هندسة البرمجيات

المرحلة الثالثة

أ.م.د.ريا باسل احمد

التنمية المستدامة وعلاقتها بهندسة البرمجيات :

التنمية المستدامة هي مفهوم يشير إلى تحقيق التوازن بين التنمية الاقتصادية، والحفاظ على البيئة، وتعزيز العدالة الاجتماعية، بحيث تلبّي احتياجات الأجيال الحالية دون التأثير على قدرة الأجيال القادمة في تلبية احتياجاتهم الخاصة. وعند ربط هذا المفهوم بهندسة البرمجيات، يظهر دور كبير لهندسة البرمجيات في تعزيز التنمية المستدامة من خلال مجموعة من المبادئ والممارسات التي يمكن أن تسهم في تحقيق هذه الأهداف.

علاقة هندسة البرمجيات بالتنمية المستدامة تشمل:

كفاءة استخدام الموارد: من خلال تطوير برامج ونظم تعتمد على الكفاءة في استهلاك الموارد (مثل الطاقة، والمعالجة الحسابية)، يمكن تقليل الأثر البيئي الناتج عن تشغيل البرمجيات.
البرمجيات البيئية: تهتم بعض مشاريع البرمجيات بتطوير حلول تكنولوجية تعزز من الحفاظ على البيئة، مثل البرمجيات المساعدة في إدارة الطاقة، أو التنبؤ بالتغيرات المناخية، أو إدارة الموارد الطبيعية.

الموارد المستدامة: يمكن لهندسة البرمجيات أن تساهم في تطوير حلول لتتبع استهلاك الموارد وإدارتها بشكل أكثر استدامة في مختلف الصناعات، مثل حلول البرمجيات التي تدير دورة حياة المنتجات أو تساعد في تقليل الفاقد في العمليات.

تقليل الفاقد والتكلفة: من خلال تحسين الكود البرمجي وتطوير برامج ذات جودة عالية وفعالة، يمكن تقليل التكاليف التشغيلية وتقليل الأثر البيئي الناتج عن استخدام البرمجيات في المؤسسات.

تطبيقات تكنولوجيا المعلومات للمجتمع: تساهم البرمجيات في العديد من الحلول الاجتماعية مثل التعليم عن بعد، الصحة الرقمية، وتقنيات دعم الفقراء والمهمشين. هذه الأنظمة تدعم المساواة الاجتماعية والتنمية البشرية المستدامة.

دورة حياة البرمجيات المستدامة: عند تصميم البرمجيات، يجب التفكير في دورة حياتها بأكملها، بدءًا من مرحلة التطوير وصولاً إلى مرحلة الصيانة. يمكن أن يشمل ذلك استخدام تقنيات لتقليل الفاقد، وزيادة كفاءة الأداء، واستخدام البرمجيات مفتوحة المصدر للمساهمة في تقليل التكاليف البيئية.

بالتالي، تساهم هندسة البرمجيات في التنمية المستدامة من خلال حلول تكنولوجية مبتكرة تهدف إلى تحسين الكفاءة الاقتصادية والاجتماعية وتقليل التأثير البيئي.

Software Engineering

Software Process

Chapter 2

- A software process is a set of related activities that leads to the production of a software product.
- These activities may involve the development of software from scratch in a standard programming language like Javaor C.
- There are many different software processes, but all must include four activities that are fundamental to software engineering:
    1. *Software specification*
    2. *Software design and implementation*
    3. *Software validation*
    4. *Software evolution*

- عملية البرامج هي مجموعة من الأنشطة ذات الصلة التي تؤدي إلى إنتاج منتج برمجي.
- قد تتضمن هذه الأنشطة تطوير برنامج من البداية بلغة برمجة قياسية مثل Java أو C.
- هناك العديد من العمليات البرمجية المختلفة ، ولكن يجب أن تتضمن جميعها أربعة أنشطة أساسية لهندسة البرمجيات:

1. مواصفات البرنامج.
2. تصميم البرامج وتنفيذها.
3. التحقق من صحة البرامج.
4. تطور البرمجيات.

Software process models

1. The waterfall model. represents them as separate process phases.
2. Incremental development. This approach interleaves the activities. It is developed as a series of versions (increments), with each version adding functionality to the previous version.
3. Reuse-oriented software engineering. This approach is based on the existence of a significant number of reusable components.

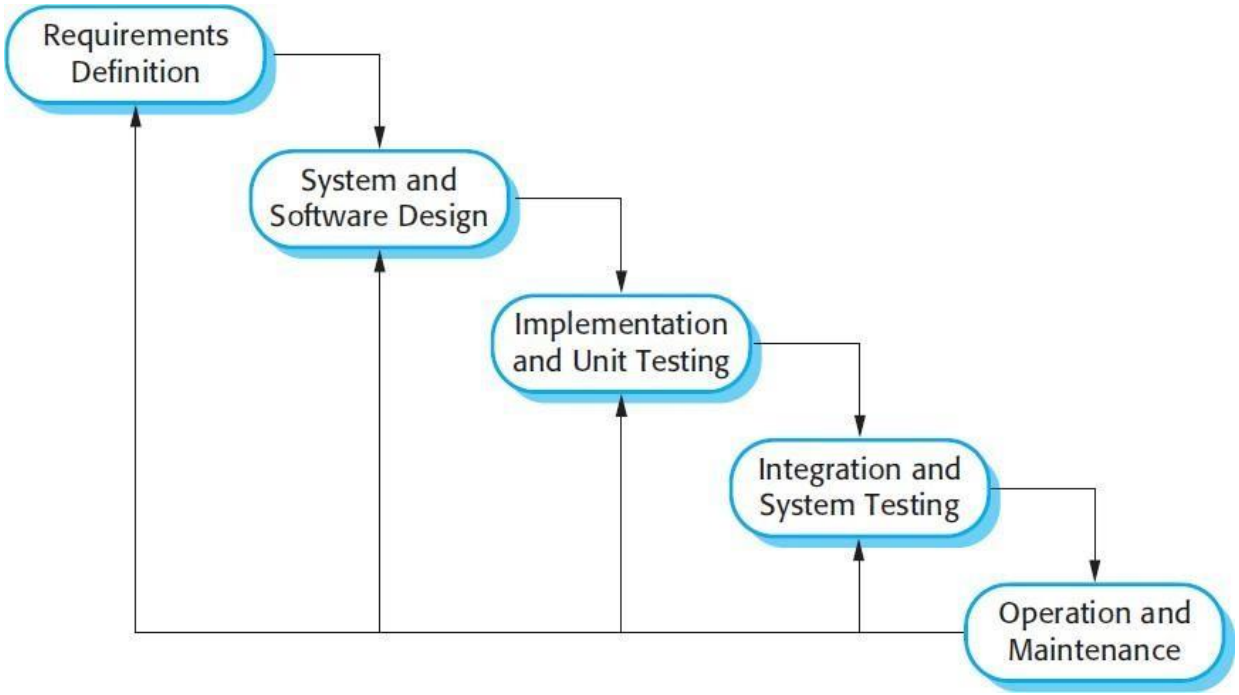
نماذج العمليات البرمجية

1. **نموذج الشلال**. يمثلها كمراحل عملية منفصلة.

2. **التطوير المتزايد**. هذا النهج يشتمل على الأنشطة ، وقد تم تطويره كسلسلة من الإصدارات (الزيادات) ، مع كل إصدار يضيف وظائف إلى الإصدار السابق.

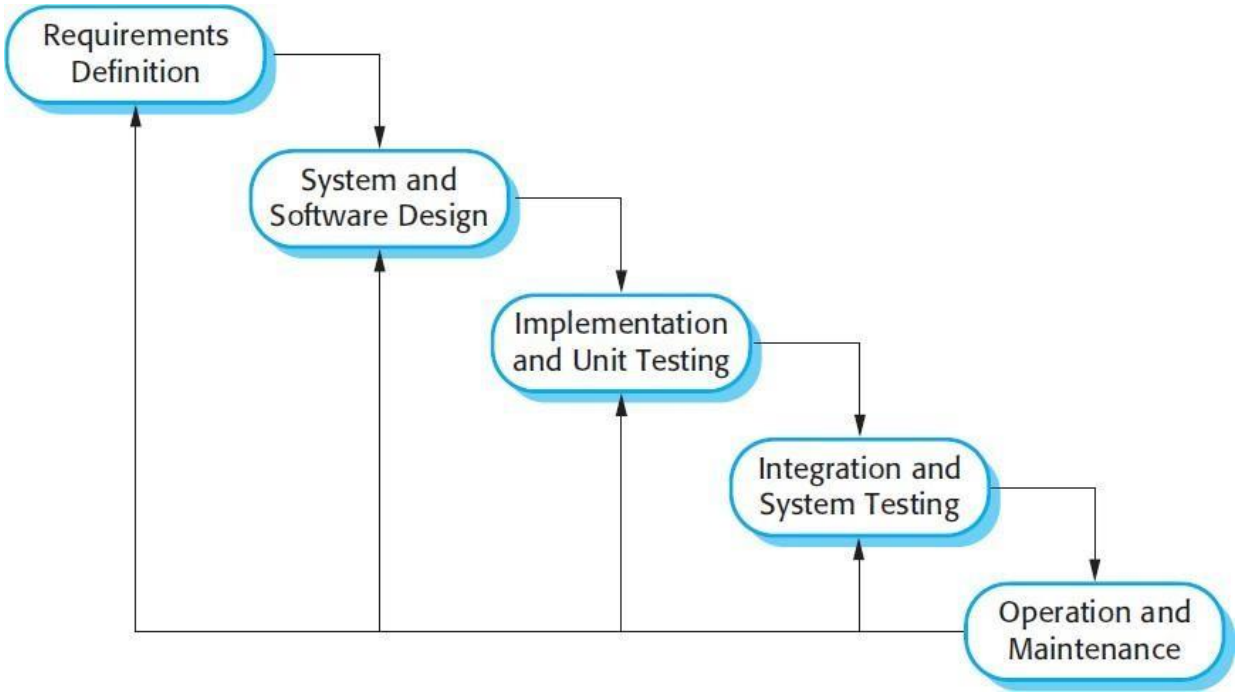
3. **إعادة استخدام هندسة البرمجيات**. يعتمد هذا النهج على وجود عدد كبير من المكونات القابلة لإعادة الاستخدام.

The waterfall model



- The result of each phase is one or more documents that are approved ('signed off').
- The following phase should not start until the previous phase has finished.
- The waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development.

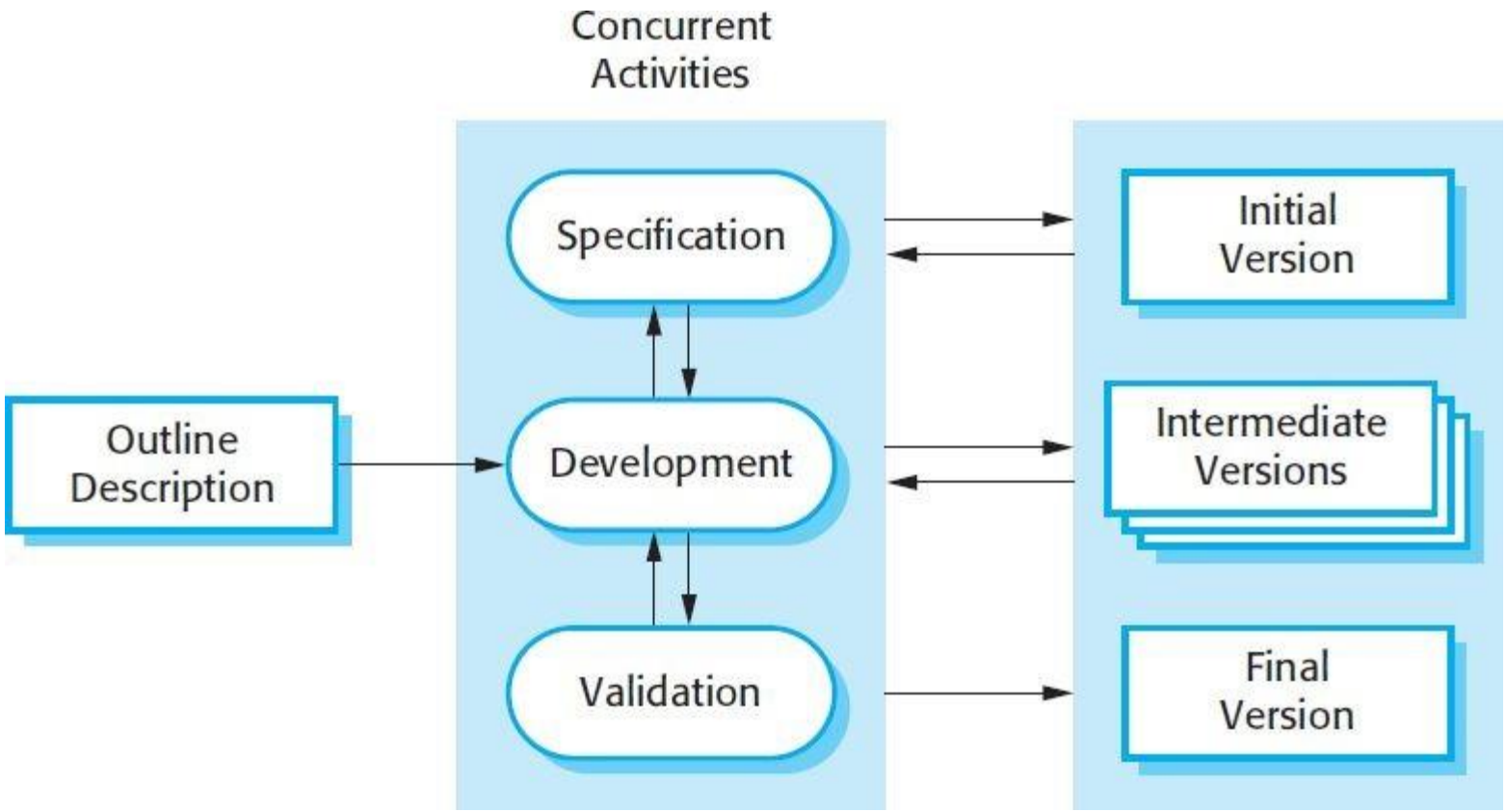
نموذج الشلال



- نتيجة كل مرحلة هي وثيقة واحدة أو أكثر تمت الموافقة عليها ("موقعة").
يجب ألا تبدأ المرحلة التالية حتى تنتهي المرحلة السابقة.

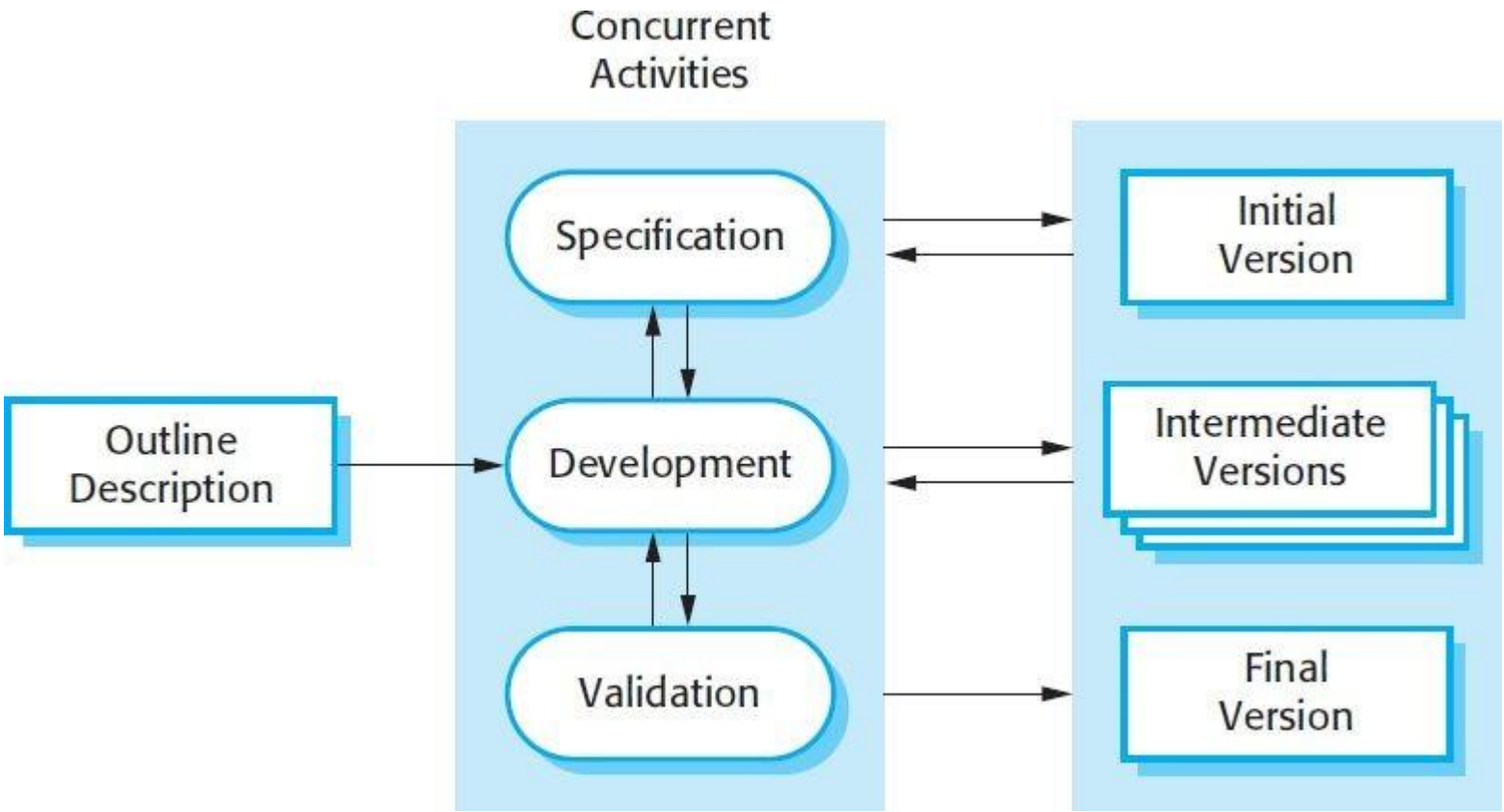
- يجب استخدام نموذج الشلال فقط عندما تكون المتطلبات مفهومة جيدا ومن غير المحتمل أن تتغير جذريا أثناء تطوير النظام.

Incremental development



- By developing the software incrementally, it is cheaper and easier to make changes in the software.
- The customer can evaluate the system at a relatively early stage in the development to see if it delivers what is required.

التطوير المتزايد



- من خلال تطوير البرنامج بشكل تدريجي ، يكون إجراء تغييرات في البرنامج أرخص وأسهل.
- يمكن للعميل تقييم النظام في مرحلة مبكرة نسبيا من التطوير لمعرفة ما إذا كان يقدم ما هو مطلوب.

