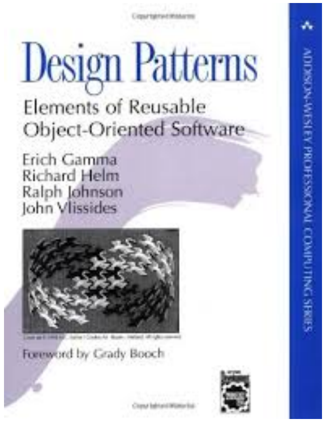
- Dynamic dispatch – method lookup – dynamic (ADT static)
- Encapsulation
- Composition / inheritance /delegation
  - Composition - Employee contains address object
  - Inheritance – hierarchy Person - Employee
  - Delegation – alternative to inheritance one entity passing something to another
- Polymorphism
  - Enables separation of concerns (SoC)
- Recursion

- History – SmallTalk 1970

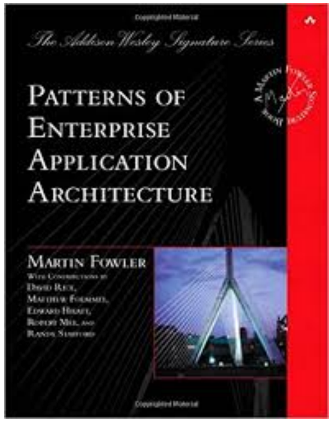
OBJECT-ORIENTED PROGRAMMING

Best practice of

composition / inheritance / delegation / encapsulation / polymorphism / ..



Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



The Addison Wesley Signature Series
PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE
MARTIN FOWLER

- Design Patterns
  - Erich Gamma
  - Martin Fowler

  - Predefined solutions to typical programmer problems
  - Building blocks for Software Engineers!

DEF: (SOC) SEPARATION OF CONCERNS

Dijkstra in 1974

The design principle for separating a computer program into distinct sections, such that each section addresses a separate concern.

A concern is a set of information that affects the code of a computer program.

A concern can be as general as the details of the hardware the code is being optimized for, or as specific as the name of a class to instantiate.

SEPARATION OF CONCERNS

..

A program that embodies SoC well is called a modular program.

