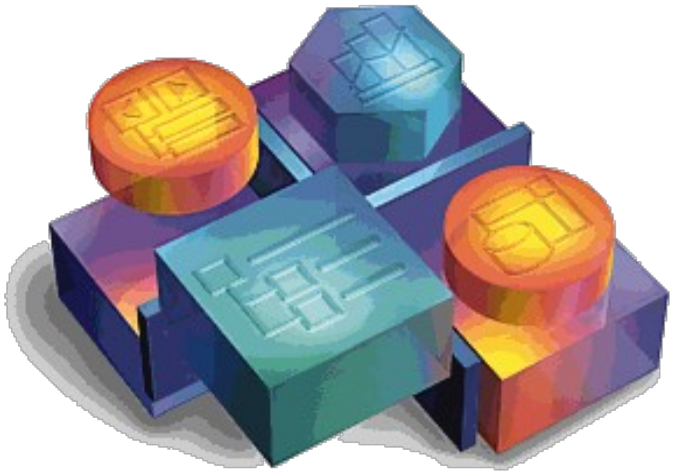
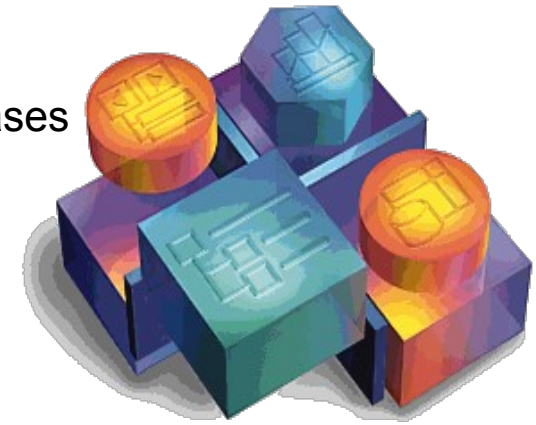
Object-Oriented Programming Design



1. Introduction to Object-Oriented Concepts
2. How to think in Terms of Objects
3. Advanced Object-Oriented Concepts
4. The anatomy of a Class
5. Class Design guidelines
6. Designing with objects
7. Mastering Inheritance and Composition
8. Frameworks and Reuse: Designing with interfaces and Abstract classes

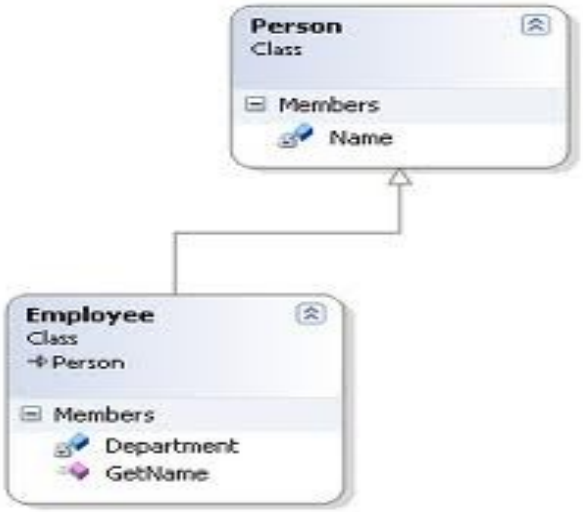
9. Building objects

10. Creating Object Models with UML
11. Objects and Portable Data: XML
12. Persistent Objects: Serialization and Relational Databases
13. Objects and the Internet
14. Objects and Client/Server Applications
15. Design Patterns

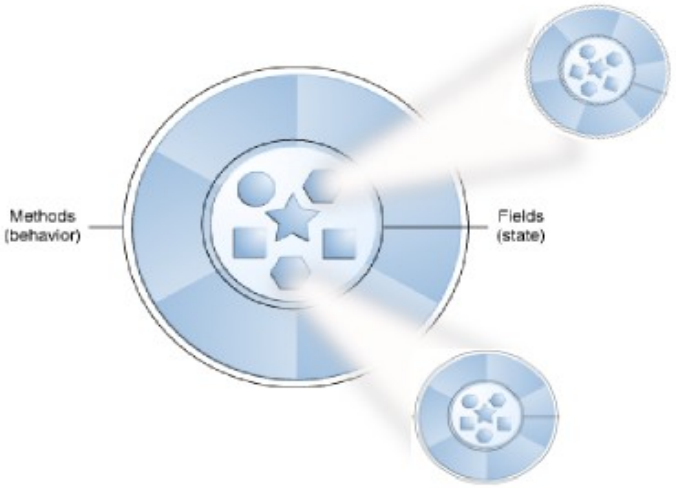


9. Building Objects

We learned that **inheritance** and **composition** represent the primary ways to build objects.