- Incremental development has three important benefits, compared to the waterfall model:
  1. The cost.
  2. It is easier to get customer feedback
  3. More rapid delivery.

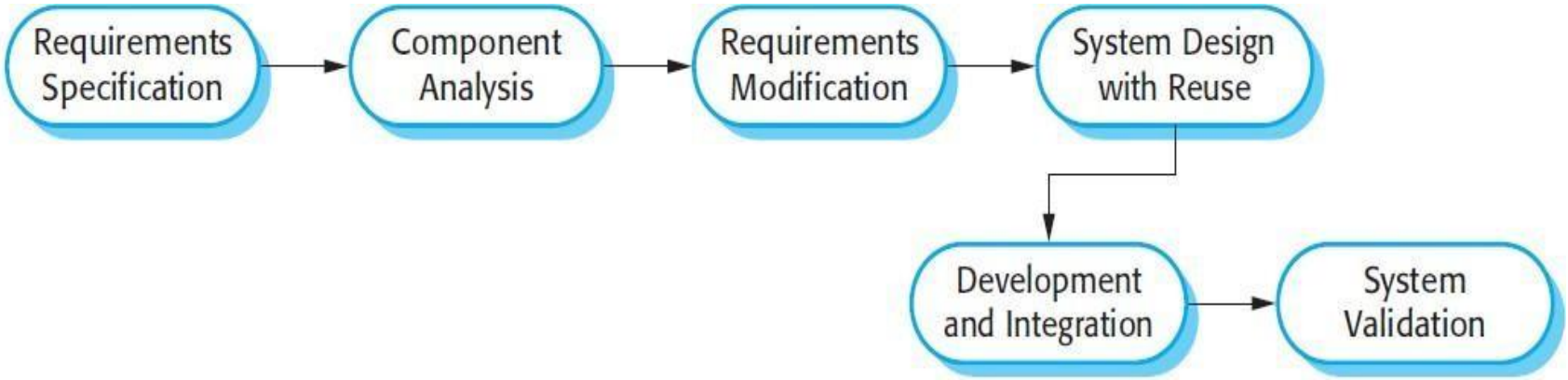
- What are the problems of this model? Explain.

- للتطوير التدريجي ثلاث فوائد مهمة ، مقارنة بنموذج الشلال:

1. التكلفة.
2. من الأسهل الحصول على ملاحظات العملاء.
3. تسليم أسرع.

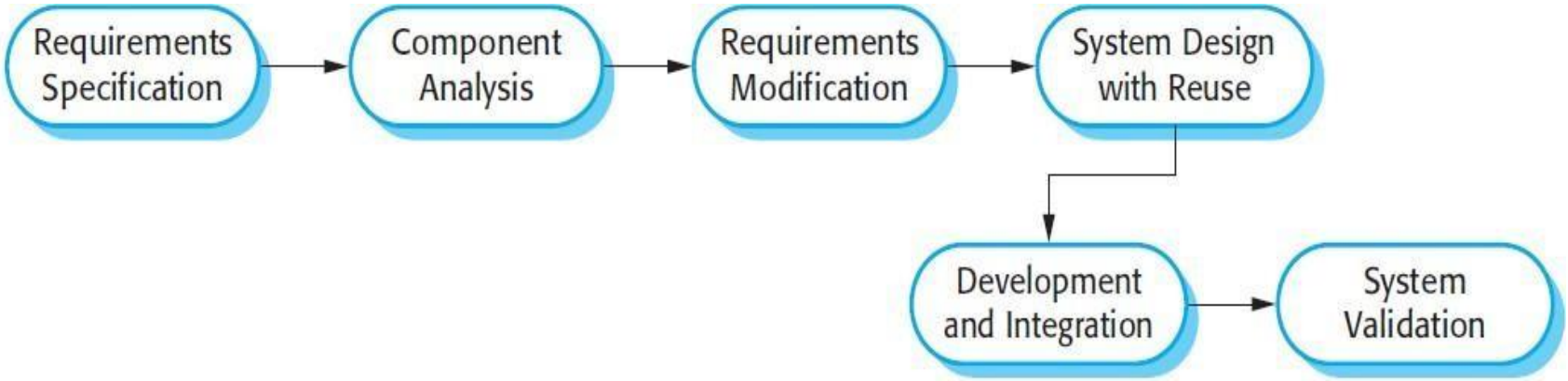
- ما هي مشاكل هذا النموذج؟ أشرح.

REUSE-ORIENted software engineering



- Reuse-oriented software engineering has the advantage of reducing the amount of software to be developed and so reducing cost and risks.
- It usually also leads to faster delivery of the software.

هندسة البرمجيات الموجهة لإعادة الاستخدام



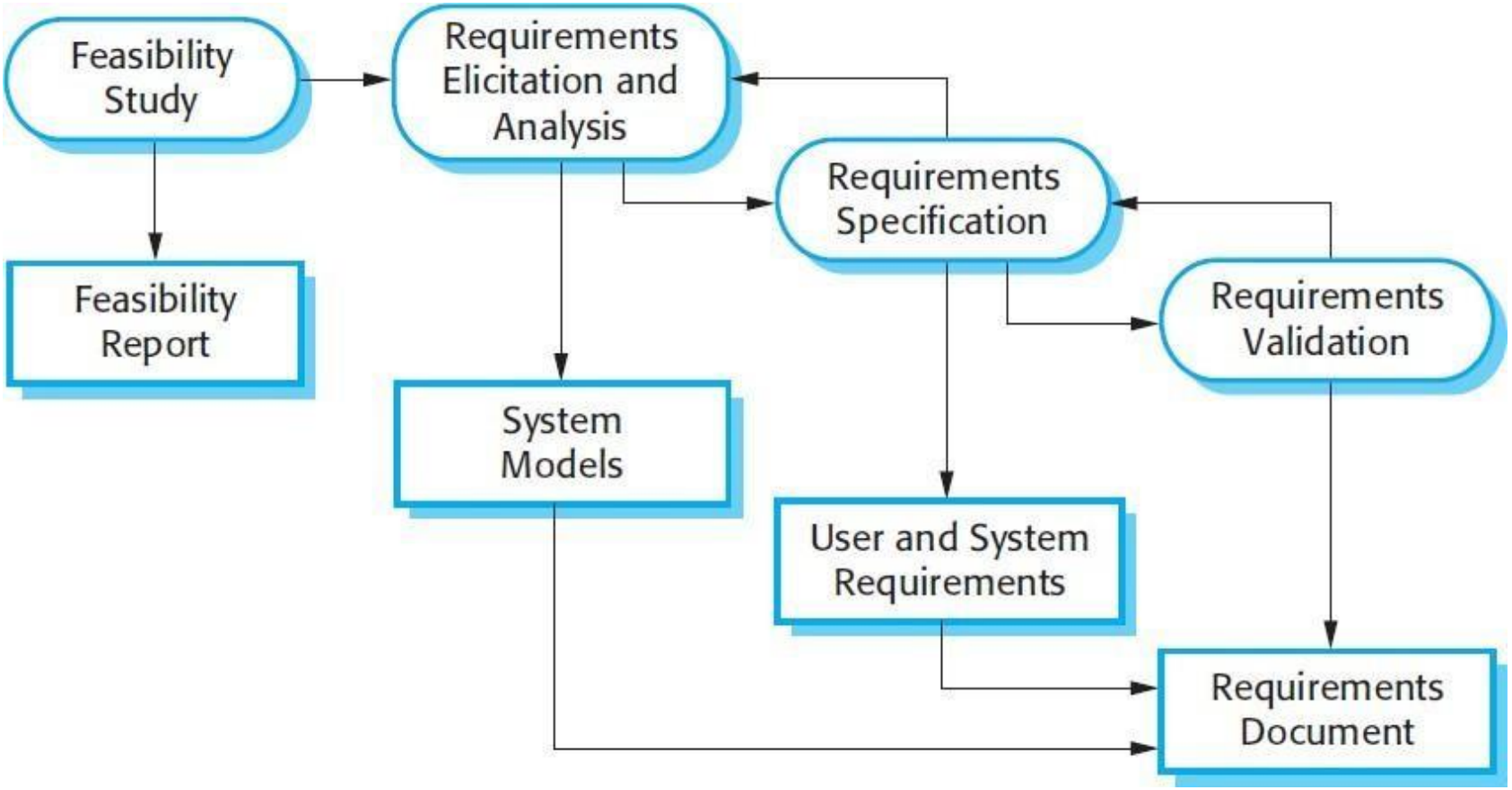
- تتمتع هندسة البرمجيات الموجهة لإعادة الاستخدام بميزة تقليل كمية البرامج المطلوب تطويرها وبالتالي تقليل التكلفة والمخاطر.
- عادة ما يؤدي ذلك أيضا إلى تسليم البرنامج بشكل أسرع.

Software specification

- Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.

مواصفات البرنامج

- مواصفات البرامج أو هندسة المتطلبات هي عملية فهم وتحديد الخدمات المطلوبة من النظام وتحديد القيود المفروضة على تشغيل النظام وتطويره.

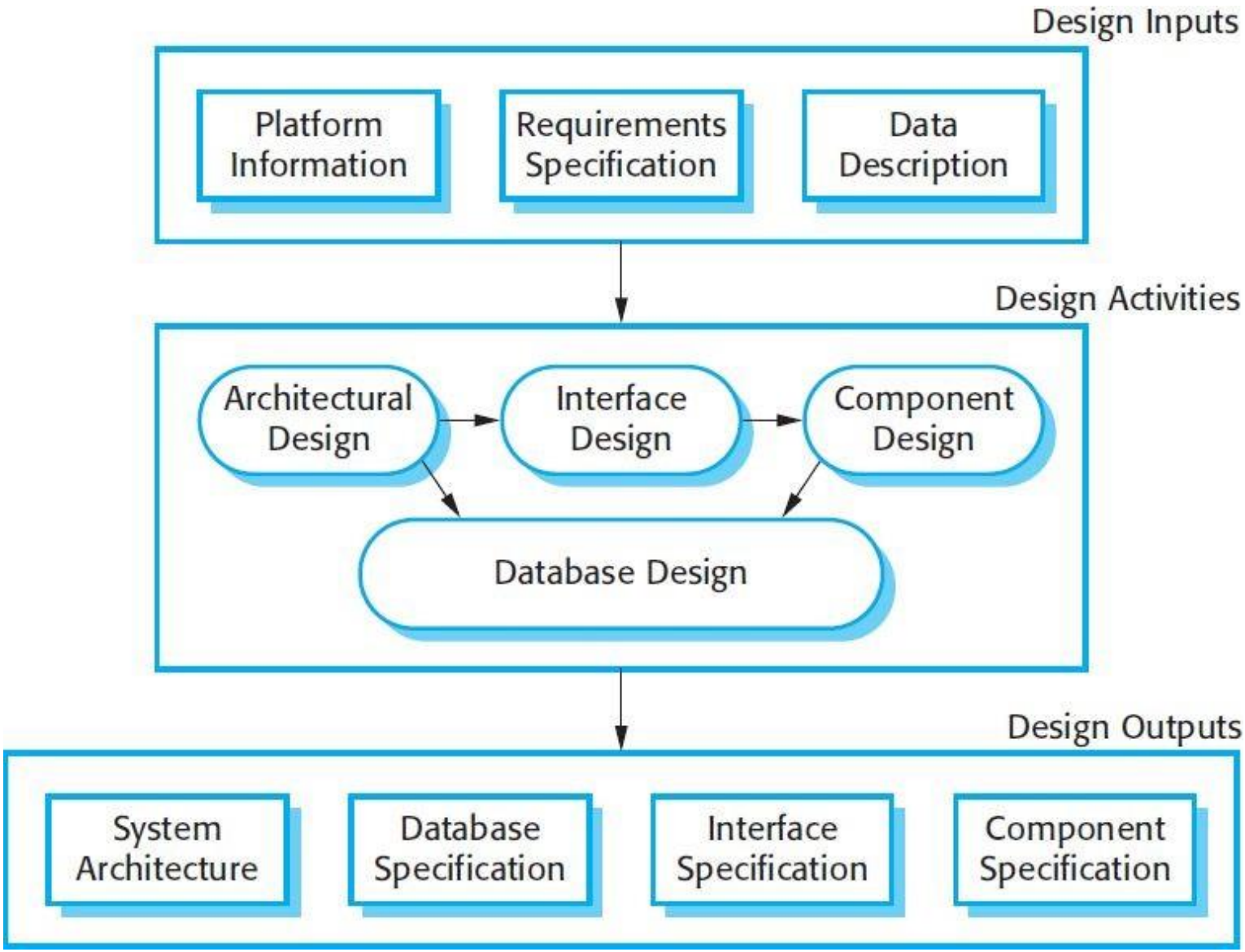

Feasibility Study
Requirements Elicitation and Analysis
Requirements Specification
Feasibility Report
System Models
Requirements Validation
User and System Requirements
Requirements Document

Software design and implementation

- The implementation stage of software development is the process of converting a system specification into an executable system.
- Designers do not arrive at a finished design immediately but develop the design iteratively.

تصميم البرامج وتنفيذها

- مرحلة التنفيذ في تطوير البرمجيات هي عملية تحويل مواصفات النظام إلى نظام قابل للتنفيذ.
- لا يصل المصممون إلى التصميم النهائي على الفور ولكن يطورون التصميم بشكل متكرر.


Design Inputs
Platform Information
Requirements Specification
Data Description
Design Activities
Architectural Design
Interface Design
Component Design
Database Design
Design Outputs
System Architecture
Database Specification
Interface Specification
Component Specification

Software validation

- Software validation or, more generally, verification and validation (V&V) is intended to show that a system both conforms to its specification and that it meets the expectations of the system customer.
- Program testing, where the system is executed using simulated test data, is the principal validation technique.

التحقق من صحة البرامج

- يُقصد بالتحقق من صحة البرامج ، أو بشكل عام ، التحقق و المصادقة (V&V) لإظهار أن النظام يتوافق مع مواصفاته وأنه يلبي توقعات عميل النظام.
- اختبار البرنامج ، حيث يتم تنفيذ النظام باستخدام بيانات الاختبار المحاكاة ، هو أسلوب التحقق الرئيسي.

- The stages in the testing process are:

1. Development testing:The components making up the system are tested by the people developing the system. Each component is tested independently, without other system components.
2. System testing: System components are integrated to create a complete system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems.
3. Acceptance testing:This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data.

- Acceptance testing is sometimes called '**alpha testing**'.
- When a system is to be marketed as a software product, a testing process called '**beta testing**' is often used. Beta testing involves delivering a system to a number of potential customers who agree to use that system.

*مراحل عملية الاختبار هي:

1. اختبار التطوير: يتم اختبار المكونات المكونة للنظام من قبل الأشخاص الذين يطورون النظام. يتم اختبار كل مكون بشكل مستقل ، بدون مكونات النظام الأخرى.
2. اختبار النظام: تم دمج مكونات النظام لإنشاء نظام كامل. تهتم هذه العملية بإيجاد الأخطاء التي تنتج عن التفاعلات غير المتوقعة بين المكونات ومشاكل واجهة المكونات.
3. اختبار القبول: هذه هي المرحلة الأخيرة في عملية الاختبار قبل قبول النظام للاستخدام التشغيلي. يتم اختبار النظام بالبيانات المقدمة من عميل النظام بدلًا من بيانات الاختبار المحاكاة.

- يطلق على اختبار القبول أحيانا اسم "اختبار ألفا".

- عندما يتم تسويق نظام ما كمنتج برمجي ، غالبا ما يتم استخدام عملية اختبار تسمى "اختبار تجريبي". يتضمن الاختبار التجريبي تقديم نظام لعدد من العملاء المحتملين الذين يوافقون على استخدام هذا النظام.

Software evOLUTION

- The flexibility of software systems is one of the main reasons why more and more software is being incorporated in large, complex systems.
- changes can be made to software at any time during or after the system development.

تطور البرمجيات

- تعد مرونة أنظمة البرامج أحد الأسباب الرئيسية وراء دمج المزيد والمزيد من البرامج في أنظمة كبيرة ومعقدة.
- يمكن إجراء تغييرات على البرنامج في أي وقت أثناء أو بعد تطوير النظام.

Software Engineering

Software Requirements

Chapter 3

Requirements: Introduction

- *Requirements* =services the system is expected to provide + constraints placed on the system
- *Requirements engineering* = gathering, negotiating, analyzing, and documenting requirements
- The requirements could be expressed at various levels of abstraction
- The way requirements are defined has a major impact on the development of the software product.

- المتطلبات =الخدمات التي من المتوقع أن يوفرها النظام +لقيود الموضوعة على النظام.
  - هندسة المتطلبات =جمع المتطلبات والتفاوض عليها وتحليلها وتوثيقها.
  - يمكن التعبير عن المتطلبات على مستويات مختلفة من التجريد.
  - طريقة تحديد المتطلبات لها تأثير كبير على تطوير منتج البرنامج.

Classification of requirements:

- ***User requirements***: higher level description of services requested and constraints imposed
- ***System requirements***:a more detailed, structured description of services and constraints. Usually included in the contract between the developer and the client

An even more detailed description of requirements can be provided in a *software design specification(closer to implementation)*

Examples of *user requirements definition and system requirements specification*

Requirements definition

Requirements specification

1. The user should be provided with facilities to define the type of external files.

1.2 Each external file type may have an associated tool which may be applied to the file.

3. Each external file type may be represented as a specific icon on the user's display.

1.4 Facilities should be provided for the icon representing an external file type to be defined by theuser.

1.5 When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file representedby the selected icon.

أمثلة على تعريف متطلبات المستخدم ومواصفات متطلبات النظام

تعريف المتطلبات

مواصفات المتطلبات

1. يجب تزويد المستخدم بوسائل لتحديد نوع الملفات الخارجية.
2. قد يكون لكل نوع ملف خارجي أداة مرتبطة به والتي يمكن تطبيقها على الملف.
3. يمكن تمثيل كل نوع ملف خارجي كرمز محدد على شاشة المستخدم.
4. يجب توفير التسهيلات لأليقونة التي تمثل نوع ملف خارجي يحدده المستخدم.
5. عندما يختار المستخدم رم زايمثل ملفًا خارجيًا ، فإن تأثير هذا التحديد هو تطبيق ألداة المرتبطة بنوع الملف الخارجي على الملف الذي يمثله الرمز المحدد.

