SUPER SOFTWARE

Hasan Ali Ahmad

Contents

Software engineering

Software engineering means building a new product or systems or modifying and developing old system and improving their mistakes, as well as finding problems that already exist.

هندسة البرمجيات تعني بناء منتج او أنظمة جديدة او تعديل وتطوير النظام القديم وتحسين اخطائه وكذلك إيجاد المشاكل الموجودة بالفعل.

Software

A software is the programs with data and documentation, and we do not mean here to document the comments between the lines of the program.

Documenting all the steps we are talking to develop any software system at all stages of analysis, design, implementation and testing.

هو البرامج التي تحتوي على بيانات وتوثيق، ولا نقصد هنا توثيق التعليقات بين سطور البرنامج.

توثيق جميع الخطوات التي نتحدث عنها لتطوير أي نظام برمجي في جميع مراحل التحليل والتصميم والتنفيذ والاختبار.

Software product:

1- Generic product:
   Stand-alone systems that are marketed and sold to any customer who wishes to buy them.

أنظمة قائمة بذاتها يتم تسويقها وبيعها لأي عميل يرغب في شرائها.

Examples: – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists

2- Customized products:

Software that is commissioned by a specific customer to meet their own needs.

برنامج تم تكليفه من قبل عميل معين لتلبية احتياجاته الخاصة

Examples: – embedded control systems, air traffic control software, traffic monitoring systems.

The good software:

Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

يجب أن توفر البرامج الجيدة الوظائف والأداء المطلوبين للمستخدم ويجب أن تكون قابلة للصيانة والاعتماد عليها وقابلة للاستخدام.

Dependency:

1- Security
2- Availability
3- Reliability
4- Safety

الاعتمادية: وهو مفهوم واسع يشمل مفاهيم أخرى وهي:

1-الأمان: لا يجب للمستخدم او أي شخص أن يصل لمعلومات داخل النظام لا يتوجب عليه الوصول إليه.

2-الاتاحة: يجب ان يكون النظام متاح دائماً.

3-الوثوقية: يجب ان يكون النظام موثوقاً (مثل نظام تحويل الأموال بين حسابين)

4-الحماية: يجب ان تتوفر الحماية في بعض الأنظمة مثل أنظمة الطيران.

The fundamental software engineering activities

1. **1 – Analysis**
2. **2 – Design**
3. **3 – Implementation**
4. **4 – Testing**

الأنشطة الأساسية لهندسة البرمجيات:

1- التحليل

2- التصميم

3- التنجيز

4- الاختبار

The difference between software and hardware

Software	Hardware
logical system element	physical system element
developed/engineered	manufactured
Usually custom-built	assembled from existing component
no spare parts	spare parts

Computer Address Software Engineering

Software systems that are intended to provide automated support for software process activities

أنظمة البرامج التي تهدف إلى توفير الدعم الآلي لأنشطة عمليات البرامج

- **Upper-Case:** Tools to support the early process activities of requirements and design.

وهذه الأدوات تكون قليلة وتقتصر فقط على المراحل الأولى من تحليل وتصميم

- **Lower-Case:** Tools to support later activities such as programming, debugging and testing.

وهي الأدوات التي تساعد في عمليات التصحيح والبرمجة والاختبار

Software development life cycle:

1. requirement analysis
2. Design
3. Implementation
4. Testing
5. Maintenance

The software process: الإجرائية البرمجية

مجموعة الأنشطة والطرق والأدوات والمخططات المترابطة بعضها البعض
التي تؤدي لفهم المشكلة وإيجاد نظام جديد مجدي لحل تلك المشكلة والذي يحقق الكلفة المنخفضة و الجودة العالية ورضا الزبون.

تلك النشاطات نقسمها إلى أربعة أقسام عامة:

1- التوصيف (التحليل)

2- التصميم

3- الاختبار

4- الصيانة

اما النموذج الاجرائي هو طريقة ورود تلك الأنشطة السابقة وفق اجراء معين او طريقة معينة.

1- التحليل: ما هو مطلوب من النظام او ما يقدمه النظام من خدمات وهذا ما نعرفه من

Stake holders:

هم اشخاص او مؤسسات تمثل بأشخاص يؤثرون او يتأثرون بطريقة مباشرة او غير مباشرة في النظام.

بعد عملية تحديد المشكلة نقوم بتحديد أصحاب المصلحة ثم نقوم **بعملية جمع المتطلبات**.

وبعد عملية جمع المتطلبات، نقوم **بتحليل المتطلبات** وتصنيفها وذلك بالاتفاق مع أصحاب المصلحة وهذا ما نسميه بالتوصيف.

2- التصميم: هذه المرحلة ننتقل من مرحلة ما هو مطلوب من النظام إلى مرحلة تحقيق ما هو مطلوب من هذا النظام

- وضع معمارية النظام.
- وضع الخوارزميات اللازمة لكل جزء من الأجزاء حيث انه يعتمد على عمل المحلل ويقوم بتوزيع الخدمات التي يقدمها المحلل الى تلك الأجزاء المتكاملة التي تكون النظام.
- تصميم قاعدة المعطيات والمخططات اللازمة للنظام.
- تصميم الواجهات الخاصة بالنظام البرمجي.

3- التنجيز: يقوم بهذه المرحلة المبرمج بكتابة الخوارزميات التي صممها المصمم وتحويلها الى كود.

