Engineering of Software Subsystems
Design Principles

![Diagram of component operation and concrete decorator operations](diagram.png)
The object-oriented paradigm is considered to be based on four principles.

- **Encapsulation**
  - *State or attributes*
  - *Behaviors that act on or use the encapsulated data*

- **Inheritance**

- **Polymorphism**

- **Identity**
  - *There are individually identifiable objects that are instantiations of a class.*
SOLID and GRASP provide two sets of object-oriented design principles.

**SOLID**
- Single responsibility
- Open/closed
- Liskov substitution
- Interface segregation
- Dependency inversion

**GRASP**
- Controller
- Creator
- Indirection
- Information expert
- High cohesion
- Low coupling
- Polymorphism
- Protected variations
- Pure fabrication

You should recall these 6 from the Object Oriented Design I unit in Intro to SE.
SOLID and GRASP provide two sets of object-oriented design principles.

- These are also principles from SOLID and GRASP

**SOLID**
- Single Responsibility
- Open/closed
- Liskov substitution
- Interface segregation
- Dependency inversion

**GRASP**
- Controller
- Creator
- Indirection
- Information expert
- High cohesion
- Low coupling
- Polymorphism
- Protected variations
- Pure fabrication

And these 5 from the Object Oriented Design II unit.
There are some other key object-oriented design concepts that we will stress.

- Decrease coupling where possible
- Prefer type (interface) inheritance over class (implementation) inheritance. “Program to the interface, not the implementation.”
- Prefer composition to inheritance: prefer “has-a” relationships to “is-a” relationships.
- Use delegation to “simulate” runtime inheritance.
More O-O Design Concepts

- **Abstraction**: provides well defined conceptual boundaries that focus on the outside view of an object and so serves to separate an object’s essential behavior from its implementation.

- **Principle of Least Commitment**: the interface of an object provides its essential behavior, and nothing more.

- **Principle of Least Astonishment**: an abstraction captures the entire behavior of an object and offers no surprises or side effects that go beyond the scope of the abstraction.

- Abstraction is one of the fundamental ways to deal with *complexity* and complexity’s close cousin – *change*. 
The domain model bridges the domain world and the implementation solution space.

**Application Domain**

What are the entities and associations within this domain?

**Domain Model**

What entities and associations are important for this application?

**Class Model**

What software structure and relationships will provide the implementation of this application?

The domain model influences the naming and structure in the software class model.

Use the language of the application and user! It does not define the implementation.
The domain model notation shows the domain entities and information about their relationships.

- Notational guidelines for domain models
  - *Only uses language from the domain, i.e. every domain user can fully understand it*
  - *Every relationship is labeled with a semantically significant relationship, i.e. not has, holds, contains*
  - *May include inheritance*
  - *May show entity attributes*
  - *May show multiplicities*