Types of software system requirements

- *Functional requirements,* describe the requested functionality/behaviour of the system: services (functions), reactions to inputs, exceptions, modes of operations
- *Non-functional requirements,* are constraints on the services offered by the system, and on the development process

- *Domain requirements,* can be either functional or non-functional and reflect the particularities of the application domain

أنواع متطلبات نظام البرمجيات

- المتطلبات الوظيفية :وصف الوظيفة / السلوك المطلوب لخدمات النظام (الوظائف)، وردود الفعل على المدخالت ، واالستثناءات ، وأنماط العمليات.
- المتطلبات غير الوظيفية :هي قيود على الخدمات التي يقدمها النظام ، وعلى عملية التطوير.
- متطلبات المجال :يمكن أن تكون وظيفية أو غير وظيفية وتعكس خصوصيات مجال التطبيق.

Functional requirements:

- Depend on the system, the software, and the users
- Can be expressed at different levels of detail (user/system requirements)
- For a system, it is desirable to have a complete and consistent set of functional requirements
  - *Completeness:* all required system facilities are defined
  - *Consistency:* there are no contradictions in requirements

المتطلبات الوظيفية:

- تعتمد على النظام والبرمجيات والمستخدمين.
- يمكن التعبير عنها بمستويات مختلفة من التفاصيل)متطلبات المستخدم /النظام(
- بالنسبة للنظام ، من المستحسن أن يكون لديك مجموعة كاملة ومتسقة من المتطلبات الوظيفية.
  - االكتمال :يتم تحديد جميع مرافق النظام المطلوبة.
  - التناسق :ال يوجد تناقضات في المتطلبات.

Non-functional requirements:

- Many applications to the system as a whole
- More critical than individual functional requirements
- More difficult to verify

Kinds of non-functional requirements:

- Product requirements
- Organizational requirements
- External requirements

المتطلبات غير الوظيفية:

- العديد من التطبيقات على النظام ككل.
- أكثر أهمية من المتطلبات الوظيفية الفردية.
- أكثر صعوبة في التحقق.

أنواع المتطلبات غير الوظيفية:

- متطلبات المنتج.
- المتطلبات التنظيمية.
- المتطلبات الخارجية.

Product requirements:	specify that the delivered product *must behave* in a particular way *e.g. execution speed, reliability, etc.*
Organizational requirements:	are a consequence of *organizational policies* and procedures *e.g. process standards used, implementation requirements, etc.*
External requirements:	arise from *factors which are external* to the system and its development process *e.g. interoperability requirements, legislative requirements, etc.*

متطلبات المنتج:	تحديد أن المنتج الذي تم تسليمه يجب أن يتصرف بطريقة معينة ، على سبيل المثال سرعة التنفيذ والموثوقية وما إلى ذلك.
المتطلبات التنظيمية:	هي نتيجة للسياسات والإجراءات التنظيمية على سبيل المثال معايير العملية المستخدمة ، متطلبات التنفيذ ، إلخ.
المتطلبات الخارجية:	تنشأ من عوامل خارجية عن النظام وعملية تطويره على سبيل المثال متطلبات التشغيل البيني والمتطلبات التشريعية وما إلى ذلك.

Requirements Measures

Property	***Measure***
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes; Number of RAM chips
Ease of use	Training time Rate of errors made by trained users Number of help frames

تدابير المتطلبات

ملكية	قياس
السرعة	المعاملات التي تمت معالجتها / وقت استجابة المستخدم / الحدث الثاني وقت تحديث الشاشة.
الحجم	ك بايت عدد شرائح ذاكرة الوصول العشوائي
سهولة االستخدام	وقت التدريب. معدل ألخطاء التي يرتكبها المستخدمون المدربون. عدد إطارات المساعدة.

Requirements Measures

Property	***Measure***
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

تدابير المتطلبات

الميزة	قياس
الموثوقية	يعني وقت الفشل احتمال عدم التوفر معدل حدوث الفشل
المتانة	حان الوقت إلعادة التشغيل بعد الفشل نسبة ألحداث المسببة للفشل. احتمال تلف البيانات عند الفشل.
قابلية التنقل	النسبة المئوية للبيانات المعتمدة على الهدف. عدد ألنظمة المستهدفة.

User Requirements

A statement in natural language plus diagrams of the services the system provides and its operational constraints. Based on information from Client, and written for him (or even by him)

- Should be understood by the user, and should not address design and implementation aspects
- Should focus on the key facilities required

<u>Problems with requirements written in natural language:</u>

- Lack of clarity, ambiguity, various interpretations possible
- Confusion, lack of separation between different types of requirements
- Mixture of several requirements in the same statement
- Hard to modularize and thus hard to find connections between requirements

متطلبات المستخدم

1. بيان باللغة الطبيعية بالإضافة إلى الرسوم البيانية للخدمات التي يوفرها النظام والقيود التشغيلية الخاصة به .ا
2. معلومات من العميل ، ومكتوبة له (أو حتى من قبله)

3. يجب أن يكون مفهوماً" من قبل المستخدم ، وأن يتناول جوانب التصميم والتنفيذ.
4. ينبغي التركيز على المرافق الرئيسية المطلوبة.

<u>مشاكل المتطلبات المكتوبة بلغة طبيعية:</u>

- عدم الوضوح والغموض والتفسيرات المختلفة الممكنة.
- االرتباك وعدم الفصل بين أنواع المتطلبات المختلفة.
- خليط من عدة متطلبات في نفس البيان.
- من الصعب أن تكون نمطي"ا وبالتالي يصعب إيجاد روابط بين المتطلبات.

Guidelines for writing requirements:

- Create and use a standard format for the entire software requirements specification
- Highlight important parts of the requirement statements
- Use consistently the language (difference between “should” and “shall”)

إرشادات لمتطلبات الكتابة:

* إنشاء واستخدام تنسيق قياسي لمواصفات متطلبات البرنامج بالكامل.
*تسليط الضوء على أجزاء مهمة من بيانات المتطلبات.
*استخدم اللغة باستمرار (الفرق بين "ينبغي "و "يتعين)“

Editor example: detailed user requirement

3.5.1 Adding nodes to a design

3.5.1.1 **The editor shall provide a facility for users to add nodes of a specified type to their design.**

3.5.1.2 The sequence of actions to add a node should be as follows:

1. The user should select the type of node to be added.

2. The user should move the cursor to the approximate node position in the diagram and indicate that the node symbol should be added at that point.

3. The user should then drag the node symbol to its final position.

Rationale: The user is the best person to decide where to position a node on the diagram. This approach gives the user direct control over node type selection and positioning.

Specification: ECLIPSE/WS/Tools/DE/FS. Section 3.5.1

مثال المحرر: متطلبات المستخدم التفصيلية

3.5.1

System Requirements

- More detailed specifications of user requirements
- Included in the contract with the client
- Used by developers as basis for design
- May be specified using various models (object-oriented models, data-flow diagrams, formal specs, etc.)
- Should indicateWHAT the system is required to do (not HOW) and under what conditions and constraints

- مواصفات أكثر تفصيال لمتطلبات المستخدم.
- مشمول في العقد المبرم مع العميل.
- يستخدمه المطورون كأساس للتصميم.

- يمكن تحديدها باستخدام نماذج مختلفة (النماذج الموجهة للكائنات ، الرسوم البيانية لتدفق البيانات ، المواصفات الرسمية ، إلخ).
- يجب أن تشير إلى ما هو مطلوب من النظام القيام به (وليس كيف) وتحت أي شروط وقيود.

Editor example: System requirement

ECLIPSE/Workstation/Tools/DE/ FS/ 3.5 .1

Functi on Add node

Description Adds a node to an exis ting design. The user selects the type of node, and its position. When added to the design, the node becometshe current selection. The user chooses the node position by moving the cursor to the area where the node is added.

Inputs Node type, Node position, Design identifier.

Source Node type and Node position are input by the user, Desi gn identifier from the database.

Outputs Design identifier.

Destination The design database. The design is commi tted to the database on completion of the operation.

Requires Design graph rooted at input design identifier.

Pre-condition The design is open and displayed on the user's screen.

Post-condition The design is unchanged apart from the addition of a node of the specified type at the given position.

Si de-effects None

Editor example: System requirement

- **الوظيفة: إضافة عقدة**

- **الوصف: يضيف عقدة إلى تصميم موجود. يحدد المستخدم نوع العقدة وموضعها. عند إضافتها إلى التصميم ، تصبح العقدة هي التحديد الحالي. يختار المستخدم موقع العقدة عن طريق تحريك المؤشر إلى المنطقة التي يتم فيها إضافة العقدة.**
- **المدخالت: نوع العقدة ، موضع العقدة ، معرف التصميم.**

- المصدر :يتم إدخال نوع العقدة وموضع العقدة بواسطة المستخدم ، ومعرف التصميم من قاعدة البيانات.

- المخرجات :مع ّرف التصميم.

- الوجهة :قاعدة بيانات التصميم ، يلتزم التصميم بقاعدة البيانات عند إتمام العملية.

- يتطلب :تصميم رسم بياني متجذر في مع ّرف تصميم االدخال.

- الشروط المسبقة :التصميم مفتوح ومعروض على شاشة المستخدم.

- الحالة اللالحقة :التصميم لم يتغير بصرف النظر عن االضافة عقدة من النوع المحدد في الموضع المحدد
- آالثار الجانبية :ال يوجد.

H.W.

- What is the difference between requirements analysis and specification?
- Why is it hard to define and specify requirements?
- What is the difference between functional and non-functional requirements?

- ما الفرق بين تحليل المتطلبات والمواصفات؟
- لماذا يصعب تحديد المتطلبات وتحديدها؟
- ما هو الفرق بين المتطلبات الوظيفية وغير الوظيفية؟

Software Engineering

Lecture 4

Functional Modeling

2022-2021

Systems, Models, and Views

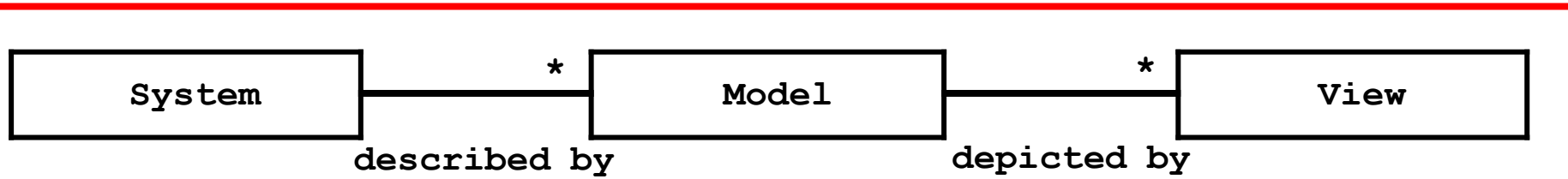
- A model is an abstraction describing system or a subset of a system
- A view depicts selected aspects of a model
- A notation is a set of graphical or textual rules for representing views

- Views and models of a single system may overlap each other

أنظمة ونماذج وطرق عرض:

- النموذج هو تجريد يصف نظامًا أو مجموعة فرعية من النظام.
- طريقة عرض تصور جوانب مختارة من النموذج.
- التدوين هو مجموعة من القواعد الرسومية أو النصية لتمثيل طرق العرض
- قد تتداخل طرق العرض والنماذج الخاصة بنظام واحد مع بعضها البعض.

Models, Views, and Systems



Why model software?

Software is already an abstraction: why model software?

- Software is getting larger, not smaller

  Windows XP: 40 million line codes

  WindowsVista : 50 million

  Windows 7: 40 million

  Window10 : 50 million

  A single programmer cannot manage this amount of code in its entirety.

- Code is often not directly understandable by developers who did not participate in the development
- We need simpler representations for complex systems

  Modeling is a mean for dealing with complexity

البرمجيات هي بالفعل فكرة مجردة :لماذا البرمجيات النموذجية؟

- البرنامج يكبر وليس أصغر

- Windows XP: 40 million line codes
- WindowsVista : 50 million
- Windows 7: 40 million
- Window10 : 50 million

لا يستطيع مبرمج واحد إدارة هذا القدر من التعليمات البرمجية بالكامل.

- في كثير من الأحيان لا يمكن فهم الكود بشكل مباشر من قبل المطورين الذين لم يشاركوا في التطوير.

- نحتاج إلى تمثيلات أبسط للأنظمة المعقدة.

النمذجة وسيلة للتعامل مع التعقيد

Concepts and Phenomena

- *Phenomenon*: An object in the world of a domain as you perceive it, for example:

  You are attending the lecture.

- *Concept*: Describes the properties of phenomena that are common, for example:

  Lectures on software engineering

- A concept is a 3-tuple:
  - Its *Name* distinguishes it from other concepts.
  - Its *Purpose* are the properties that determine if a phenomenon is a member of a concept.
  - Its *Members* are the phenomena which are part of the concept.

المفاهيم والظواهر

- ظاهرة :كائن في عالم المجال كما تدركه ، على سبيل المثال:
أنت تحضر المحاضرة.

- المفهوم :يصف خصائص الظواهر الشائعة ، على سبيل المثال:
محاضرات عن هندسة البرمجيات

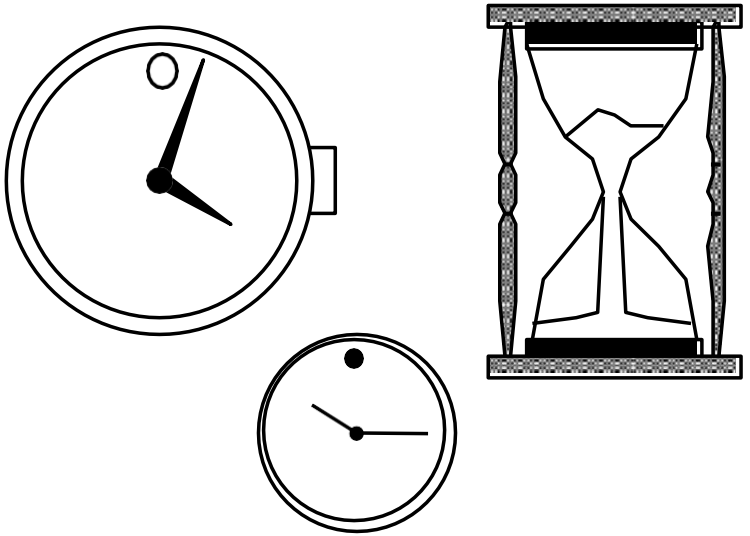
- المفهوم عبارة عن 3 مجموعات:
- اسمها يميزها عن غيرها من المفاهيم.
- الغرض منه هو الخصائص التي تحدد ما إذا كانت الظاهرة عضوًا في المفهوم.
- أعضائها هم الظواهر التي هي جزء من المفهوم.

Concepts and Phenomena

Name **Purpose** **Members**

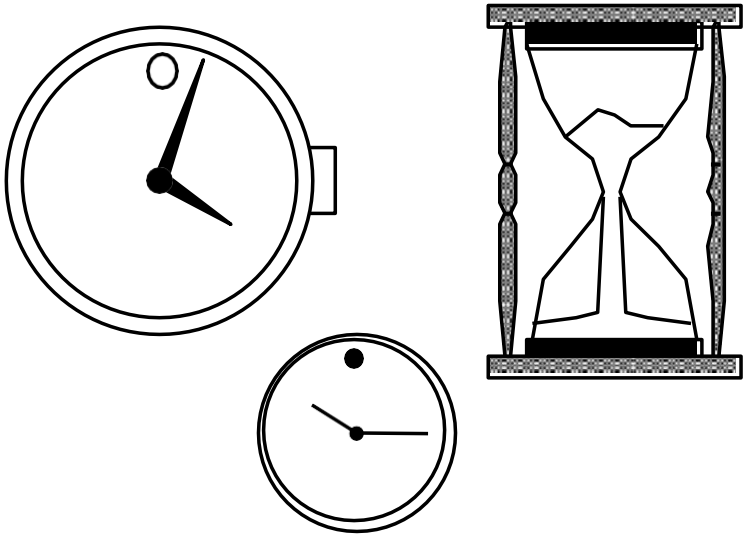
`Clock`

`A device that measures time.`



- Abstraction: Classification of phenomena into concepts
- Modeling: Development of abstractions to answer specific questions about a set of phenomena while ignoring irrelevant details.

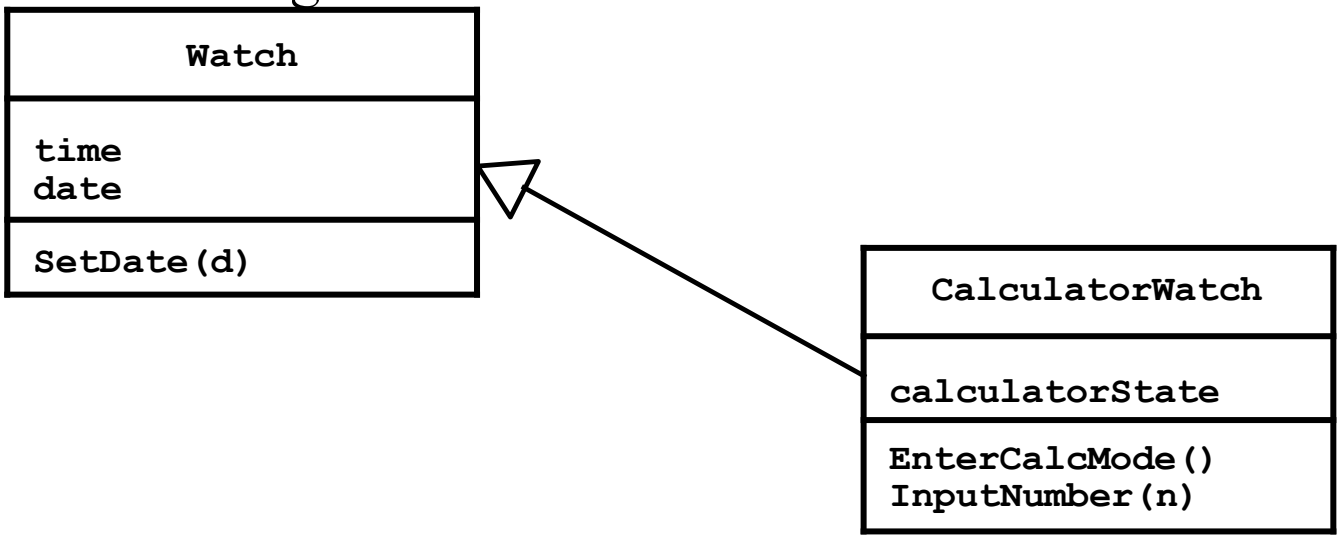
المفاهيم والظواهر

الاسم	الغرض	الأعضاء
الساعة	جهاز يقيس الوقت.	

- التجريد :تصنيف الظواهر إلى مفاهيم.
- النمذجة :تطوير التجريدات للإجابة على أسئلة محددة حول مجموعة من الظواهر مع تجاهل التفاصيل غير ذات الصلة.

Class

- Class:
  - An abstraction in the context of object-oriented languages
- Like an abstract data type, a class encapsulates both state (variables) and behavior (methods)
- Unlike abstract data types, classes can be defined in terms of other classes using inheritance



Class

- صنف:

تجريد في سياق اللغات الشيئية.