Inheritance



Composition

9. Building Objects

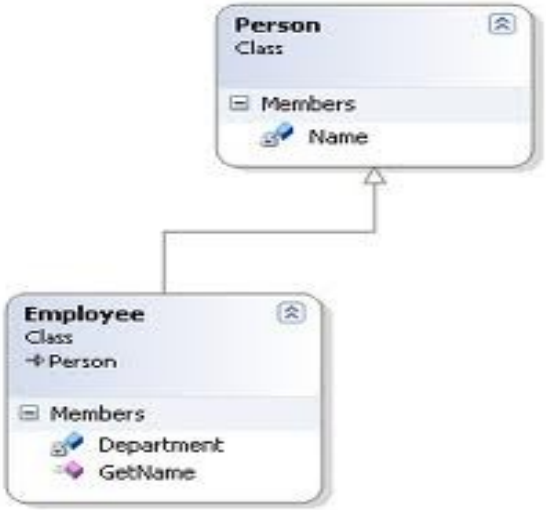
- We learned how **inheritance**, **interfaces**, **abstract classes**, and **composition** all fit together.

- We will see in more details the issue of how objects are related to each other in a overall design.

9. Building Objects

Inheritance Relationship

- Let's revisit the example of the Person and Employee Classes. Basically, an employee is a person. An **employee object** does not send a message to a **person object.** This is because an **employee object** is a p**erson object**.

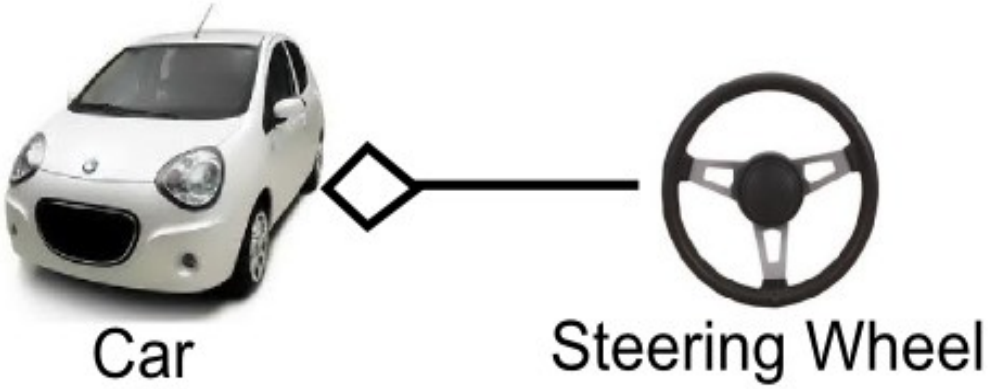


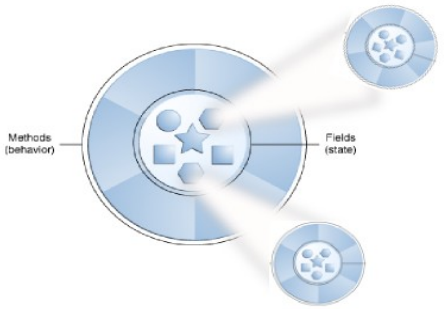
9. Building Objects

Composition Relationship

► **Composition represents interactions between distinct objects.**

► Composition is stated in terms of **has-a.** We know intuitively that a car « **has-a** » steering wheel.

