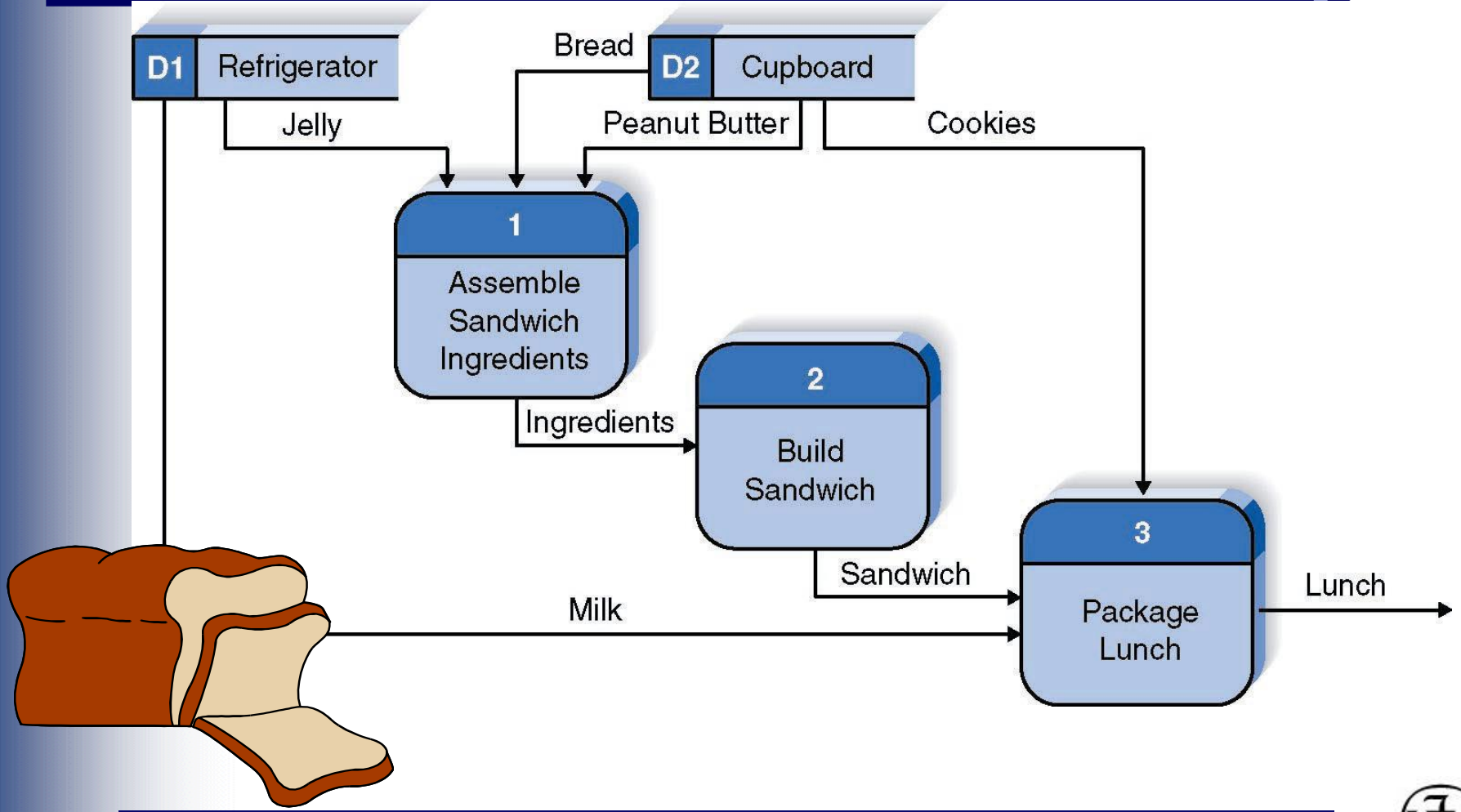
Non-Object-Oriented…

- Process models
  - Based on behaviour and actions
- Data Models
  - Based on static (fixed) representations of data