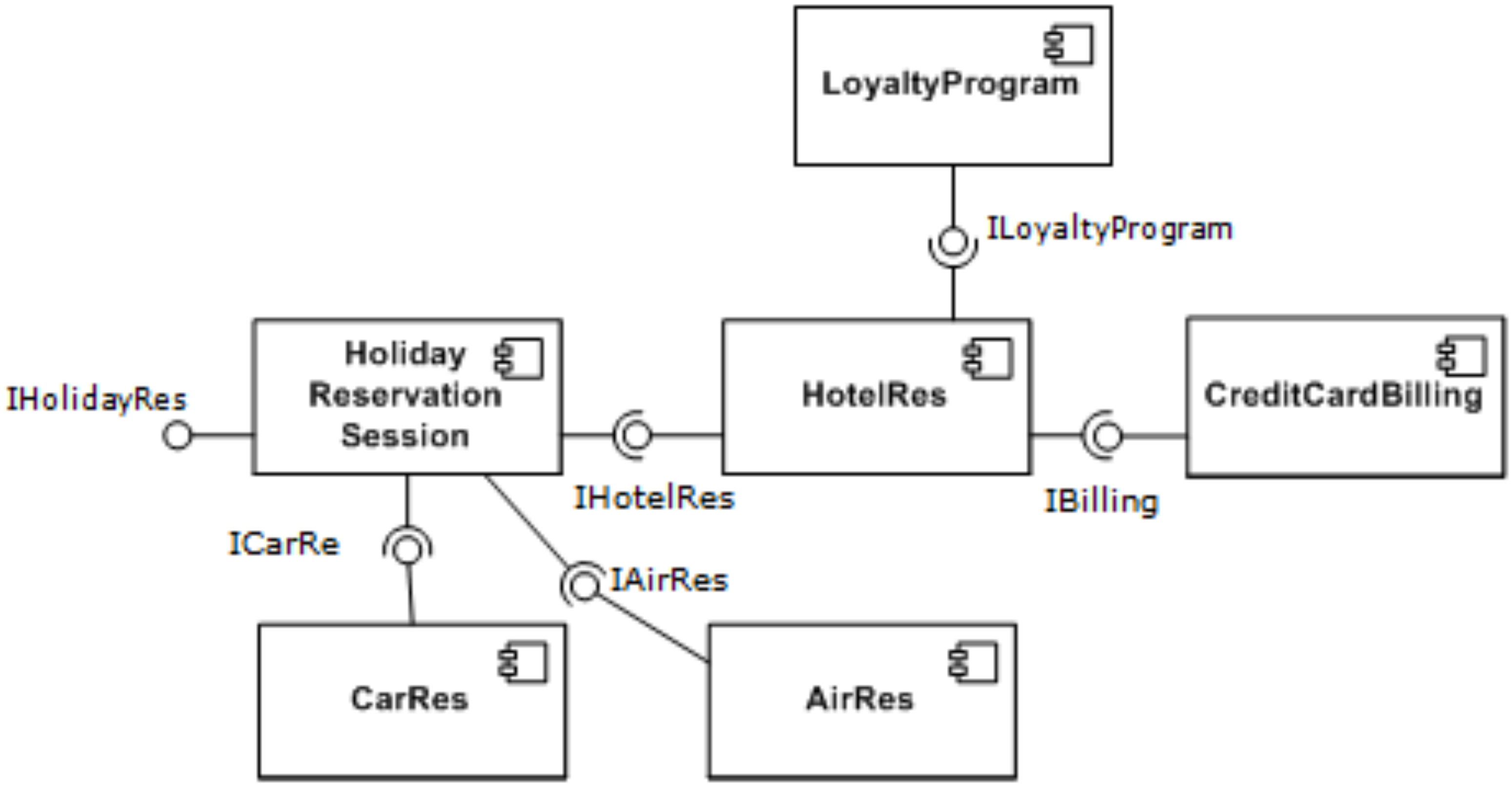
Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a well-defined interface.

Encapsulation is a means of information hiding.

Layered designs in information systems are another embodiment of separation of concerns

- (e.g., presentation layer, business logic layer, data access layer, persistence layer)

BUILDING LARGE APP
COMPONENT BASED DEVELOPMENT



BUILDING LARGE APP COMPONENT BASED DEVELOPMENT

- Emphasizes the separation of concerns
- Reuse-based approach to defining, implementing and composing loosely coupled independent components into systems.

The notion of component

- An individual software component is a software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data).
- All system processes are placed into separate components

BUILDING LARGE APP COMPONENT BASED DEVELOPMENT

- Components can produce or consume events and can be used for event-driven architectures
- In web services, and more recently, in service-oriented architectures (SOA), a component is converted by the web service into a service and subsequently inherits further characteristics beyond that of an ordinary component.