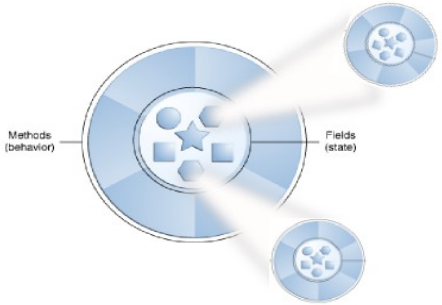




9. Building Objects

Composition Relationship

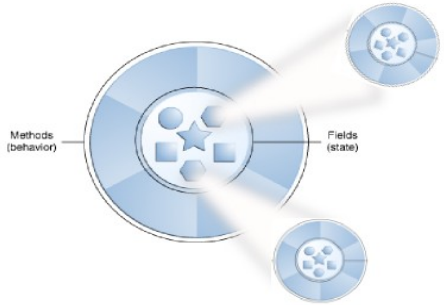
Composition helps to make parts **interchangeable.** In software development**, interchangeable** parts mean **reuse.**



9. Building Objects

Building in Phases

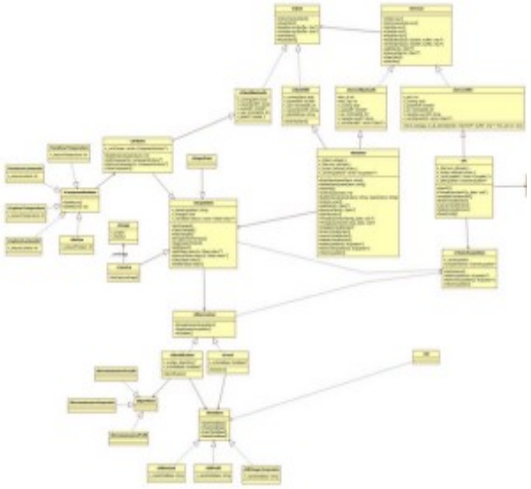
- Another advantage in using composition is that systems and subsystems can be built independently, and perhaps more importantly, tested and maintained independently.



9. Building Objects

Building in Phases

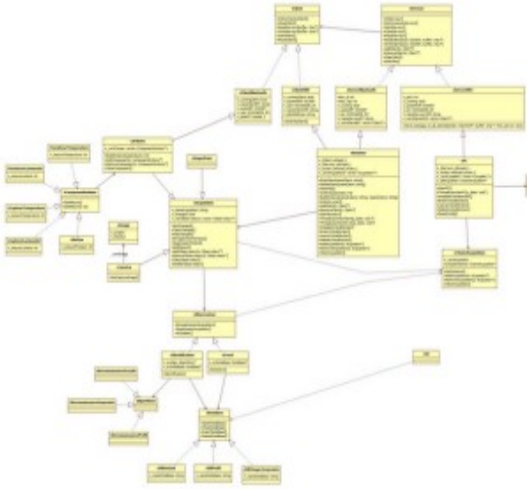
- There is no question that today's software systems are quite complex. To build quality software, you must follow one overriding rule to be successful: Keep things as simple as possible. For large software systems to work properly and be easily maintained, they must be broken into smaller, more manageable parts



9. Building Objects

Building in Phases

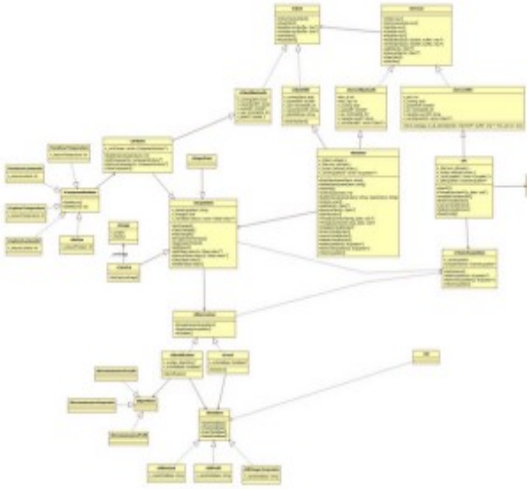
- There is no question that today's software systems are quite complex. To build quality software, you must follow one overriding rule to be successful: Keep things as simple as possible. For large software systems to work properly and be easily maintained, they must be broken into smaller, more manageable parts



9. Building Objects

Types of Composition

- Generally, there two types of composition: **association** and **aggregation.** In both cases, these relationships represent **collaborations** between the objects.