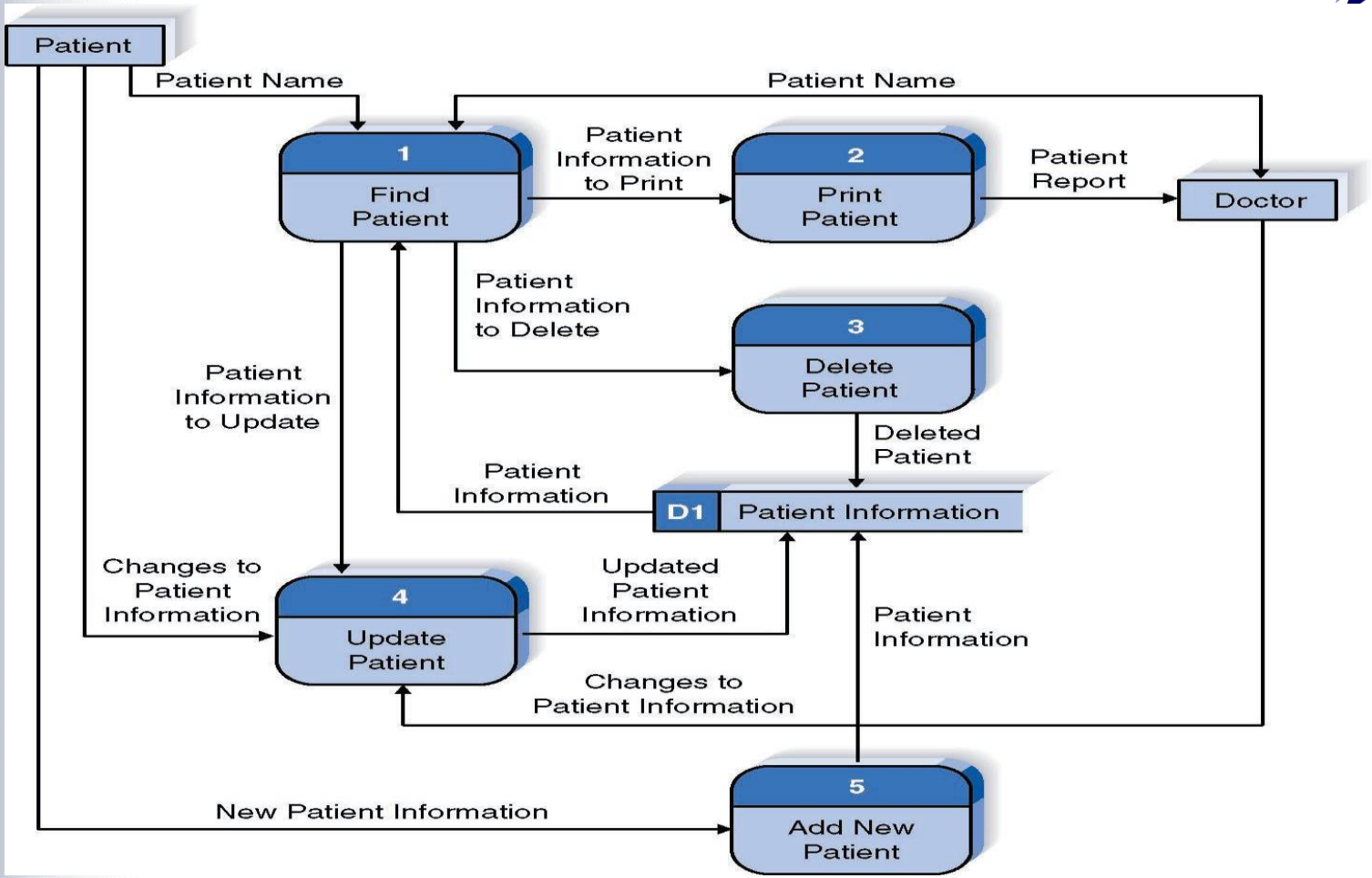
A "Simple" Process for Making Lunch



Process Modelling:

Data Flow Diagrams

Reading a DFD



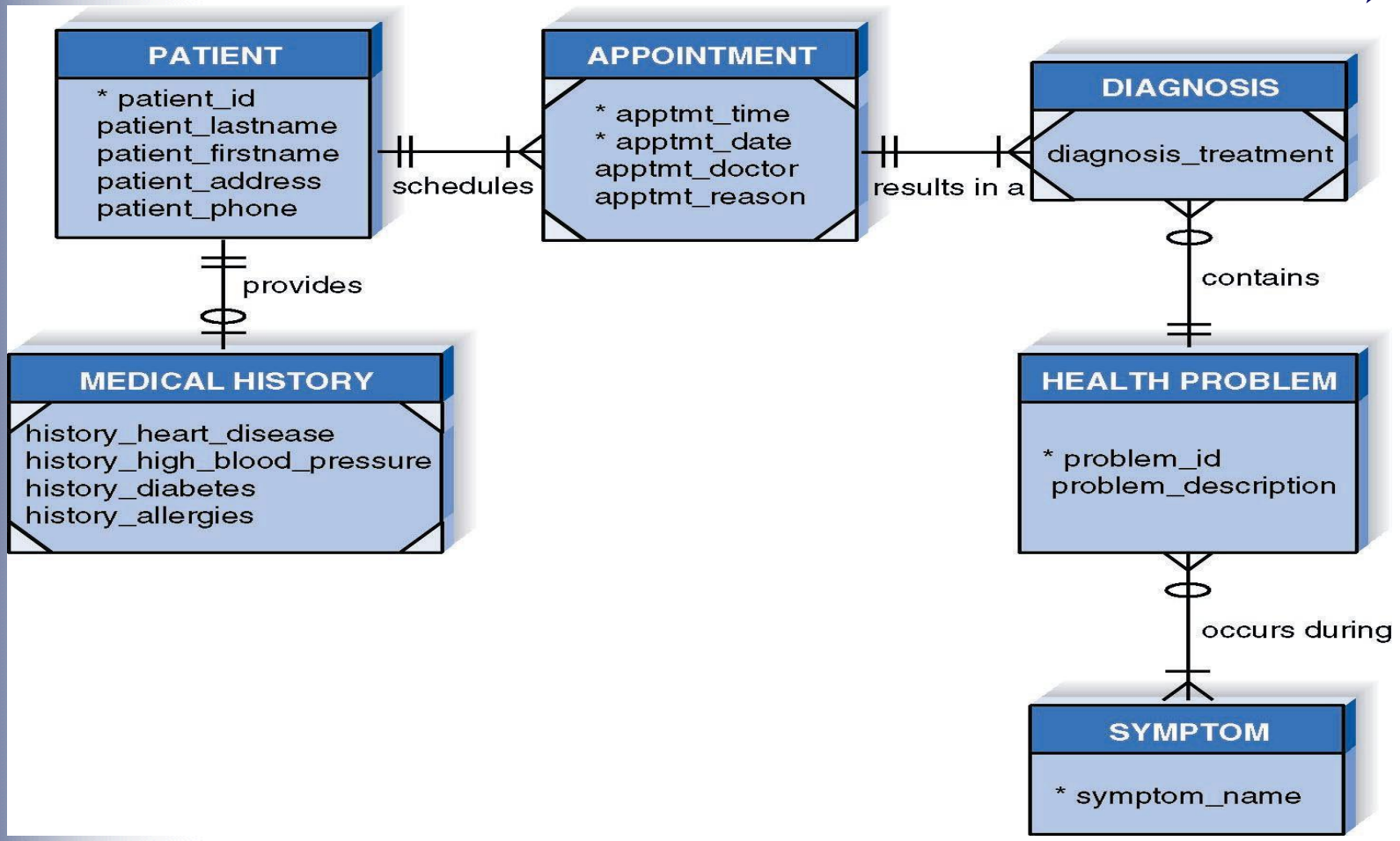
Data Modelling:

Entity-Relationship Diagrams (ERDs)