- مثل نوع البيانات المجردة ، يشمل الفصل كلا من الحالة (المتغيرات) والسلوك (الطرق).
- على عكس أنواع البيانات المجردة ، يمكن تعريف الفئات من حيث الفئات الأخرى باستخدام الوراثة.

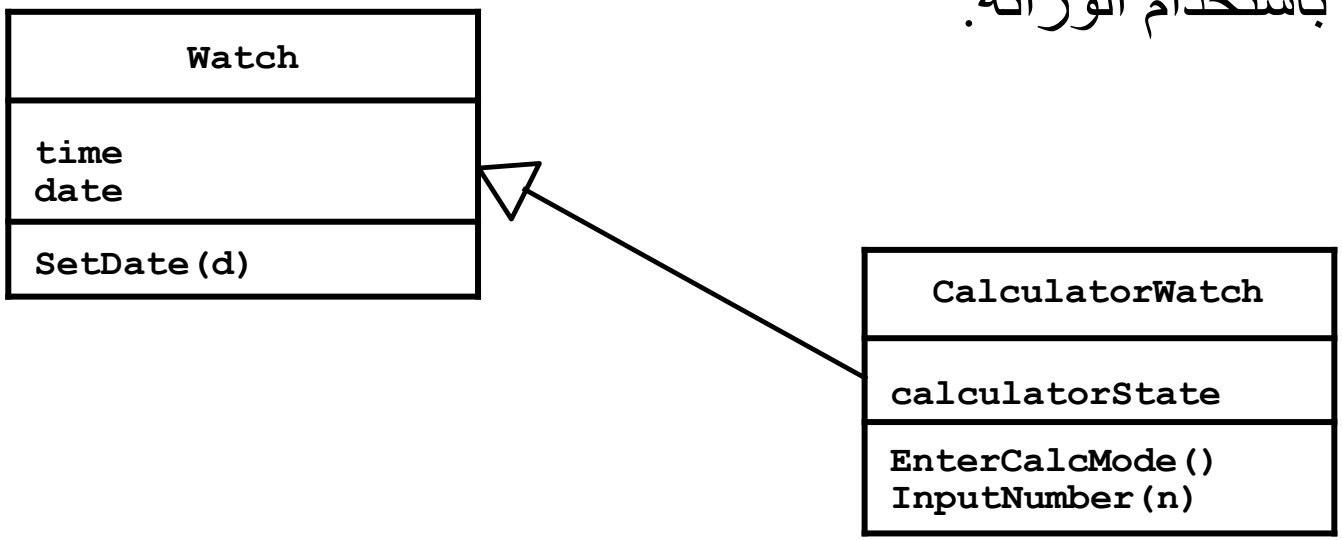


Diagram Types

- Use case diagrams

  Describe the functional behavior of the system as seen by the user.

- Class diagrams

  Describe the static structure of the system: Objects,Attributes, and Associations.

- Sequence diagrams

  Describe the dynamic behavior between actors and the system and between objects of the system.

- Statechart diagrams

  Describe the dynamic behavior of an individual object as a finite state machine.

- Activity diagrams

  Model the dynamic behavior of a system, in particular the workflow, i.e. a flowchart.

أنواع المخططات

- Use case diagrams:
وصف السلوك الوظيفي للنظام كما يراه المستخدم.

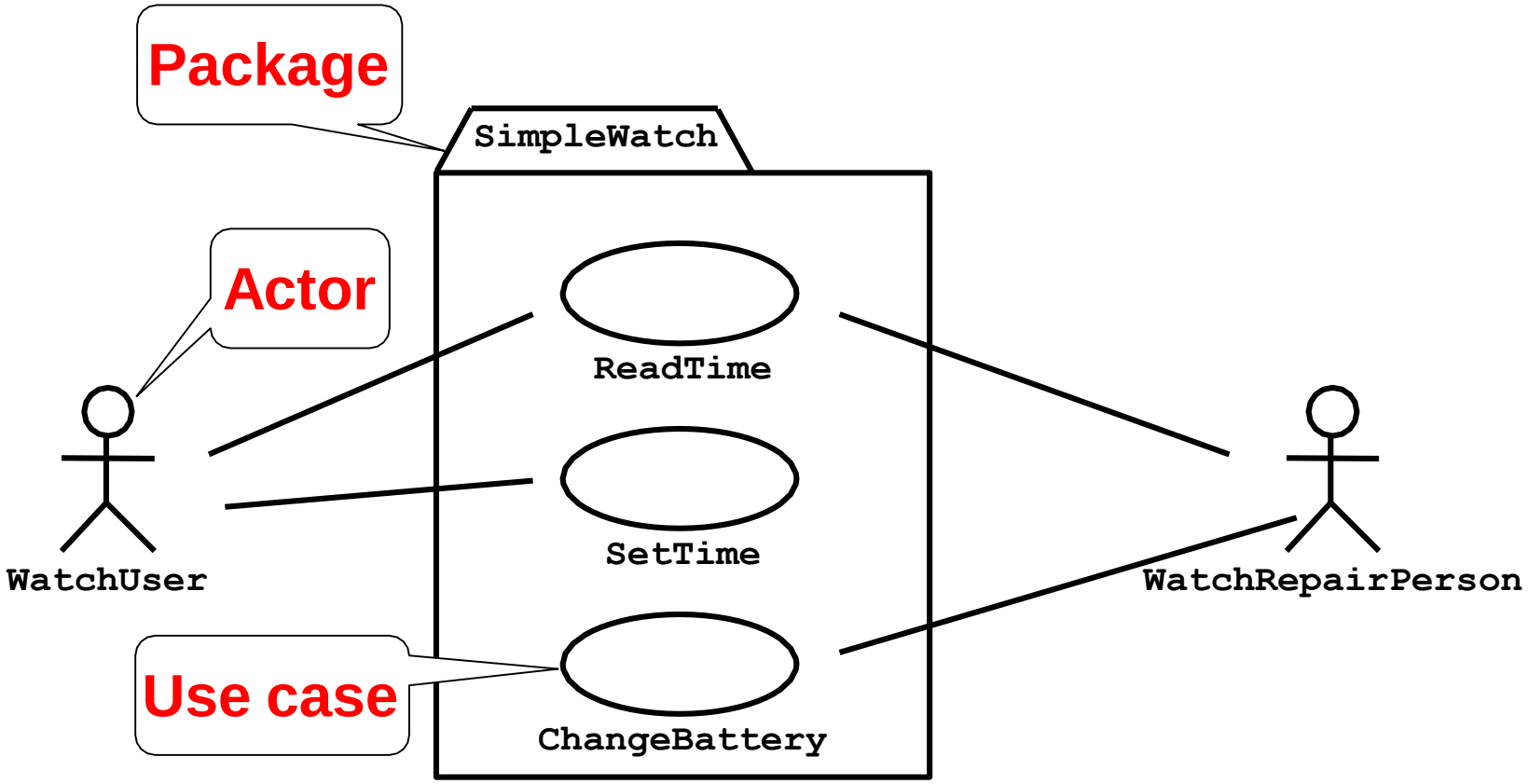
- Class diagrams
صف البنية الثابتة للنظام :الكائنات والسمات والجمعيات.

- Sequence diagrams
وصف السلوك الديناميكي بين الممثلين والنظام وبين كائنات النظام.

- Statechart diagrams
وصف السلوك الديناميكي لكائن فردي كآلة حالة محدودة.

- Activity diagrams
نموذج السلوك الديناميكي للنظام ، ولا سيما سير العمل ، أي مخطط انسيابي.

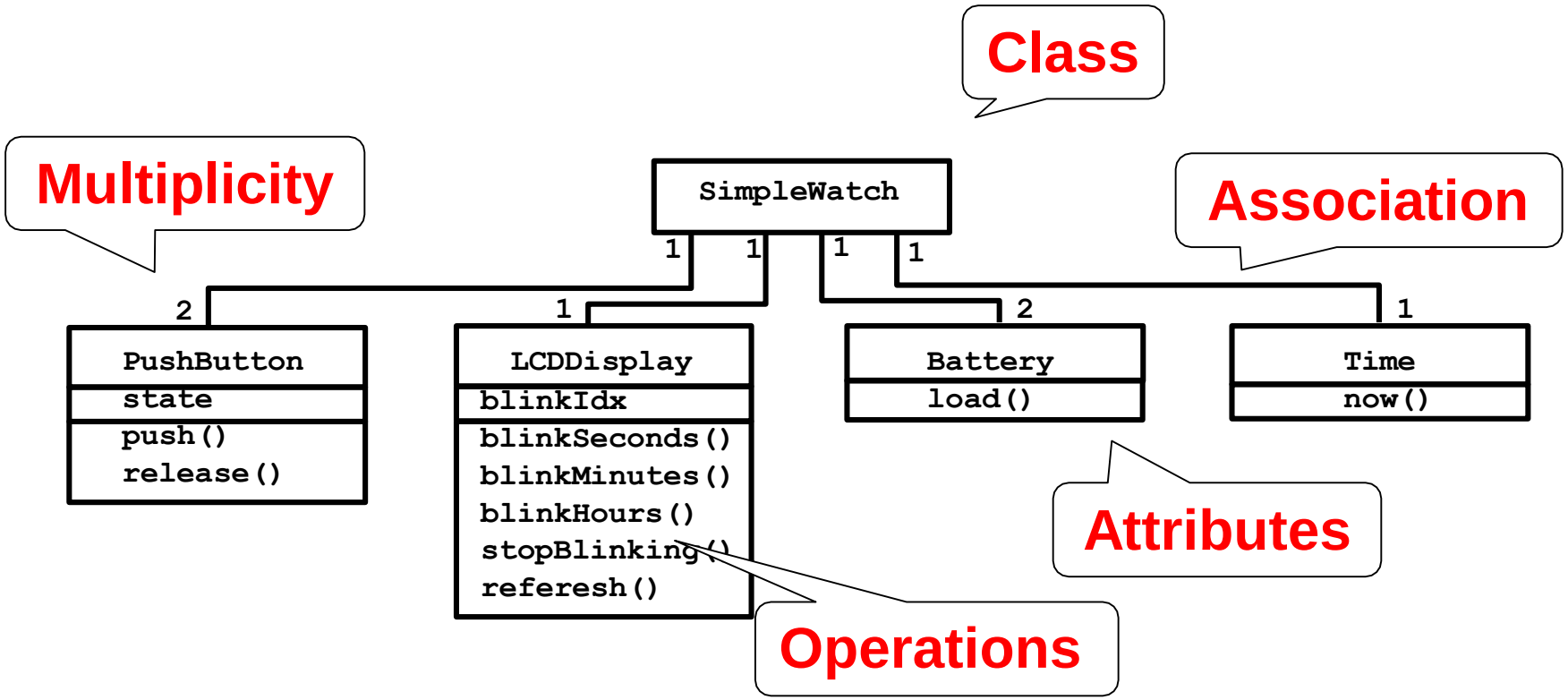
Use Case Diagrams



Use case diagrams represent the functionality of the system from user's point of view

Use case diagrams يمثل وظائف النظام من وجهة نظر المستخدم.

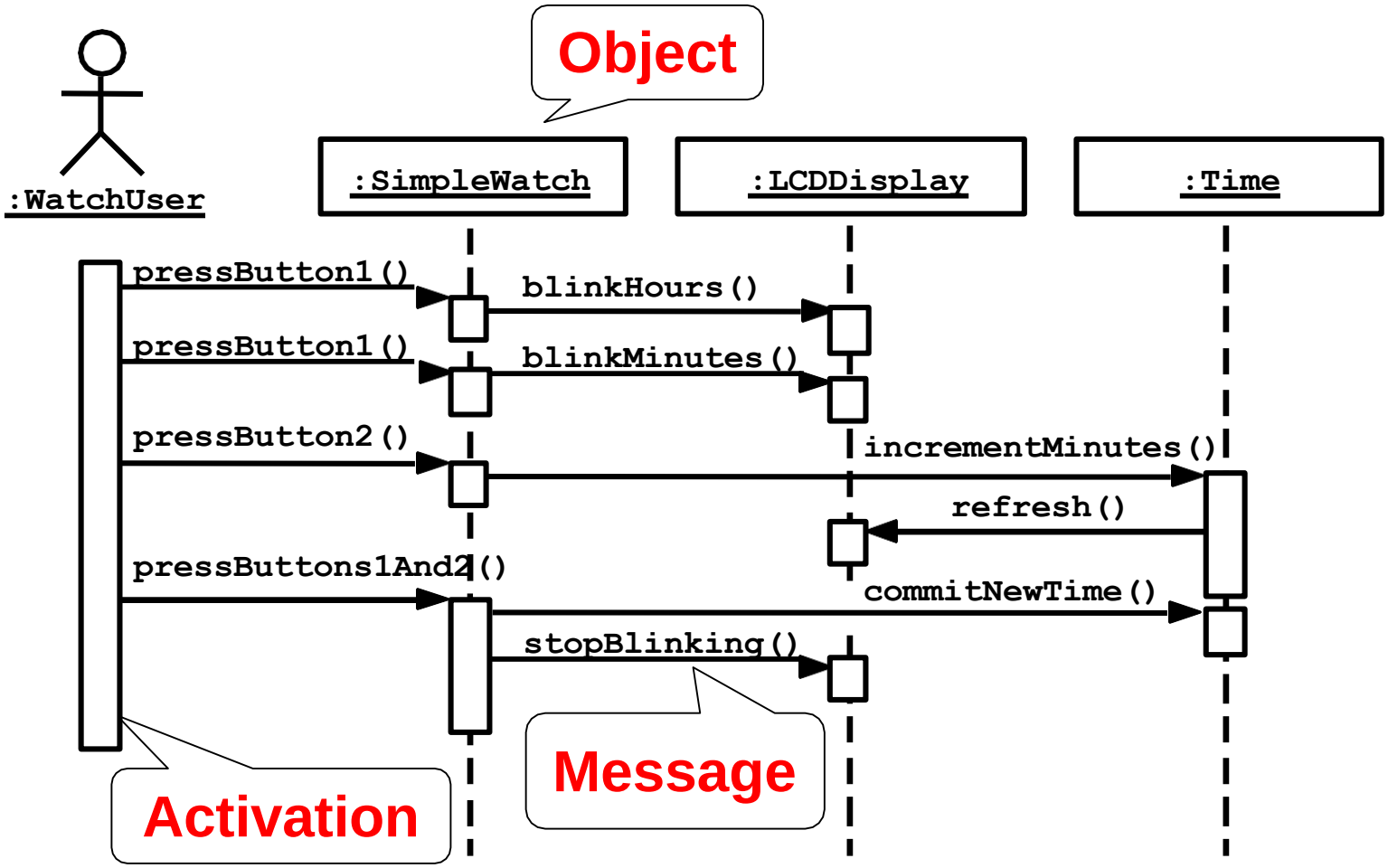
Class Diagrams



Class diagrams represent the structure of the system

Class diagrams تمثل هيكل النظام

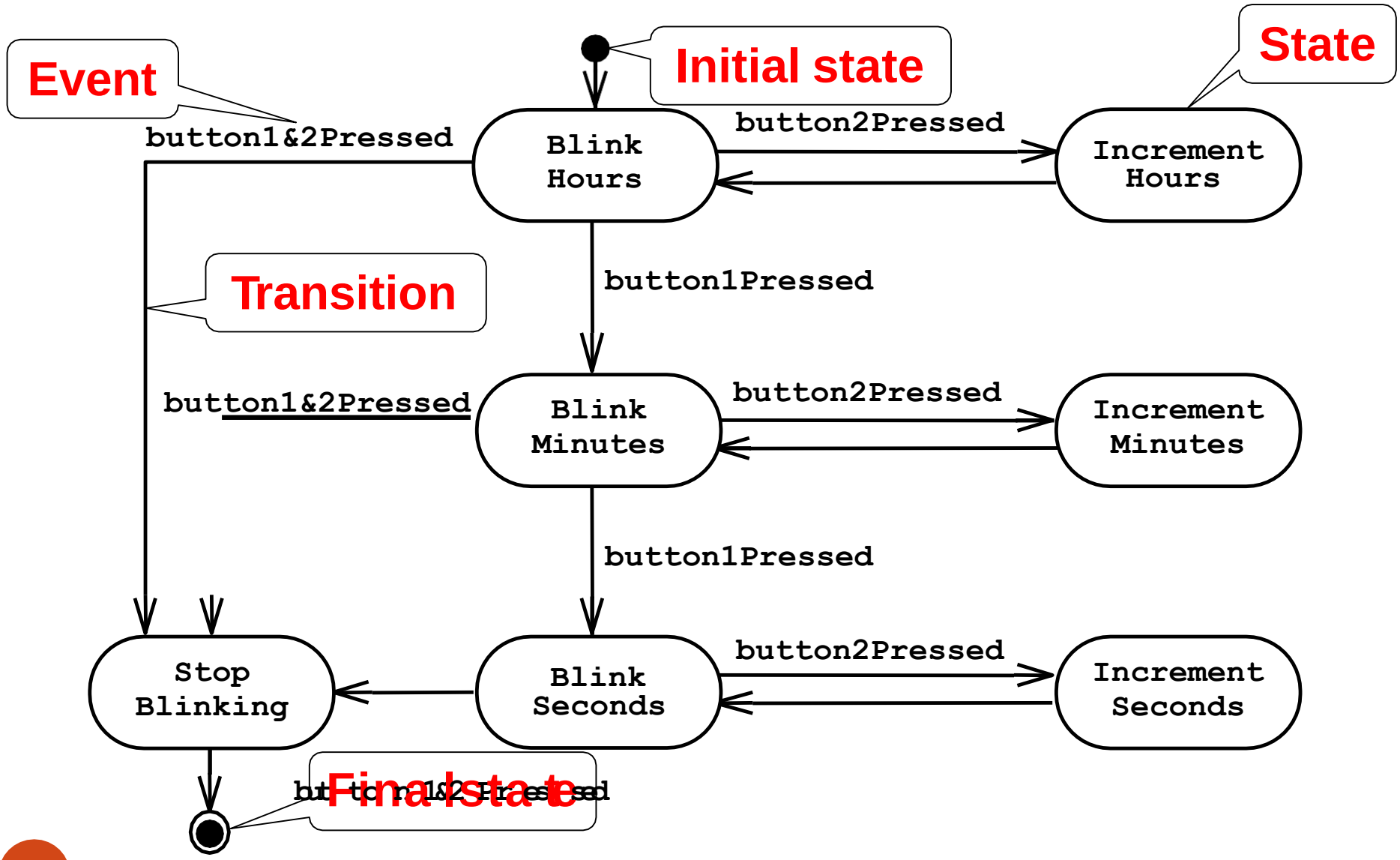
Sequence Diagram



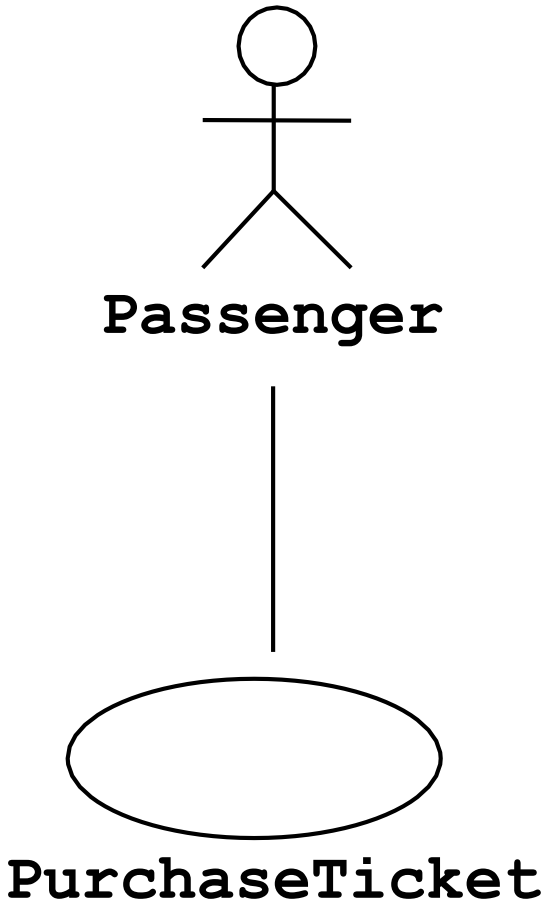
Sequence diagrams represent the behavior as interactions

Sequence diagrams يمثل السلوك كتفاعلات.