9. Building Objects

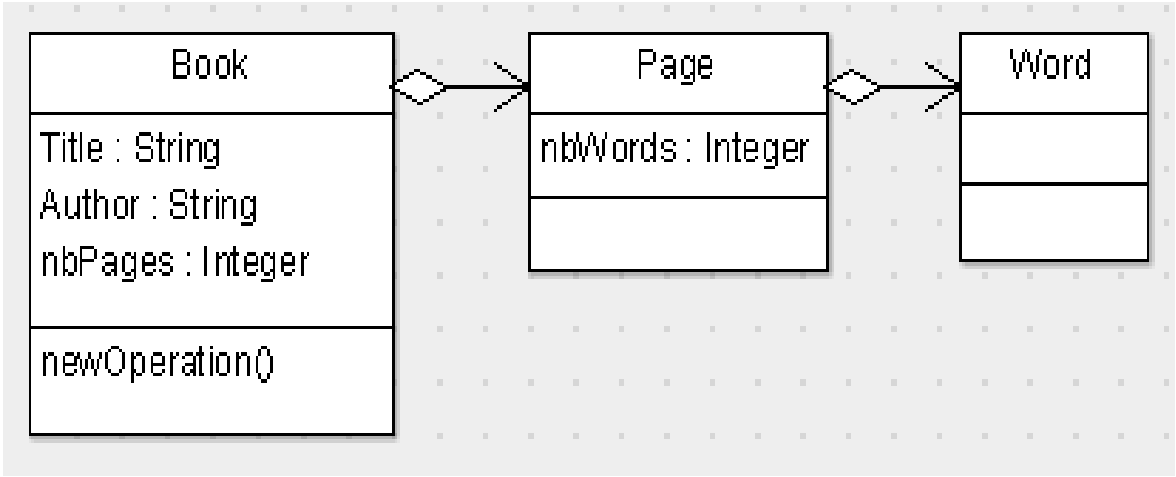
Types of Composition

- All forms of composition include a has-a relationship. However, there subtle differences between associations and aggregations based on how you visualize the parts of the whole. In a aggregation, you normally see the parts that make up a whole.

9. Building Objects

Aggregations

- Perhaps the most intuitive form of composition is aggregation. Aggregation means that a complex object is composed of other objects.



9. Building Objects

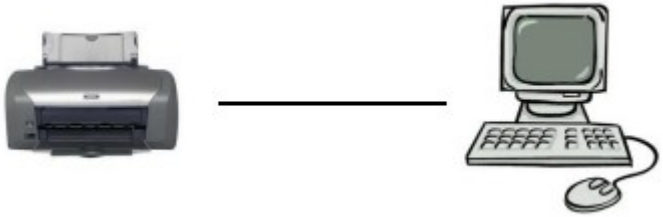
Associations

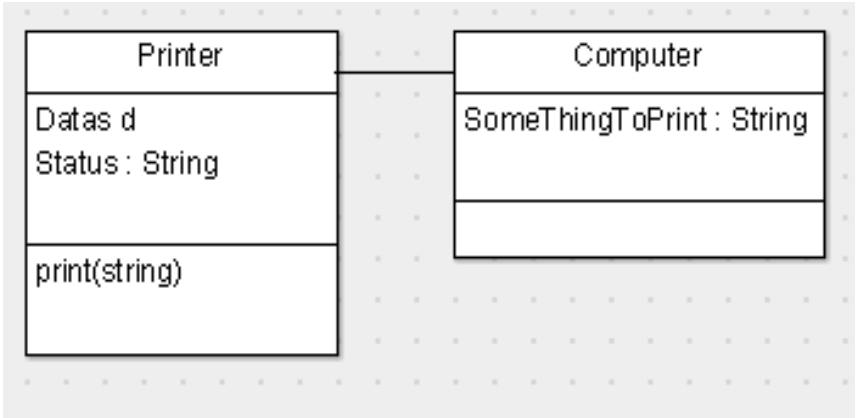
- Although aggregations represent relationships where you normally only see the whole, associations present both the whole and the parts.