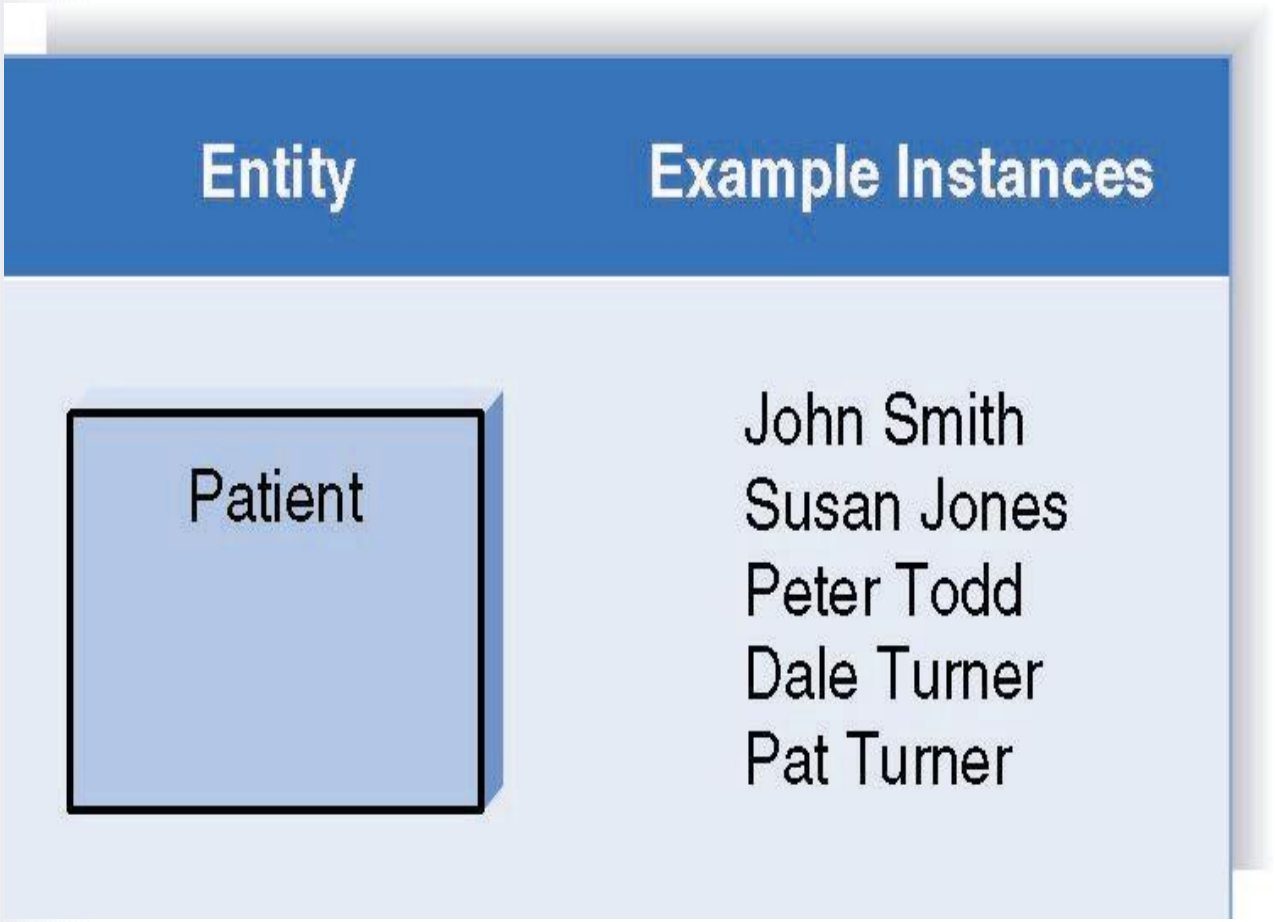
What Is an ERD?

- A picture showing the information created, stored, and used by a business system.
- Entities generally represent people, places, and things of interest to the organization.
- Lines between entities show relationships between entities.

An ERD Example



Entities and Instances



Object-Oriented Approaches

- Combine processes and data
- Are more 'natural'

Basic Characteristics of Object Oriented Systems

- Classes and Objects
- Methods and Messages
- Encapsulation and Information Hiding
- Inheritance
- Polymorphism

Classes and Objects

- Class – Template to define specific instances or objects
- Object – Instantiation of a class
- Attributes – Describes the object
- Behaviours – specify what object can do