OBJECT VS. COMPONENT

Component not language specific

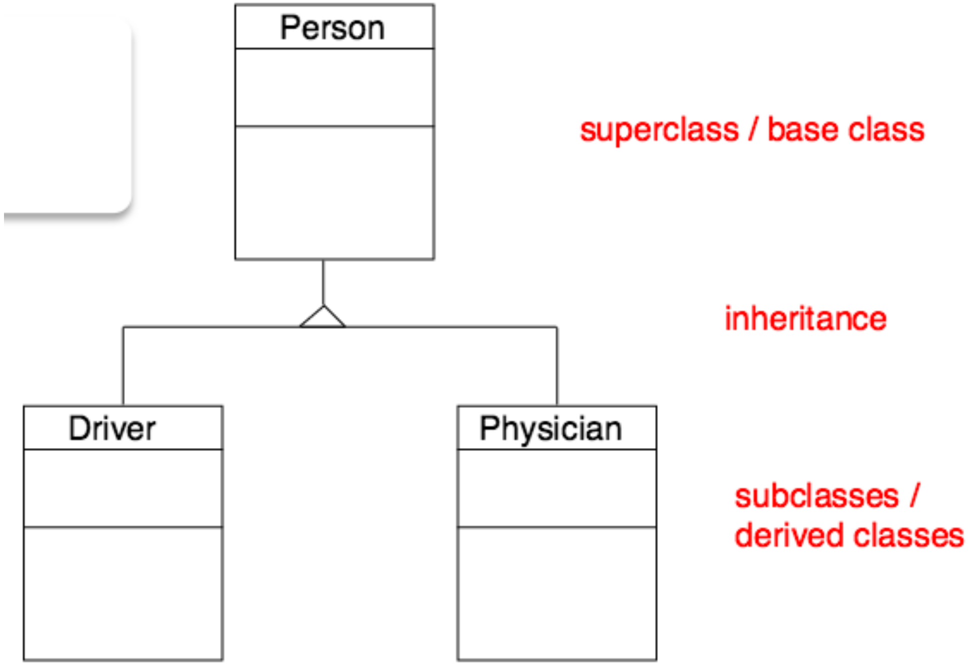
- Organization unit, building block, functional element.
- Comparison
  - An object is a component
  - Collection of objects is a component

Components connect together, and usually have dependencies, although we think of a component as an independent functional block**.**

- e.g. OSGi standard – automobiles and industry automation

Component has usually specification and realization (Interfaces and implementation in the Object-based design)

OBJECT VS. COMPONENT

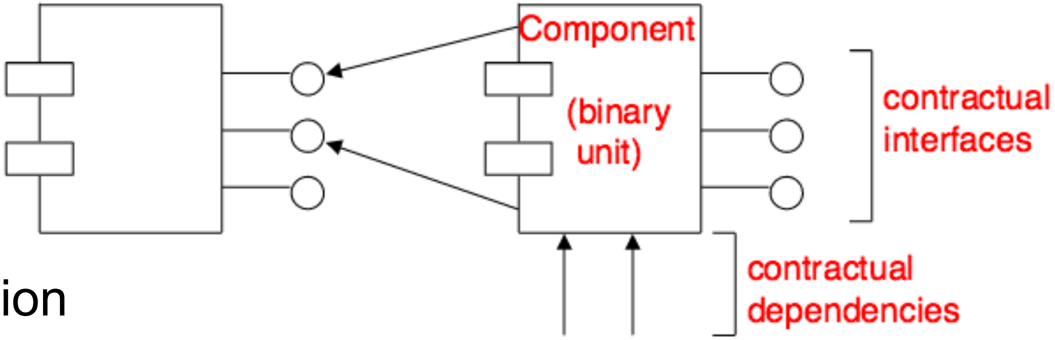


Object-based design

– construct app from objects

Component-based design

– construct app from preexisting service-providing components



Properties:

- Encapsulation
- Specification – interface
- Improved reuse and evolution
- Abstraction

VALUE OBJECT VS REFERENCE OBJECT

```
Person joe1 = getJoe();
Person joe2= getJoe();
joe1 == joe2
Person bob = getBob();
bob.born.equals(joe1.born)
```

Object-based design - objects have identity

- **Reference object – e.q. a Customer**
  - One object identifies a customer in the real world
  - Any reference to the customer is a pointer to the Customer objects!
  - Changes to the customer object available to all users!
  - Compare identity

- **Value Object -** a small object that represents a simple entity like Date, Money
  - Multiple value objects represent the same real world thing
  - Hundred of objects that represent Jun 5th, 2015
  - Comparing dates does not compare identify but the value!
- Its equality is not based on identity:
  - two value objects are equal when they have the same value,
  - not necessarily being the same object.

COMPONENT IN JAVA EE

Usually it is enough to use annotation

Java EE Container at an Application Server recognizes the component and applies life-cycle

```
@Entity
Class Person {
   String name;
   String getName() {
         return name;
   }
   void setName(String name) {
         this.name = name;
   }
}
```

```
@Stateless
Class PersonService {

   @Inject
   EntityManager em;

   void register(Person person) {
         person = em.persist(person);
   }
}
```