Statechart Diagrams



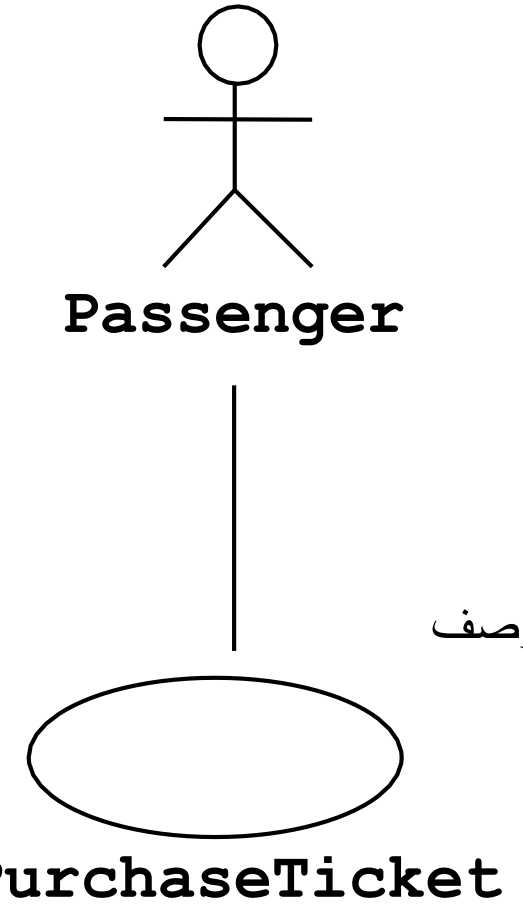
Use Case Diagrams



Used during requirements elicitation to represent external behavior

- ***Actors*** represent roles, that is, a type of user of the system
- ***Use cases*** represent a sequence of interaction for a type of functionality
- The use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment

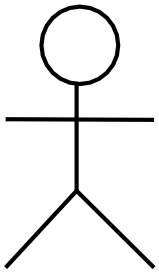
Use Case Diagrams



تُستخدم أثناء استنباط المتطلبات لتمثيل السلوك الخارجي:

- تمثل الجهات الفاعلة الأدوار ، أي نوع من مستخدمي النظام.
- تمثل حالات الاستخدام سلسلة من التفاعلات لنوع من الوظائف.
- نموذج حالة الاستخدام هو مجموعة جميع حالات الاستخدام. إنه وصف كامل لوظائف النظام وبيئته.

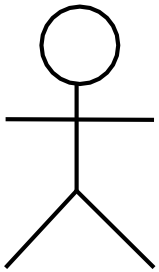
Actors



Passenger

- An actor models an external entity which communicates with the system:
  - User
  - External system
  - Physical environment
- An actor has a unique name and an optional description.
- Examples:
  - Passenger:A person in the train
  - GPS satellite: Provides the system with GPS coordinates

Actors

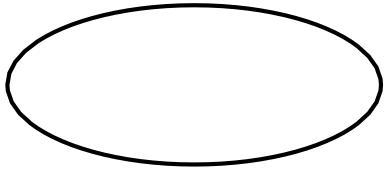


Passenger

- يمثل الممثل كيانًا خارجيًا يتواصل مع النظام:
  - مستخدم.
  - نظام خارجي.
  - بيئة فيزيائية.
- الممثل له اسم فريد ووصف اختياري.
- أمثلة:
  - الراكب: شخص في القطار.
  - قمر GPSالصناعي: يزود النظام بإحداثيات GPS.

Use Case

A use case represents a class of functionality provided by the system as an event flow.



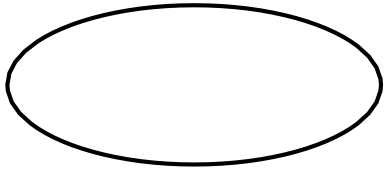
PurchaseTicket

A use case consists of:

- Unique name
- Participating actors
- Entry conditions
- Flow of events
- Exit conditions
- Special requirements

Use Case

تمثل حالة الاستخدام فئة من الوظائف التي يوفرها النظام كتدفق حدث.



PurchaseTicket

تتكون حالة الاستخدام من:

* اسم فريد.
* الجهات المشاركة.
* شروط الدخول.
* تدفق الأحداث.
* شروط الخروج.
* متطلبات خاصة.

Use Case Example

Name: `Purchase ticket`

Participating actor: `Passenger`

Entry condition:

- `Passenger` standing in front of ticket distributor.
- `Passenger` has sufficient money to purchase ticket.

Exit condition:

- `Passenger` has ticket.

Event flow:

1. `Passenger` selects the number of zones to be traveled.
2. Distributor displays the amount due.
3. `Passenger` inserts money, of at least the amount due.
4. Distributor returns change.
5. Distributor issues ticket.

Anything missing?

Exceptional cases!

Use Case Example

الاسم: شراء تذكرة

الفاعل المشارك: مسافر

شرط الدخول:
صاحب العلاقة يقف أمام موزع التذاكر.
صلحب العلاقة لديه ما يكفي من المال لشراء تذكرة.

شرط الخروج:
الراكب لديه تذكرة.

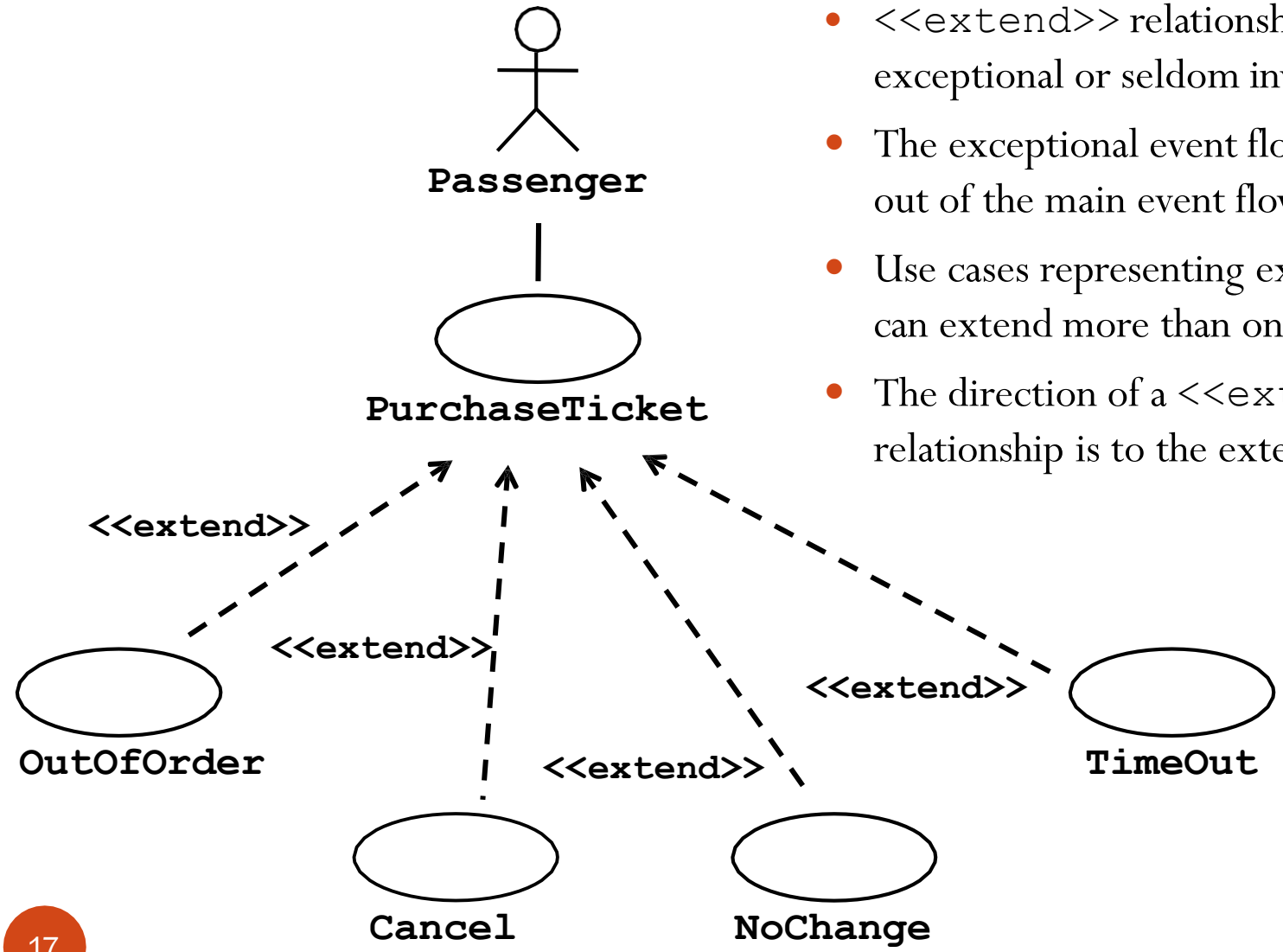
تدفق الحدث:

1. يختار المسافر عدد المناطق التي سيتم السفر إليها.
2. يعرض الموزع المبلغ المستحق.
3. يقوم المسافر بإدخال نقود لا تقل عن المبلغ المستحق
4. إرجاع الموزع التغيير.
5. يصدر الموزع التذكرة.

أي شيء مفقود؟

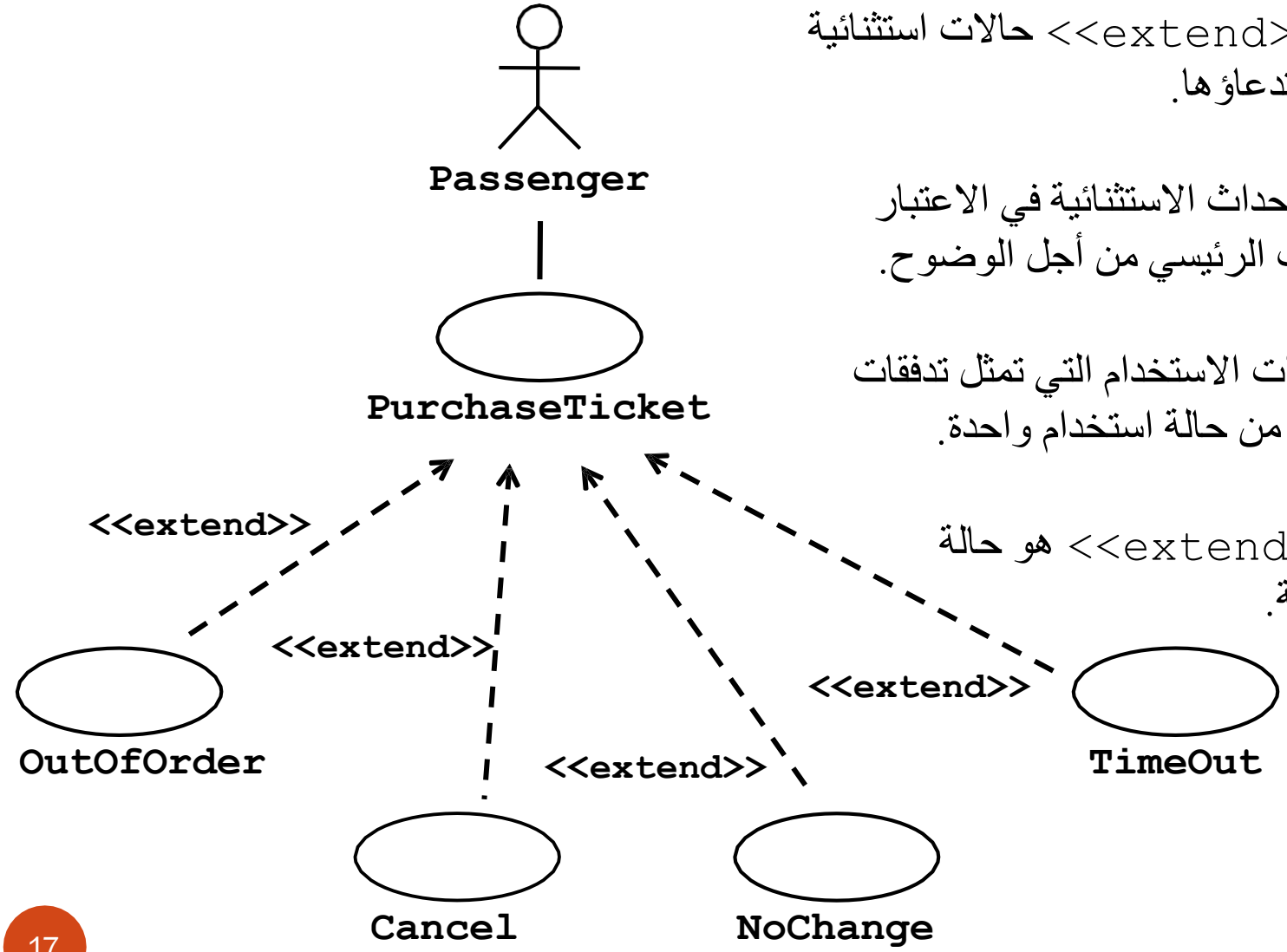
حالات إستثنائية!

The <<extend>> Relationship



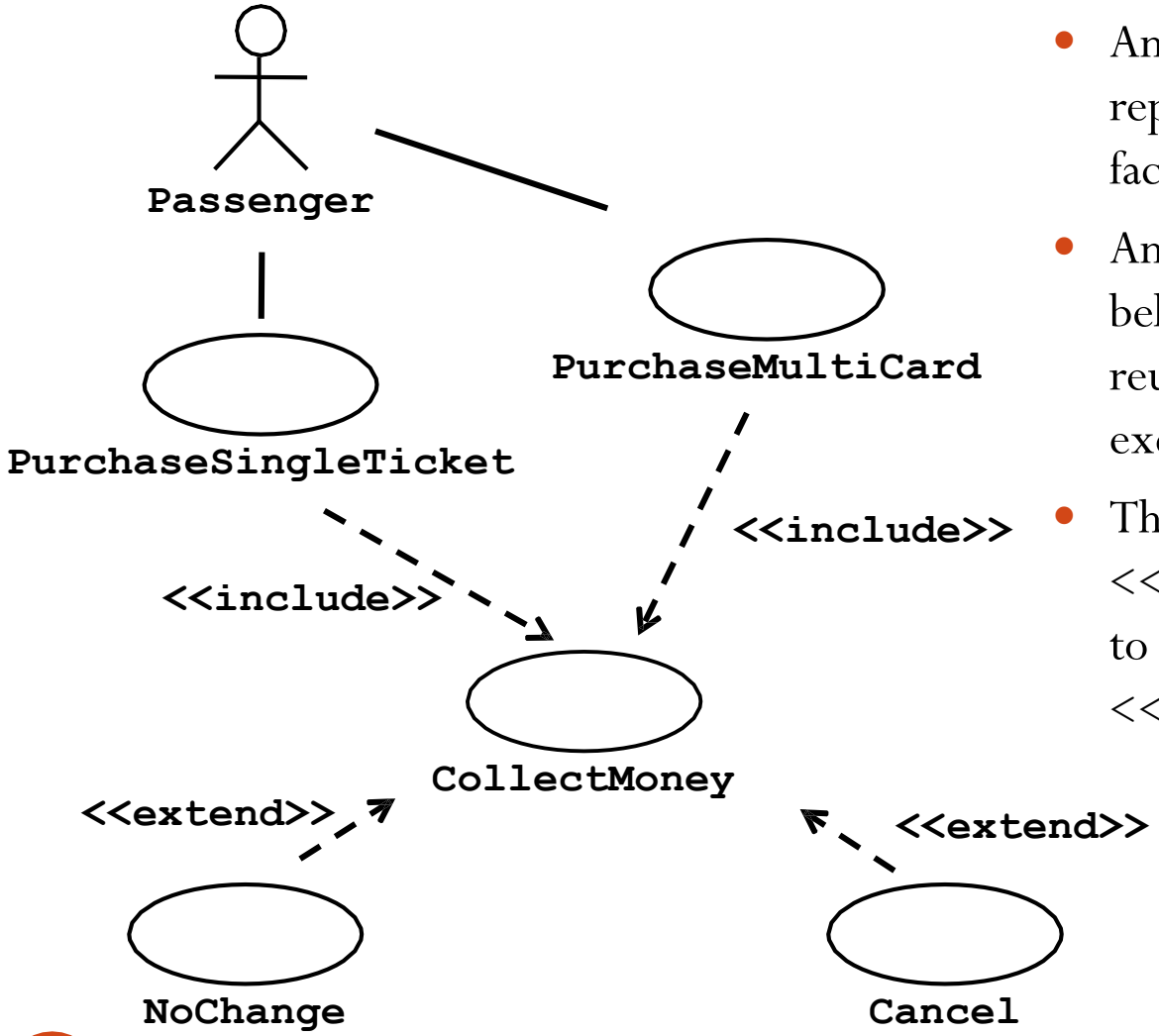
- <<extend>> relationships represent exceptional or seldom invoked cases.
- The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extend>> relationship is to the extended use case