Classes and Objects

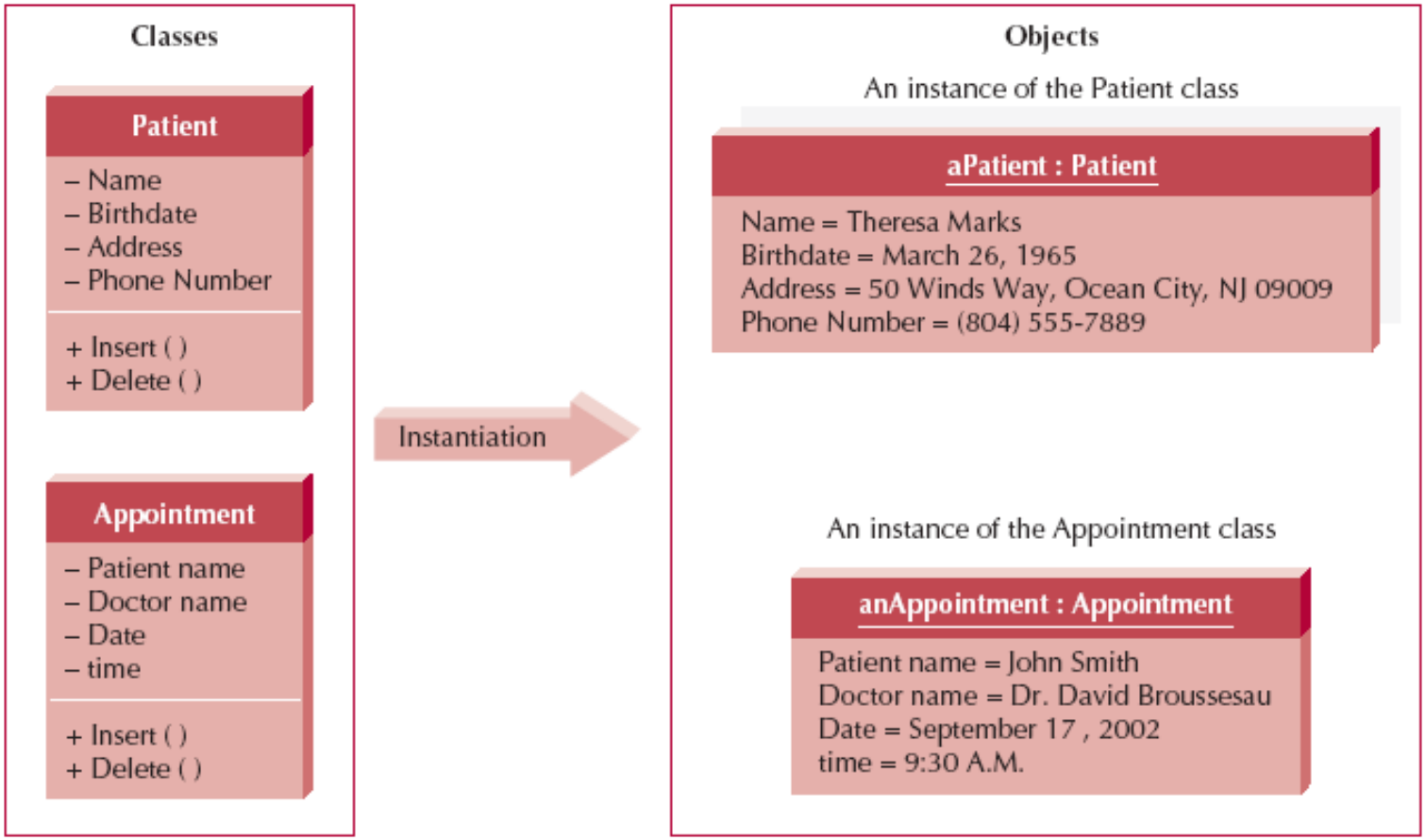


FIGURE 2-1 Classes and Objects

Methods and Messages

- Methods implement an object's behaviour
  - Analogous to a function or procedure
- Messages are sent to trigger methods
  - Procedure call from one object to the next

Messages and Methods

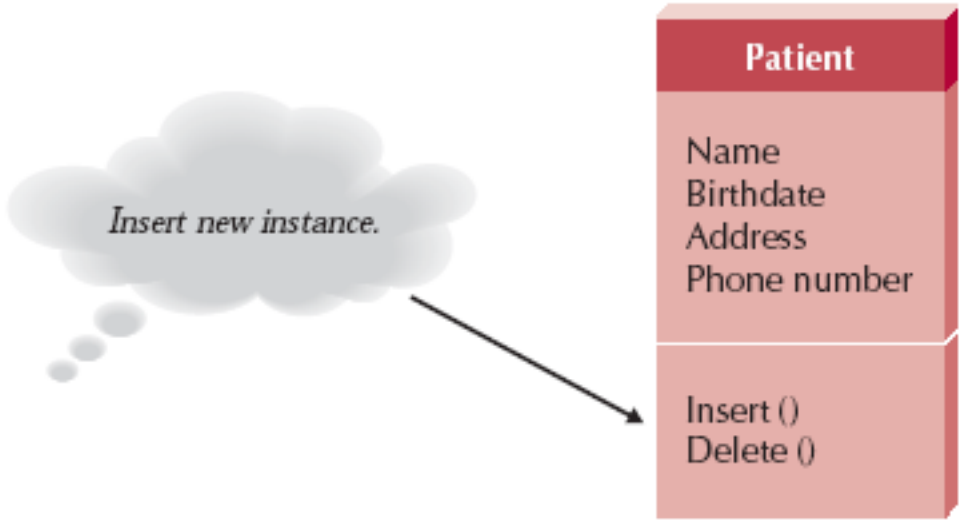


FIGURE 2-2
Messages and Methods

A message is sent to the application.

The object's insert method will respond to the message and insert a new patient instance.