9. Building Objects

Associations

► An example of association: A computer and a printer.





9. Building Objects

Cardinality

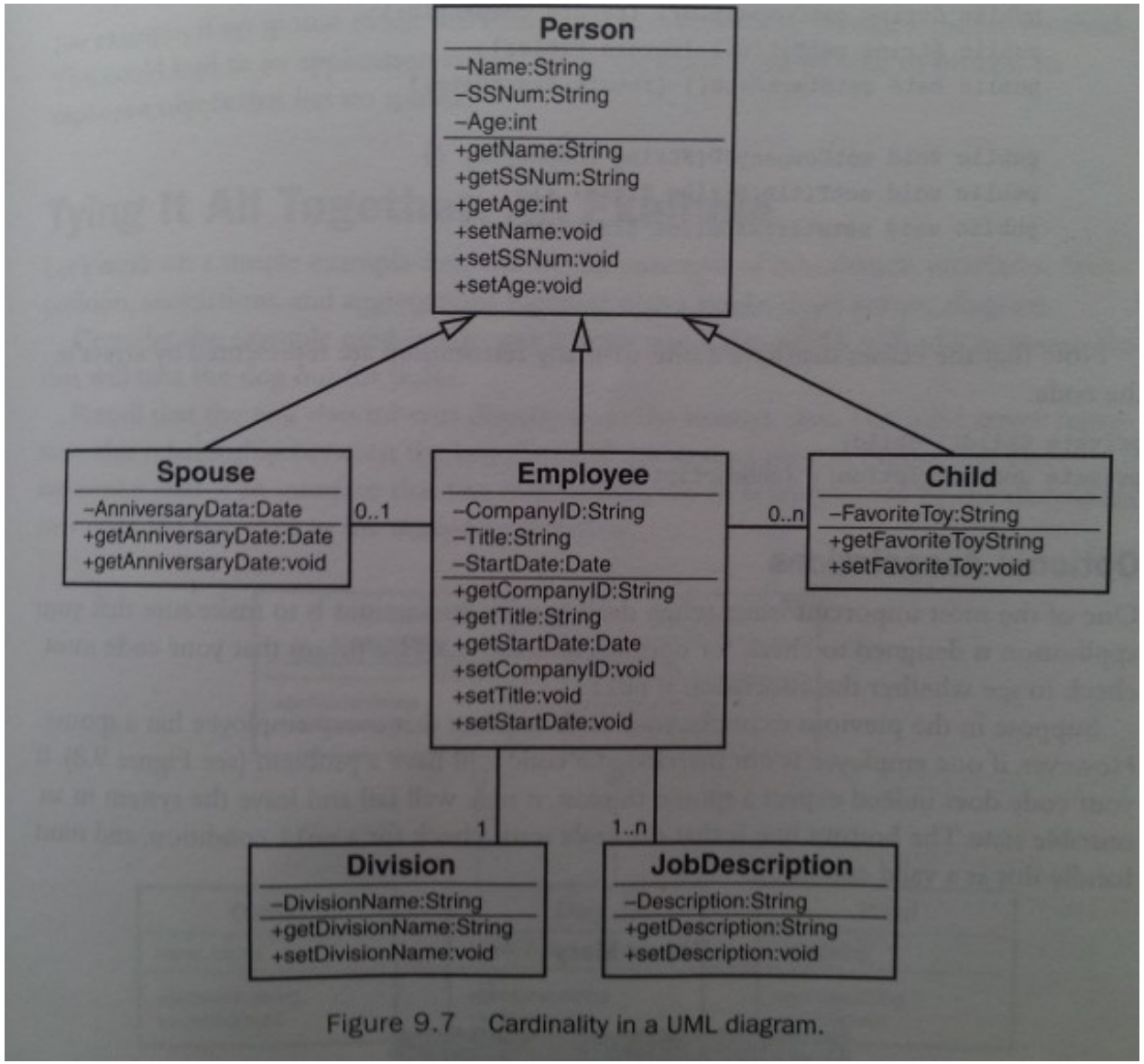


Figure 9.7 Cardinality in a UML diagram.