The <<extend>> Relationship



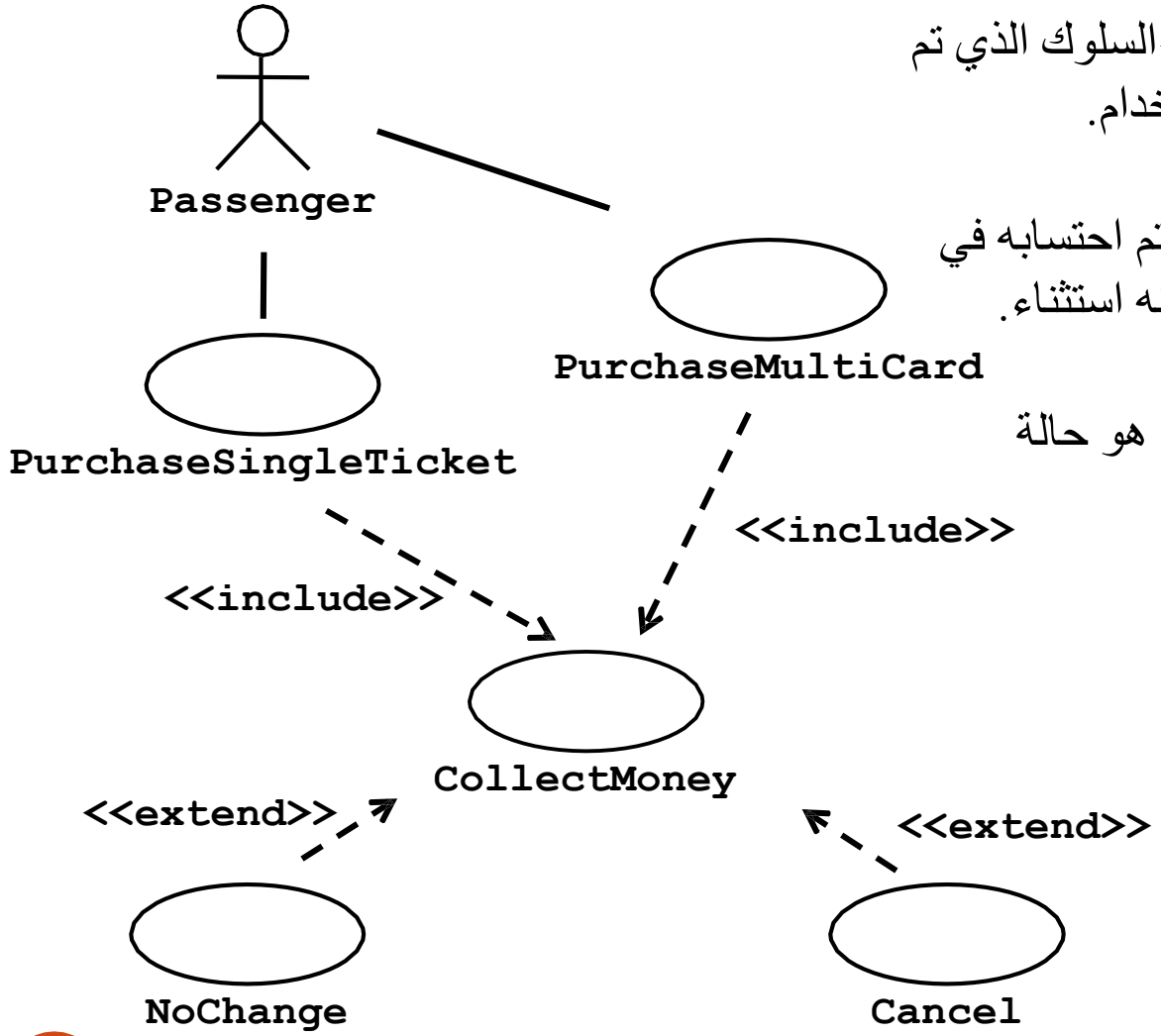
- تمثل العلاقات <<extend>> حالات استثنائية أو نادرًا ما يتم استدعاؤها.
- يتم أخذ تدفقات الأحداث الاستثنائية في الاعتبار خارج تدفق الحدث الرئيسي من أجل الوضوح.
- يمكن أن تمتد حالات الاستخدام التي تمثل تدفقات استثنائية إلى أكثر من حالة استخدام واحدة.
- اتجاه العلاقة <<extend>> هو حالة الاستخدام الموسعة.

The <<include>> Relationship



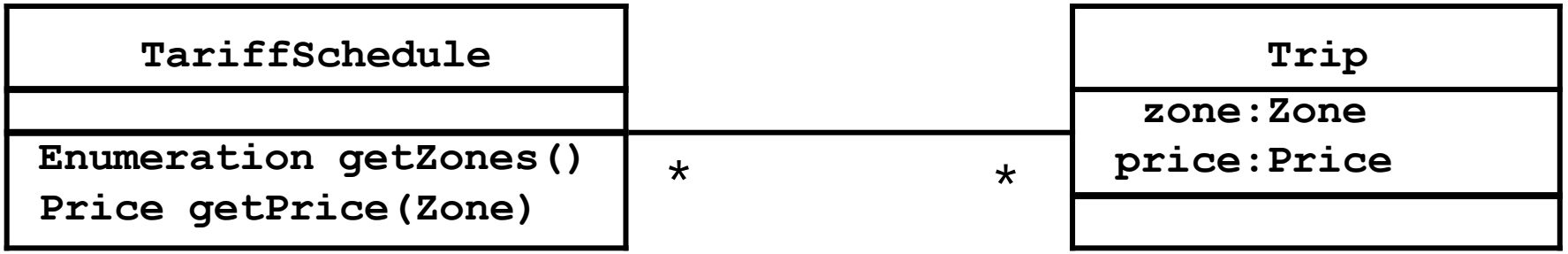
- An <<include>> relationship represents behavior that is factored out of the use case.
- An <<include>> represents behavior that is factored out for reuse, not because it is an exception.
- The direction of a <<include>> relationship is to the using use case (unlike <<extend>> relationships).

The <<include>> Relationship



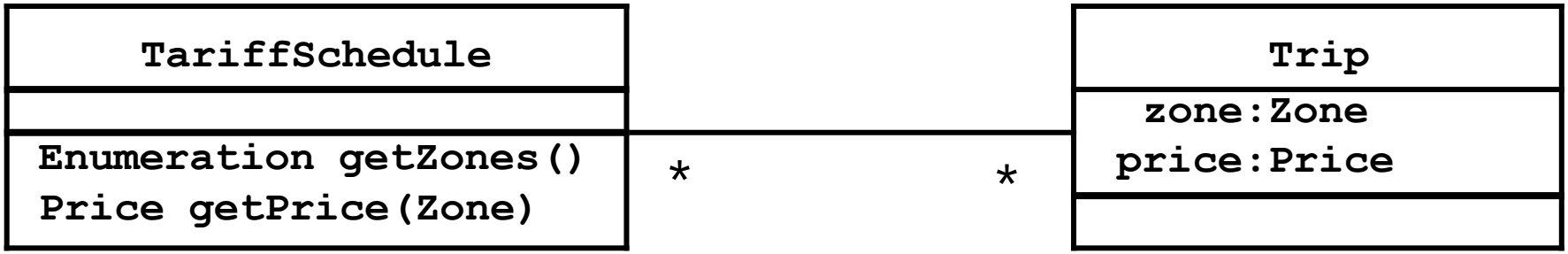
- تمثل العلاقة <<include>>السلوك الذي تم أخذه في الاعتبار خارج حالة الاستخدام.
- يمثل <<include>> سلوكًا تم احتسابه في عوامل لإعادة الاستخدام ، وليس لأنه استثناء.
- اتجاه العلاقة <<include>> هو حالة الاستخدام (على عكس علاقات <<extend>>)

Class Diagrams



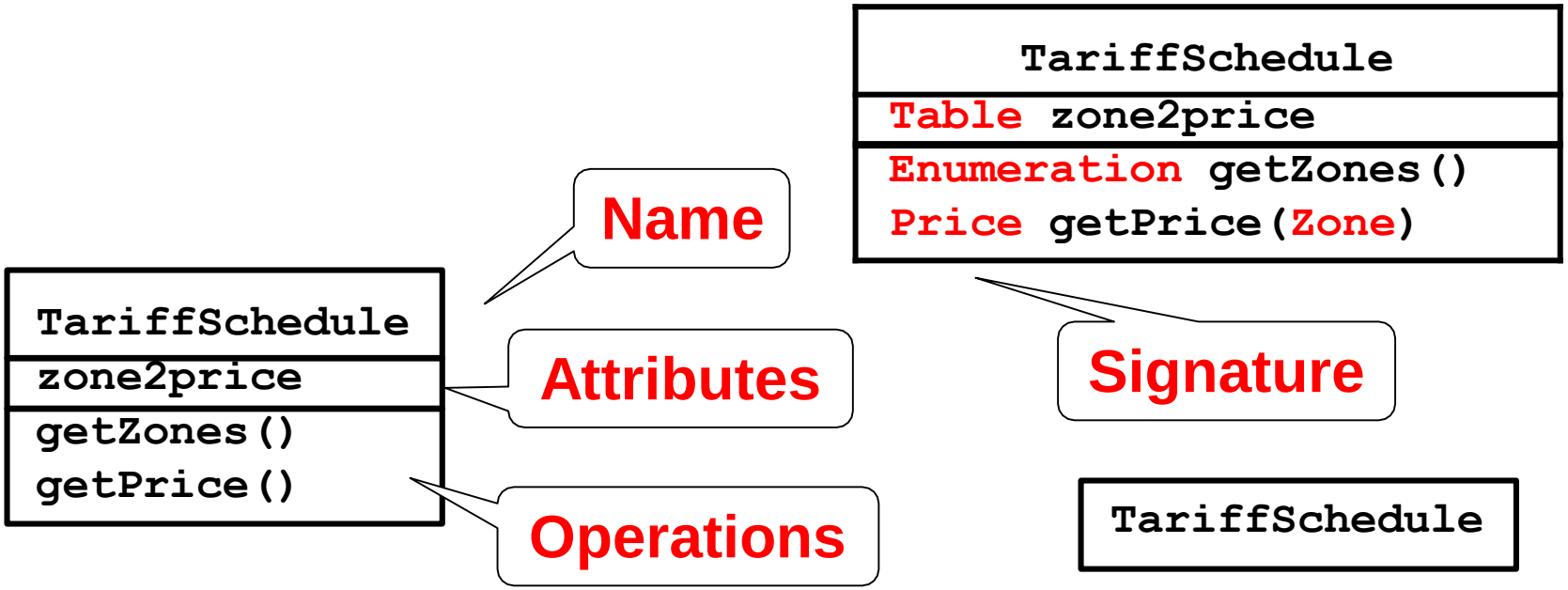
- Class diagrams represent the structure of the system.
- Class diagrams are used
  - during requirements analysis to model problem domain concepts
  - during system design to model subsystems and interfaces
  - during object design to model classes.

Class Diagrams



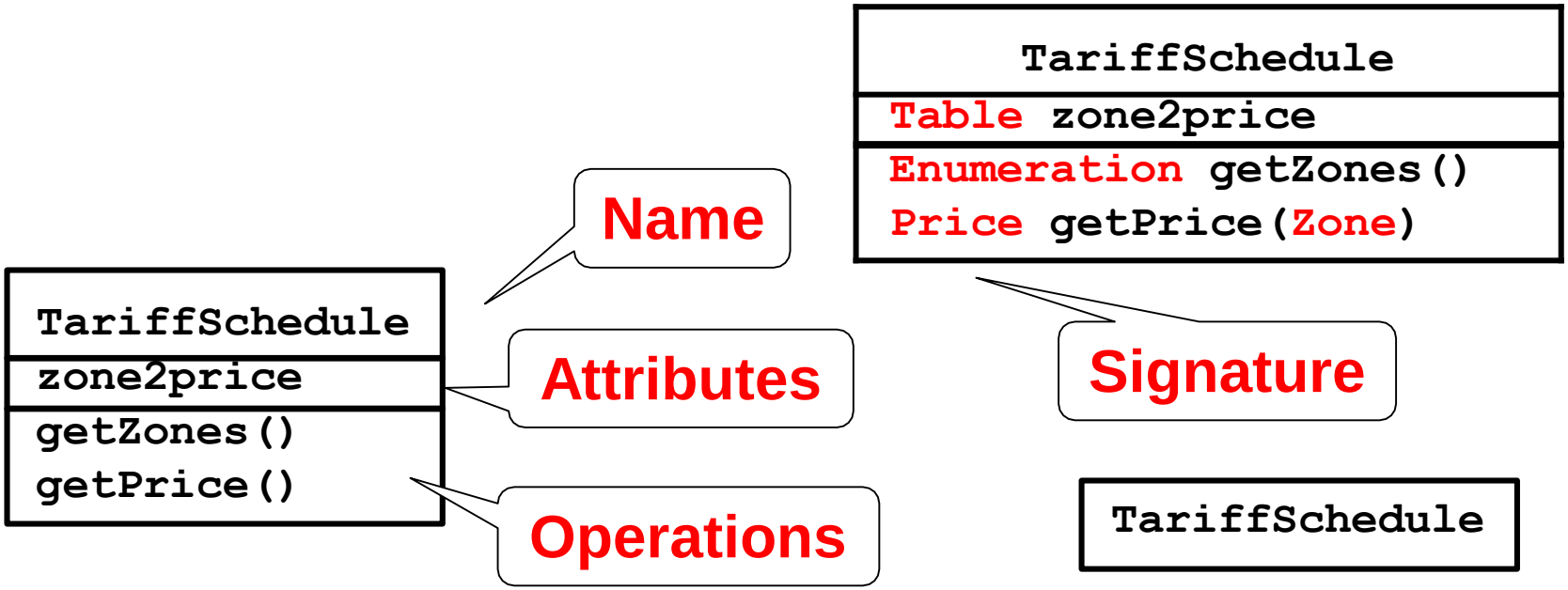
* الرسوم البيانية للصنف تمثل بنية النظام.
* يتم استخدام الرسوم البيانية للفئة.
- أثناء تحليل المتطلبات لنمذجة مفاهيم مجال المشكلة.
- أثناء تصميم النظام لنمذجة الأنظمة الفرعية والواجهات.
- أثناء تصميم الكائن إلى فئات النموذج.

Classes



- A ***class*** represent a concept.
- A class encapsulates state ***(attributes)*** and behavior ***(operations)***.
- Each attribute has a ***type***.
- Each operation has a ***signature***.
- The class name is the only mandatory information.

Classes



- فئة تمثل مفهومًا.
- فئة تلخص الحالة (السمات) والسلوك (العمليات)
- كل سمة لها نوع.
- لكل عملية توقيع.
- اسم الفصل هو المعلومات الإلزامية الوحيدة.

Actor vs. Instances

- What is the difference between an actor and a class ?
- Actor:

  An entity outside the system to be modeled, interacting with the system ("Pilot")
- Class:

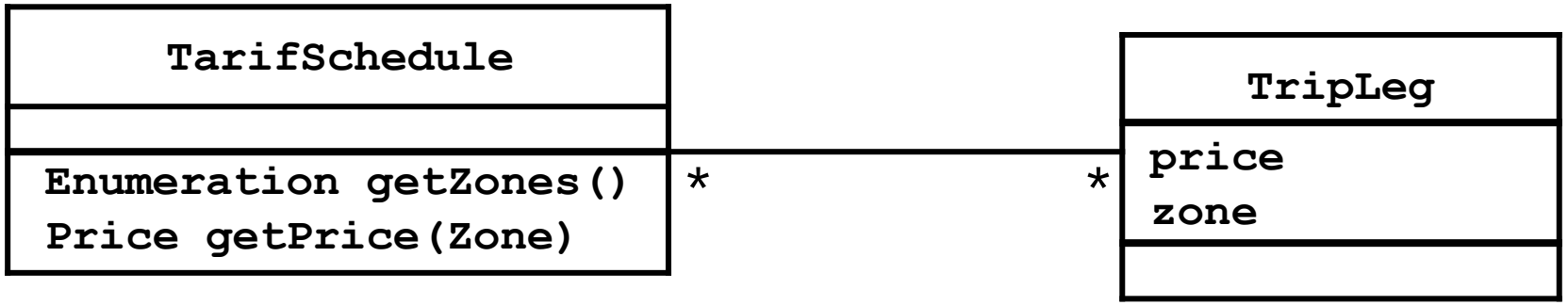
  An abstraction modeling an entity in the problem domain, inside the system to be modeled ("Cockpit")
- Object:

  A specific instance of a class ("Joe, the inspector").

Actor vs. Instances

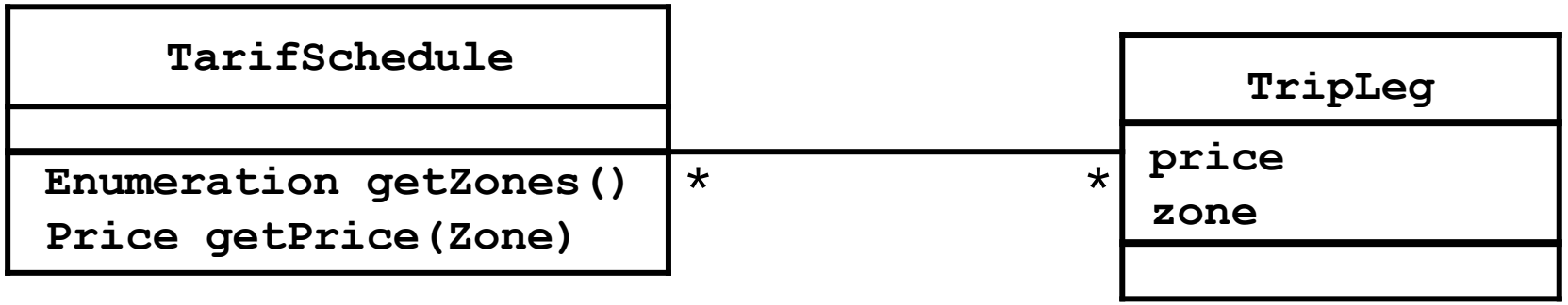
* ما هو الفرق بين الممثل والطبقة؟
* الفاعل: كيان خارج النظام يتم تشكيله ، يتفاعل مع النظام ("الطيار").
* الفئة: تجريد نمذجة كيان في مجال المشكلة ، داخل النظام المراد نمذجته ("قمرة القيادة").
* الكائن: مثيل محدد للفئة (“Joe، المفتش”).

Associations



- Associations denote relationships between classes.
- The multiplicity of an association end denotes how many objects the source object can legitimately reference.