Encapsulation and Information Hiding

- Encapsulation
  - combination of data and process into an entity
- Information Hiding
  - Only the information required to use a software module is published to the user
- Reusability is the Key Point
  - an object is used by calling methods

Inheritance

- Superclasses or general classes are at the top of a hierarchy of classes
- Subclasses or specific classes are at the bottom
- Subclasses inherit attributes and methods from classes higher in the hierarchy

Class Hierarchy

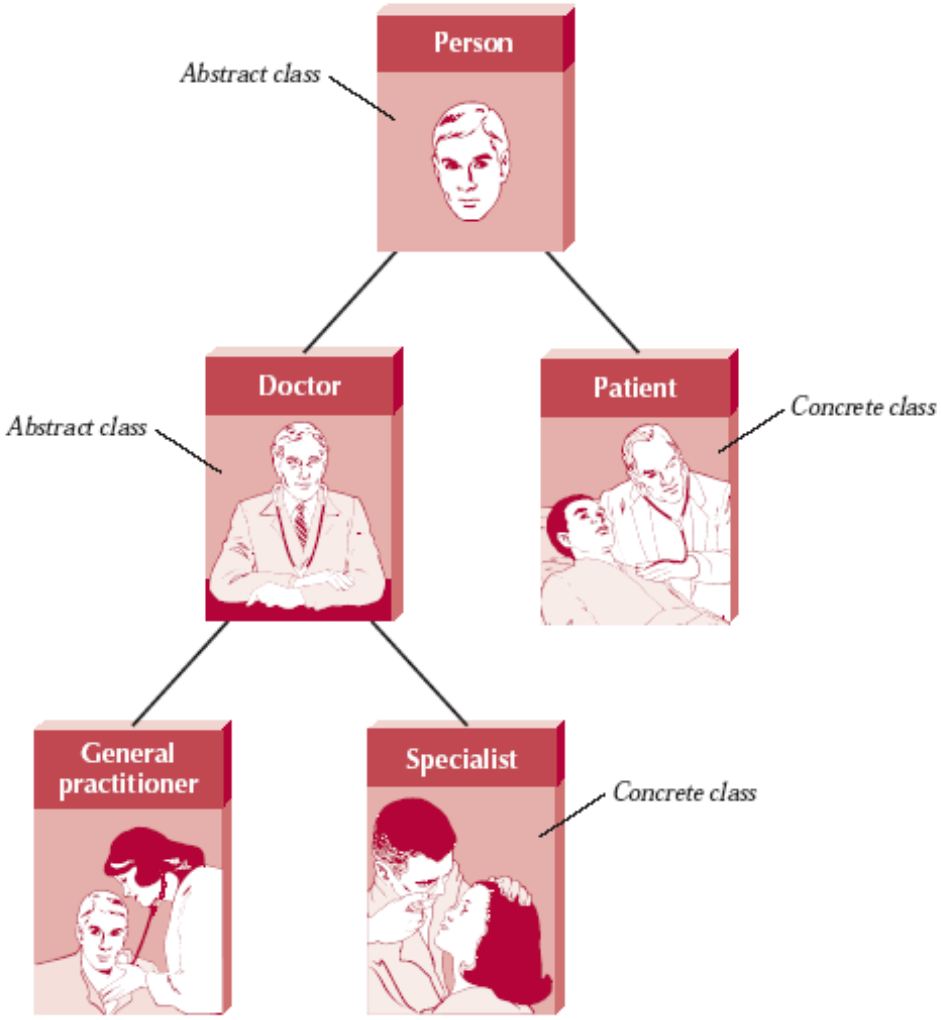


FIGURE 2-3
Class Hierarchy

Inheritance

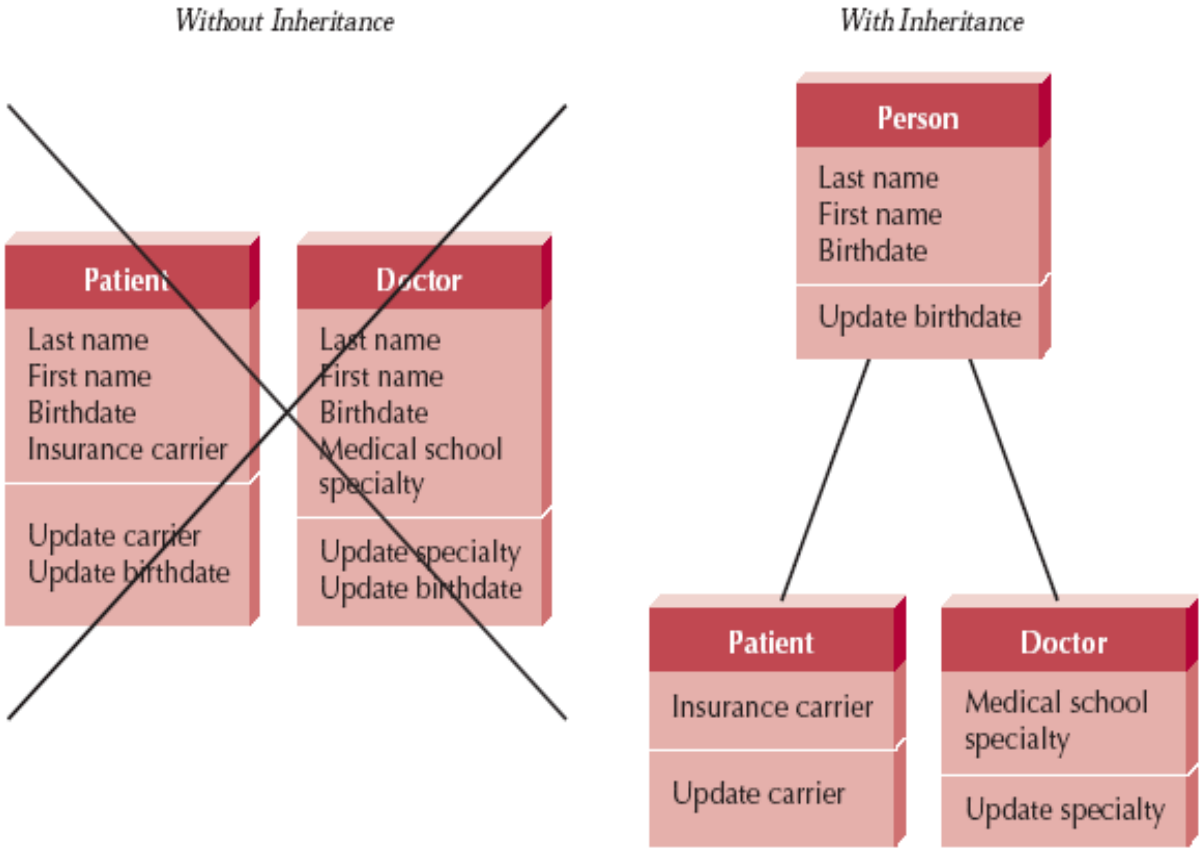


FIGURE 2-4
Inheritance

Polymorphism

- A message can be interpreted differently by different classes of objects

- e.g. A ‘Create_Record’ message is essentially the same thing, but causes ‘Create_Patient_Record’ by a ‘Patient_Database’ object, or ‘Create_Doctor_Record’ by a ‘Healthcare_Staff_Database’ object