9. Building Objects

Cardinality

► To determine cardinality, ask the following questions:

-Which objects collaborate with which other objects?

-How many objects participate in each collaboration?

-Is the collaboration optional or mandatory?

9. Building Objects

Cardinality

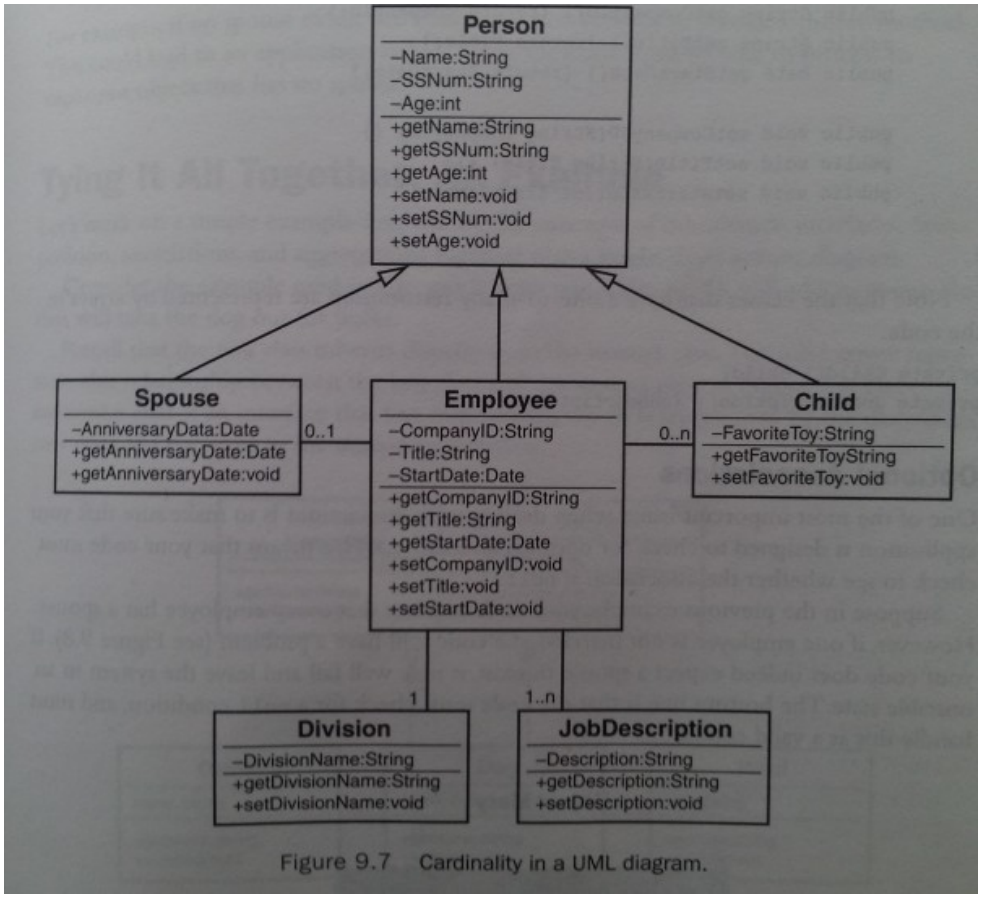


Figure 9.7 Cardinality in a UML diagram.

Table 9.1 Cardinality of Class Associations

Optional/Association	Cardinality	Mandatory
Employee/Division	1	Mandatory
Employee/JobDescription	1...n	Mandatory
Employee/Spouse	0...1	Optional
Employee/Child	0...n	Optional

1. Introduction to Object-Oriented Concepts
2. How to think in Terms of Objects
3. Advanced Object-Oriented Concepts
4. The anatomy of a Class
5. Class Design guidelines
6. Designing with objects
7. Mastering Inheritance and Composition
8. Frameworks and Reuse: Designing with interfaces and Abstract classes
9. Building objects

10. Creating Object Models with UML

11. Objects and Portable Data: XML
12. Persistent Objects: Serialization and Relational Databases
13. Objects and the Internet
14. Objects and Client/Server Applications
15. Design Patterns