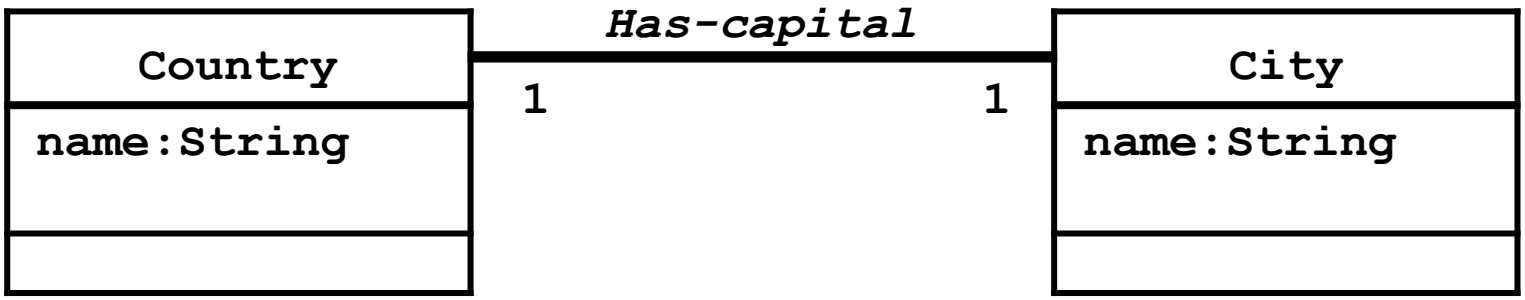
Associations



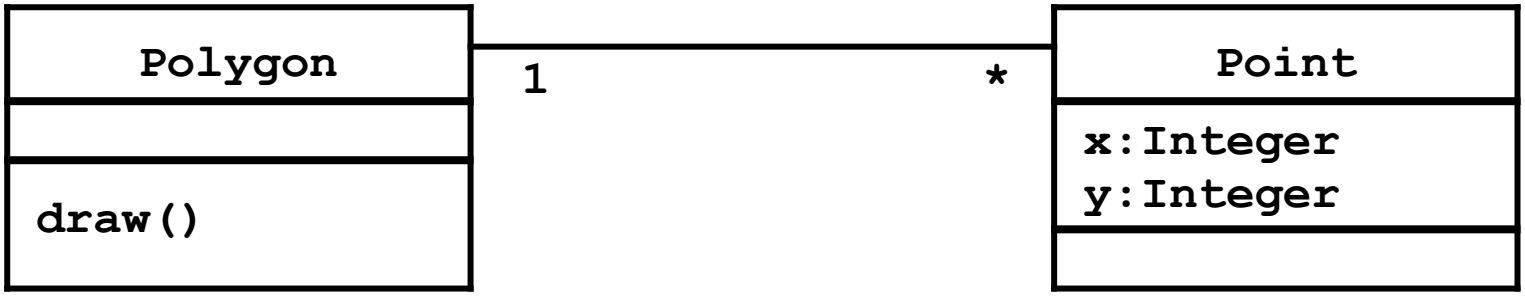
- الجمعيات تشير إلى العلاقات بين الطبقات.

- يشير تعدد نهاية الارتباط إلى عدد الكائنات التي يمكن للكائن المصدر الرجوع إليها بشكل شرعي.

1-to-1 and 1-to-Many Associations



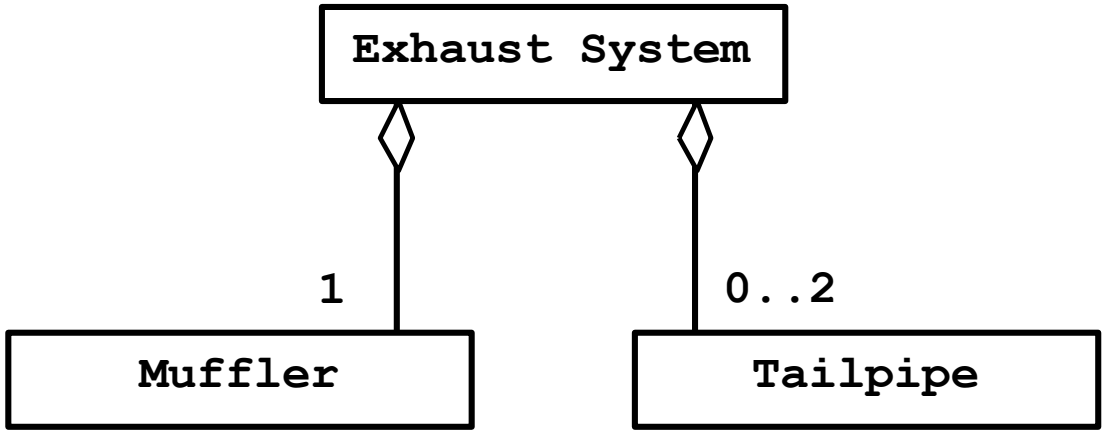
1-to-1 association



1-to-many association

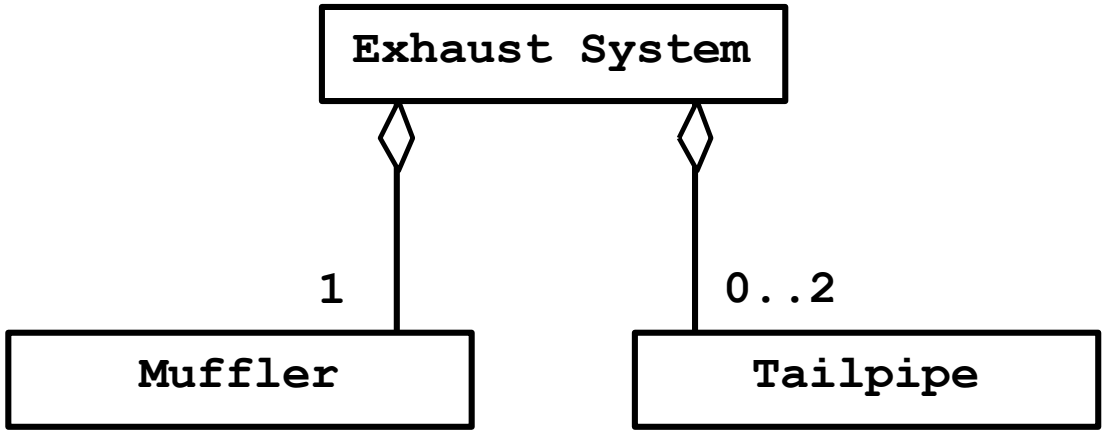
Aggregation

- An ***aggregation*** is a special case of association denoting a "consists of" hierarchy.
- The ***aggregate*** is the parent class, the ***components*** are the children class.

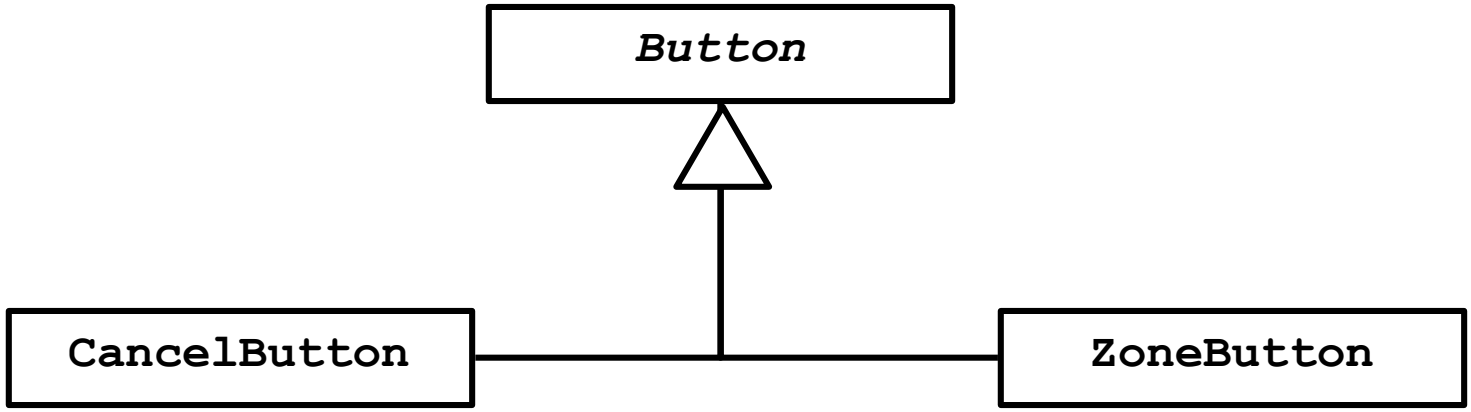


Aggregation

- التجميع هو حالة خاصة من الارتباط تدل على التسلسل الهرمي "يتكون من".
- المجموع هو فئة الأصل ، والمكونات هي فئة الأطفال.

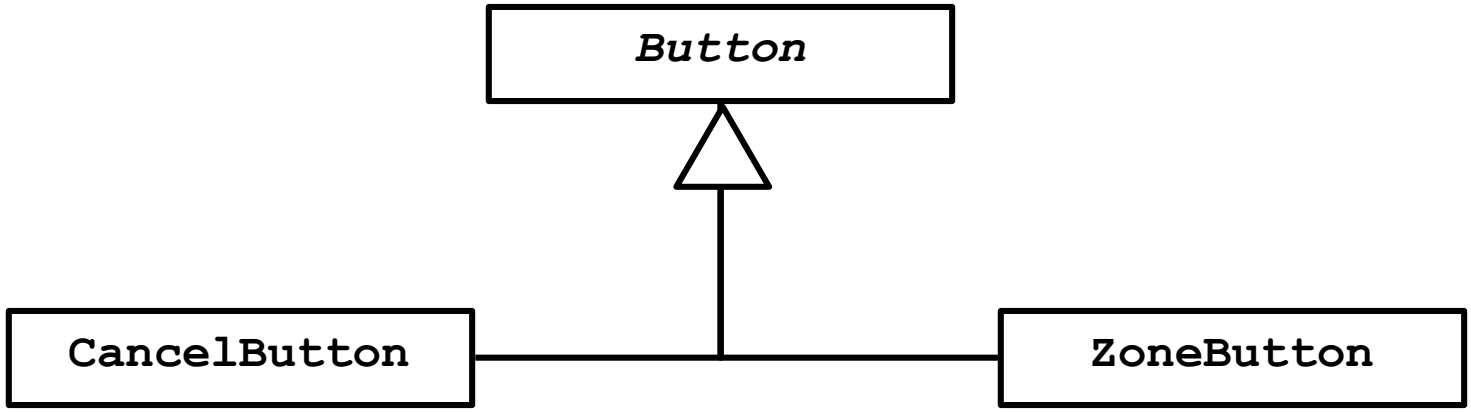


Generalization



- Generalization relationships denote inheritance between classes.
- The children classes inherit the attributes and operations of the parent class.
- Generalization simplifies the model by eliminating redundancy.

تعميم:



* علاقات التعميم تدل على الميراث بين الطبقات.
* ترث الفئات الفرعية سمات وعمليات الفئة الأصلية.
* يبسط التعميم النموذج من خلال التخلص من التكرار.

Activity Diagrams

- An activity diagram shows flow control within a system

- In activity diagram, the states are activities ("functions")
- Two types of states:
  - *Action state:*
    - Cannot be decomposed any further
    - Happens "instantaneously" with respect to the level of abstraction used in the model
  - *Activity state:*
    - Can be decomposed further
    - The activity is modeled by another activity diagram

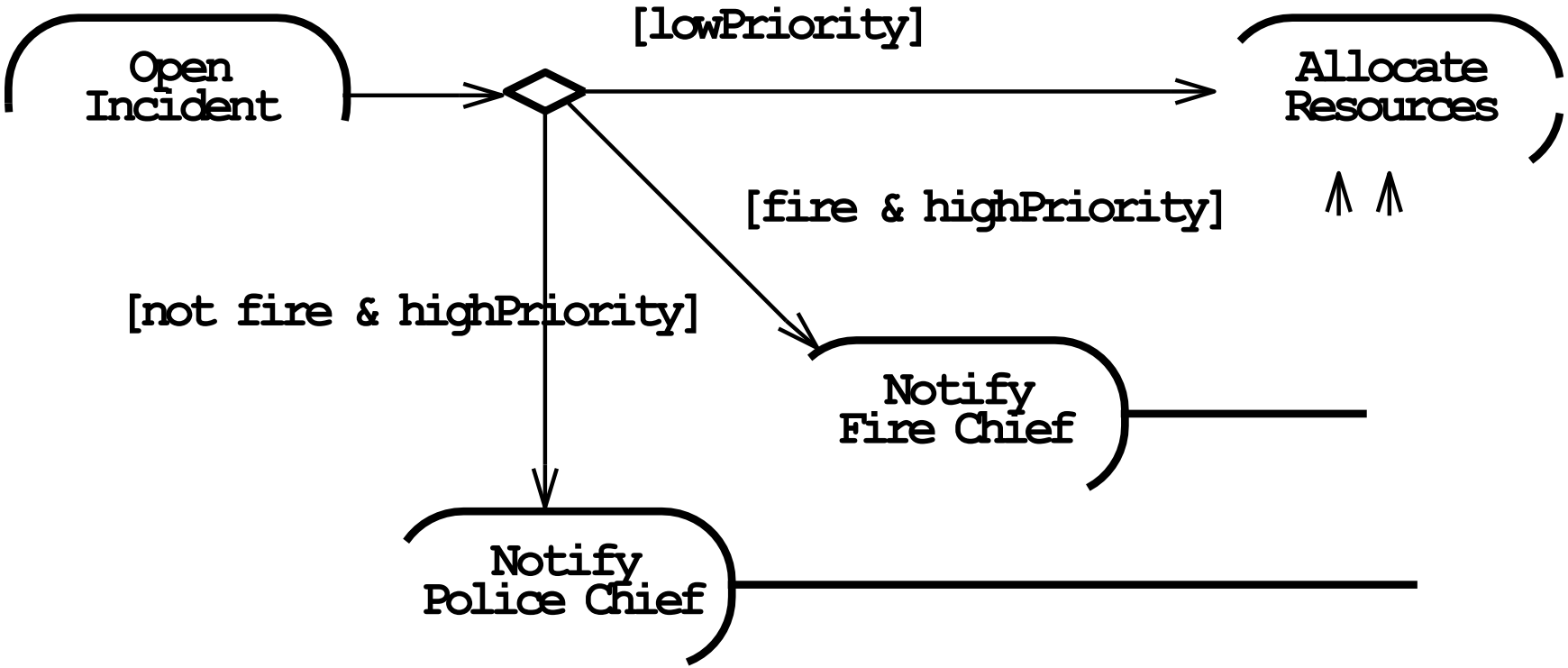
Activity Diagrams

- يُظهر مخطط النشاط التحكم في التدفق داخل النظام.

- مخطط الخمول ، الحالات عبارة عن أنشطة ("وظائف")
- نوعان من الحالات:

- حالة العمل:
  - لا يمكن أن تتحلل أكثر من ذلك.
  - يحدث "على الفور" بالنسبة لمستوى التجريد المستخدم في النموذج.
- حالة النشاط:
  - يمكن أن تتحلل أكثر.
  - تم نمذجة النشاط بواسطة مخطط نشاط آخر.

Activity Diagram: Modeling Decisions



Software Engineering

Lecture 5

System Modeling

(Structure and Behavior)

Prof. Dr. Safa A. Najim/ CIS Dept./ IT College/
Basrah Univ.

Structural models

- Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
- Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.

النماذج الهيكلية:

- تعرض النماذج الهيكلية للبرامج تنظيم النظام من حيث المكونات التي يتكون منها هذا النظام وعلاقاتها.
- قد تكون النماذج الهيكلية نماذج ثابتة توضح هيكل تصميم النظام أو نماذج ديناميكية توضح تنظيم النظام عند تنفيذه.
- تقوم بإنشاء نماذج هيكلية للنظام أثناء مناقشة وتصميم بنية النظام.

Class diagrams

- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An object class can be thought of as a general definition of one kind of system object.
- An association is a link between classes that indicates that there is some relationship between these classes.
- When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

مخططات الصنف:

- تُستخدم الرسوم البيانية للفئة عند تطوير نموذج نظام موجه للكائنات لإظهار الفئات في النظام والارتباطات بين هذه الفئات.

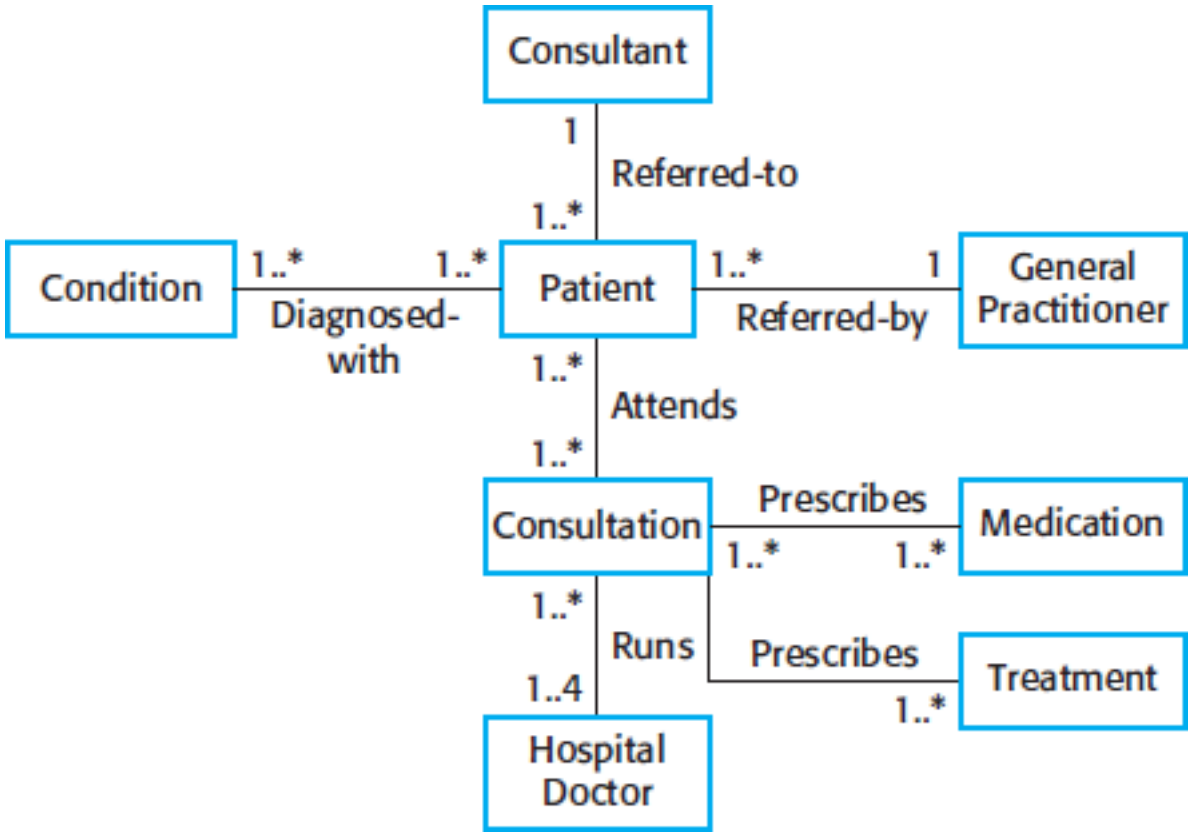
- يمكن اعتبار فئة الكائن بمثابة تعريف عام لنوع واحد من كائن النظام.