Polymorphism & Encapsulation

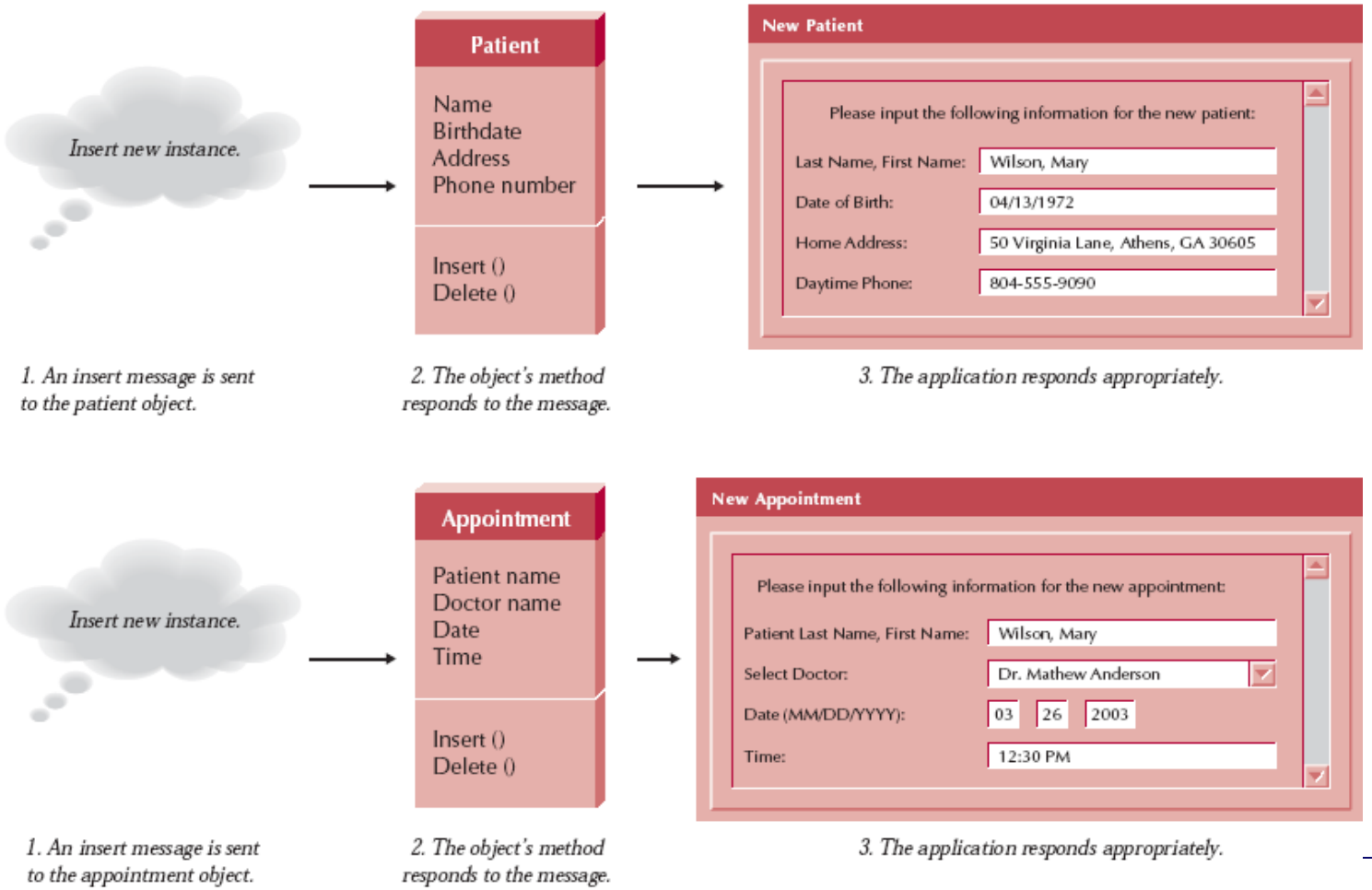


FIGURE 2-5 Polymorphism and Encapsulation

Concept	Supports	Leads to
Classes, objects, methods, and messages	■ A more realistic way for people to think about their business ■ Highly cohesive units that contain both data and processes	■ Better communication between user and analyst developer ■ Reusable objects ■ Benefits from having a highly cohesive system (see cohesion in Chapter 13)
Encapsulation and information hiding	■ Loosely coupled units	■ Reusable objects ■ Fewer ripple effects from changes within an object or in the system itself ■ Benefits from having a loosely coupled system design (see coupling in Chapter 13)
Inheritance	■ Allows us to use classes as standard templates from which other classes can be built	■ Less redundancy ■ Faster creation of new classes ■ Standards and consistency within and across development efforts ■ Ease in supporting exceptions
Polymorphism and Dynamic Binding	■ Minimal messaging that is interpreted by objects themselves	■ Simpler programming of events ■ Ease in replacing or changing objects in a syste ■ Fewer ripple effects from changes within an object or in the system itself
Use-case driven and use cases	■ Allows users and analysts to focus on how a user will interact with the system to perform a single activity	■ Better understanding and gathering of user nee ■ Better communication between user and analys
Architecture centric and functional, static, and dynamic views	■ Viewing the evolving system from multiple points of view	■ Better understanding and modeling of user nee ■ More complete depiction of information system
Iterative and incremental development	■ Continuous testing and refinement of the evolving system	■ Meeting real needs of users ■ Higher quality systems

FIGURE 2-8 Benefits of the Object Approach

The Unified Modelling Language, Version 2.0

- Functional Diagrams
- Structure Diagrams
- Behaviour Diagrams

Functional Diagrams

- Activity Diagrams
  - Illustrate business workflows
- Use-Case Diagrams
  - Capture business requirements
  - Illustrates interaction between system and environment

Structure Diagrams

- Class diagrams
  - relationship between classes
- Object diagrams
  - Relationships between objects

Behaviour Diagrams

- Interaction Diagrams ....
  - Sequence diagrams
    - Show Time-based ordering and behaviour of objects and their activities
- State Machines ...
  - Behavioural State Machines (Statechart diagrams)
    - Examines behaviour of one class/object

Object Oriented Systems Analysis and Design

- Use-case driven
- Iterative and Incremental
- Often associated with PHASED Development (a RAD methodology)

Basic Method for Development of Object Oriented Systems

- Identifying business value
- Analyze feasibility
- Develop workplan
- Staff the project

- Requirements determination
- Functional modelling
- Structural modelling
- Behavioural modelling

- Moving on to design