- الارتباط هو رابط بين الفئات يشير إلى وجود علاقة ما بين هذه الفئات.

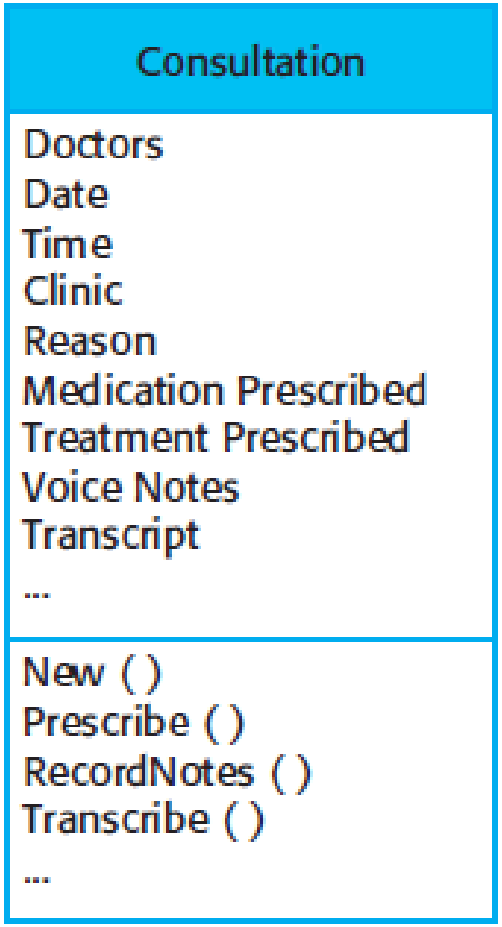
- عندما تقوم بتطوير نماذج خلال المراحل الأولى من عملية هندسة البرمجيات ، فإن الأشياء تمثل شيئًا ما في العالم الحقيقي ، مثل المريض ، أو الوصفة الطبية ، أو الطبيب ، إلخ.

Classes and associations

الفئات والجمعيات:



The Consultation class



Patient object class will have the attribute Address and you may include an operation called ChangeAddress, which is called when a patient indicates that they have moved from one address to another.

Model it?

سيكون لفئة كائن المريض عنوان السمة ويمكنك تضمين عملية تسمى تغيير العنوان ، والتي يتم استدعاؤها عندما يشير المريض إلى أنه قد انتقل من عنوان إلى آخر.

Generalization

- Generalization is an everyday technique that we use to manage complexity.
- Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes.
- This allows us to infer that different members of these classes have some common characteristics e.g. squirrels and rats are rodents.

تعميم:

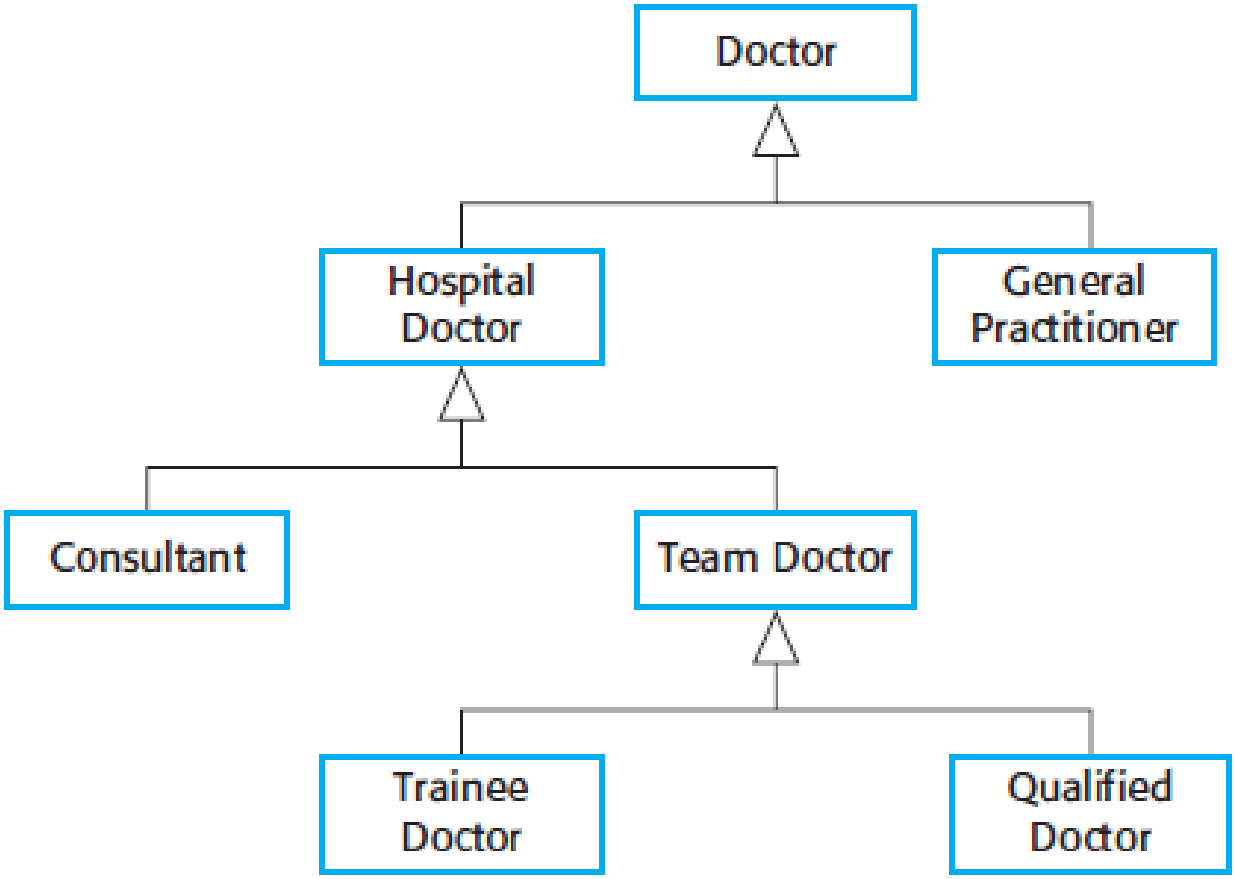
- التعميم هو تقنية يومية نستخدمها لإدارة التعقيد.

- بدلاً من معرفة الخصائص التفصيلية لكل كيان نعيشه ، نضع هذه الكيانات في فئات عامة (حيوانات ، سيارات ، منازل ، إلخ) ونتعرف على خصائص هذه الفئات.

- هذا يسمح لنا باستنتاج أن أعضاء مختلفين من هذه الفئات لديهم بعض الخصائص المشتركة على سبيل المثال السناجب والجرذان من القوارض.

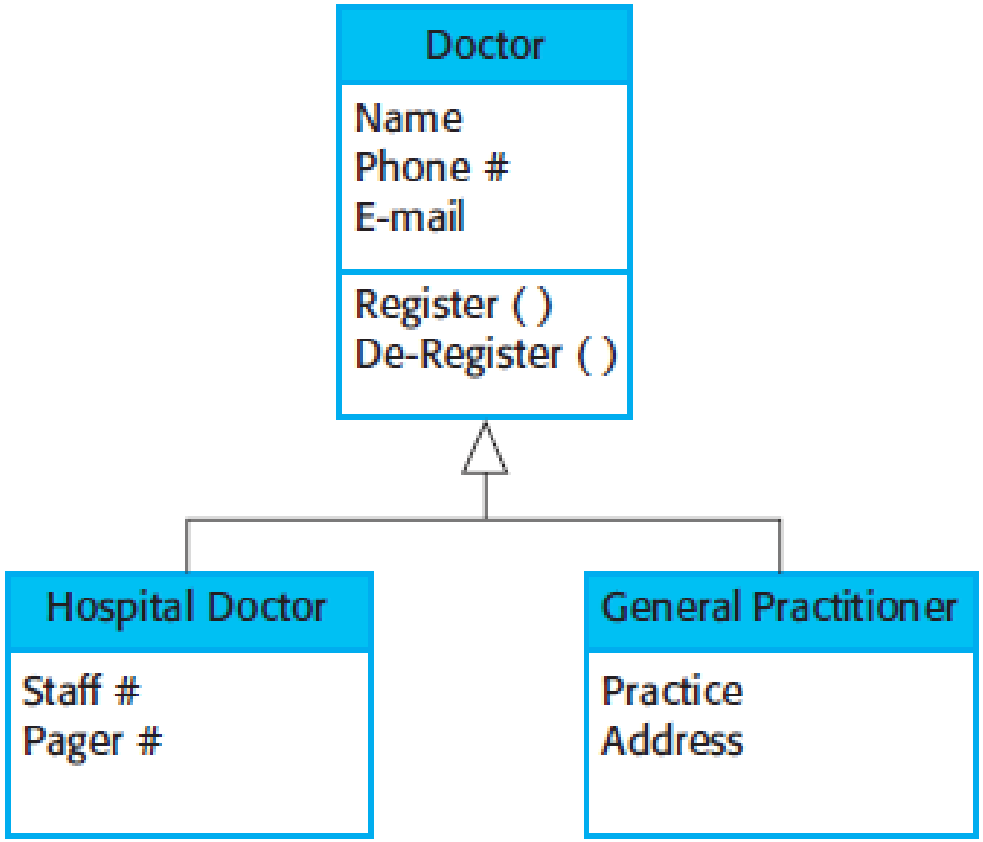
A generalization hierarchy

تسلسل هرمي للتعميم



A generalization hierarchy with added detail

تسلسل هرمي مع التفاصيل المضافة



Behavioral models

- Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment.
- You can think of these stimuli as being of two types:
  - Data Some data arrives that has to be processed by the system.
  - Events Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

النماذج السلوكية:

- النماذج السلوكية هي نماذج للسلوك الديناميكي للنظام أثناء تنفيذه .إنهم يظهرون ما يحدث أو ما يفترض أن يحدث عندما يستجيب النظام لحافز من بيئته.

- يمكنك التفكير في هذه المحفزات على أنها من نوعين:

- البيانات تصل بعض البيانات ويجب معالجتها بواسطة النظام.

- الأحداث يحدث بعض الأحداث التي تؤدي إلى تشغيل معالجة النظام .قد تحتوي الأحداث على بيانات مرتبطة ، على الرغم من أن هذا ليس هو الحال دائمًا.

Data-driven modeling

- Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing.
- Data-driven models show the sequence of actions involved in processing input data and generating an associated output.
- They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system.

النمذجة القائمة على البيانات:

- العديد من أنظمة الأعمال عبارة عن أنظمة معالجة بيانات مدفوعة أساسًا بالبيانات .يتم التحكم فيها عن طريق إدخال البيانات إلى النظام ، مع القليل من معالجة الأحداث الخارجية نسبيًا.

- تُظهر النماذج المبنية على البيانات تسلسل الإجراءات المتضمنة في معالجة بيانات الإدخال وإنشاء مخرجات مرتبطة.
- إنها مفيدة بشكل خاص أثناء تحليل المتطلبات حيث يمكن استخدامها لإظهار المعالجة الشاملة في النظام.

Event-driven modeling

- Real-time systems are often event-driven, with minimal data processing.
- Event-driven modeling shows how a system responds to external and internal events.
- It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another.

النمذجة المدفوعة بالحدث:

- غالبًا ما تكون أنظمة الوقت الفعلي مدفوعة بالأحداث ، مع الحد الأدنى من معالجة البيانات.
- النمذجة المدفوعة بالأحداث توضح كيفية استجابة النظام للأحداث الخارجية والداخلية.
- يقوم على افتراض أن النظام لديه عدد محدود من الحالات وأن الأحداث (المنبهات) قد تتسبب في الانتقال من حالة إلى أخرى.

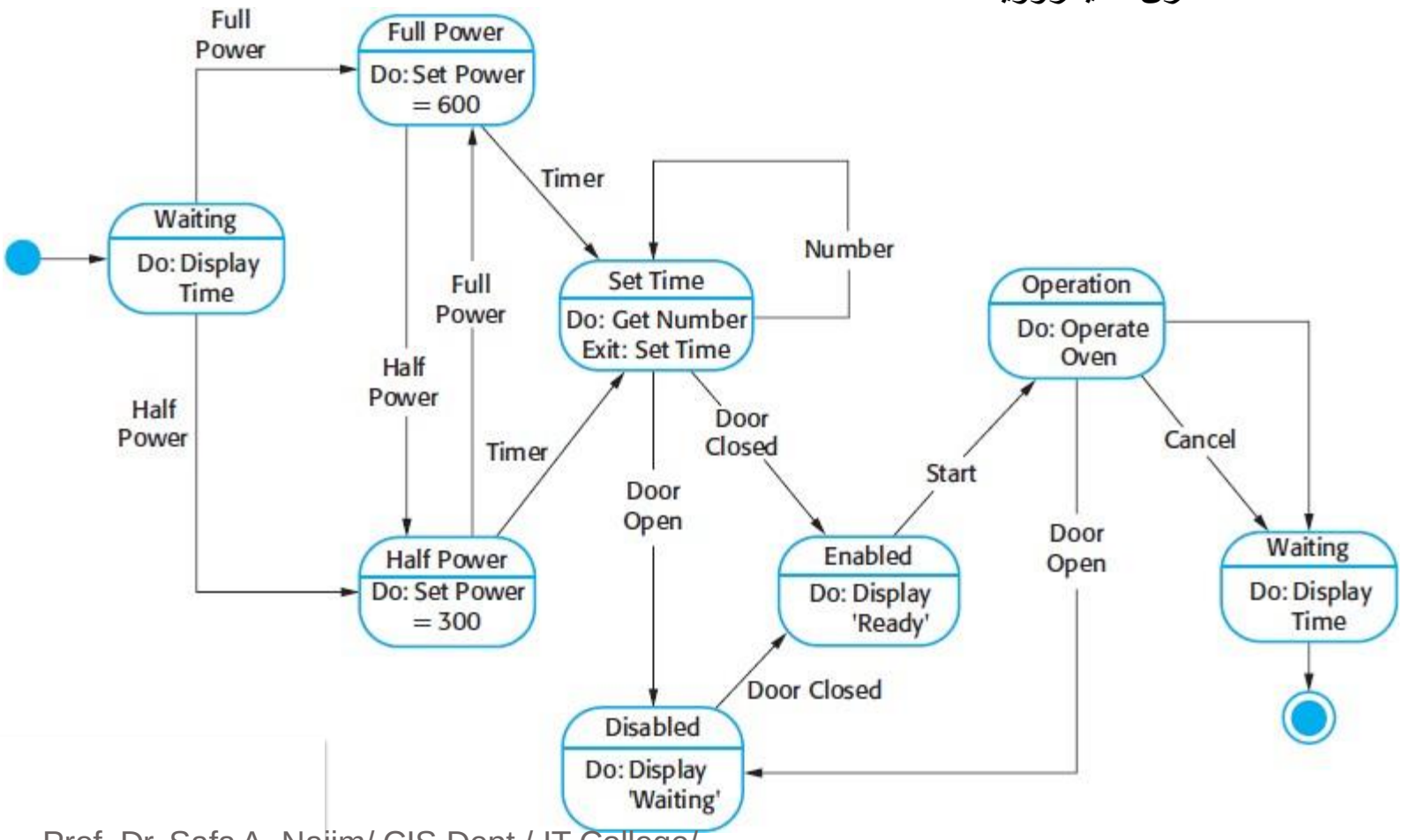
State machine models

- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.

نماذج آلات الحالة:
تُظهر نماذج آلة الحالة حالات النظام كعقد وأحداث كأقواس بين هذه العقد .عند وقوع حدث ما ، ينتقل النظام من حالة إلى أخرى.

State diagram of a microwave oven

مخطط الحالة لفرن الميكروويف:



States and stimuli for the microwave oven (a)

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows ‘Half power’.
Full power	The oven power is set to 600 watts. The display shows ‘Full power’.
Set time	The cooking time is set to the user’s input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows ‘Not ready’.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows ‘Ready to cook’.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows ‘Cooking complete’ while buzzer is sounding.

Prof. Dr. Safa A. Najim/ CIS Dept./ IT College/
Basrah Univ.

حالات ومحفزات فرن الميكروويف (أ):

الحالة	الوصف
انتضار	الفرن ينتظر الإدخال. تظهر الشاشة الوقت الحالي.
نصف القوة	تم ضبط طاقة الفرن على 300 واط. يظهر على الشاشة "نصف القوة".
قوة كاملة	تم ضبط طاقة الفرن على 600 واط. تظهر الشاشة "القوة الكاملة".
ضبط الوقت	يتم ضبط وقت الطهي على قيمة إدخال المستخدم. تُظهر الشاشة وقت الطهي المحدد ويتم تحديثه عند ضبط الوقت.
تعطيل	تم تعطيل تشغيل الفرن من أجل السلامة. ضوء الفرن الداخلي مضاء. يظهر على الشاشة "غير جاهز".
تفعيل	تم تفعيل تشغيل الفرن. ضوء الفرن الداخلي مطفأ. يظهر على الشاشة "جاهز للطهي".
العملية	الفرن قيد التشغيل. ضوء الفرن الداخلي مضاء. يظهر على الشاشة العد التنازلي للمؤقت. عند الانتهاء من الطهي ، يُطلق الجرس لمدة خمس ثوانٍ. ضوء الفرن مضاء. تعرض الشاشة "الطهي مكتمل" أثناء سماع صوت الجرس.

States and stimuli for the microwave oven (b)

Stimulus	Description
Half power	The user has pressed the half-power button.
Full power	The user has pressed the full-power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the Start button.
Cancel	The user has pressed the Cancel button.

حالات ومحفزات فرن الميكروويف (ب):

التحفيز	الوصف
نصف قوة	ضغط المستخدم على زر نصف الطاقة.
قوة كاملة	ضغط المستخدم على زر الطاقة الكاملة.
مؤقت	ضغط المستخدم على أحد أزرار المؤقت.
رقم	ضغط المستخدم على مفتاح رقمي.
الباب مفتوح	مفتاح باب الفرن غير مغلق.
الباب مغلق	مفتاح باب الفرن مغلق.
بدء	ضغط المستخدم على زر البدء.
الغاء	ضغط المستخدم على زر إلغاء.

Microwave oven operation

تشغيل فرن الميكروويف