4- الاختبار: مرحلة تجريب النظام واكتشاف الاخطاء

Software process model:

1. Plan-driven

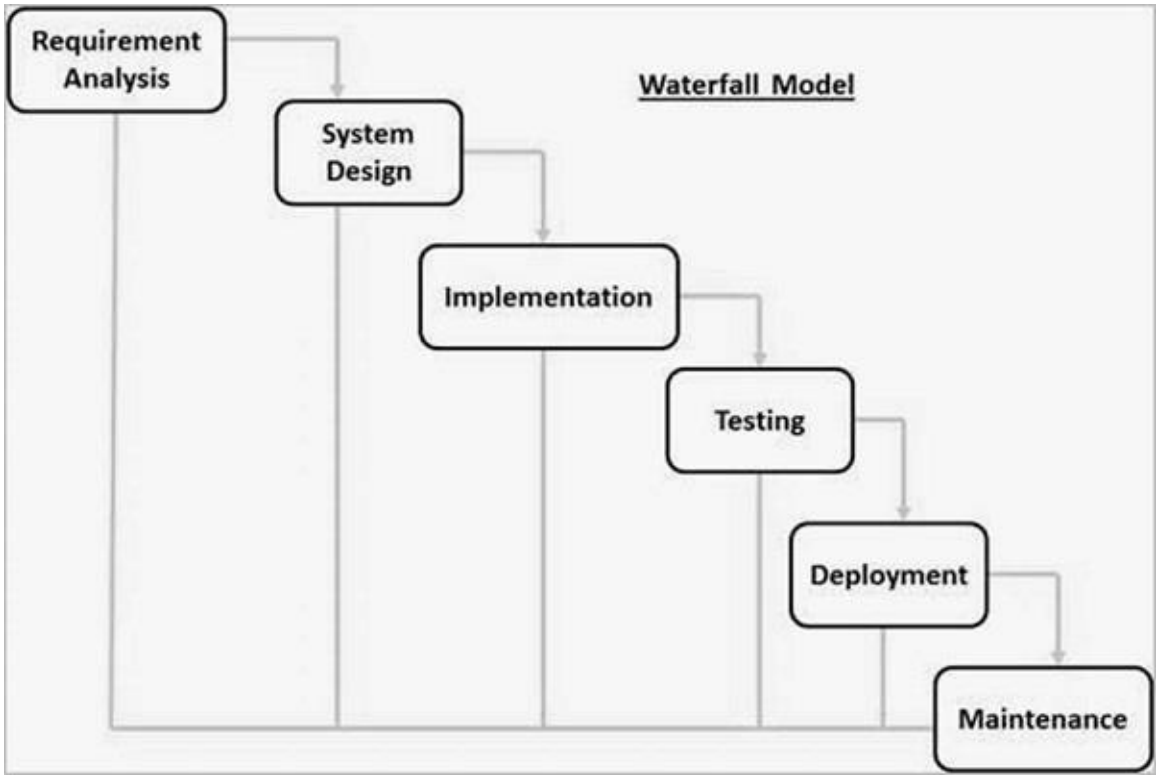
كل الأنشطة السابقة يتم التخطيط لها بشكل كامل وتام ونضع المخاطر بعين الاعتبار ونضع خطط بديلة في الحالات الطارئة.

2. Agile

الهدف من هذه النماذج هي الانتهاء من عملية تطوير النظام بأسرع وقت ممكن وذلك على حساب الجودة وهو لا يحتاج لتخطيط عالي.

Plan- driven process:

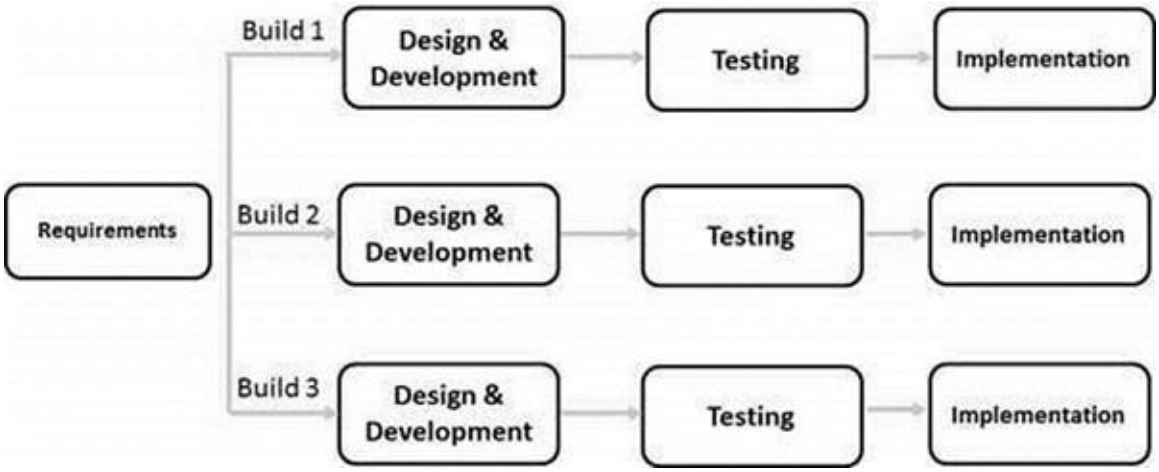
1- The waterfall model النموذج الشلالي



مساوئ: يستهلك وقت كبير – لا يقبل أي تغيير في المتطلبات

محاسن: سهل – لا يحتاج خبرات في الفريق

2- Incremental process model النموذج التزايدي



يتم التطوير على أكثر من نسخة

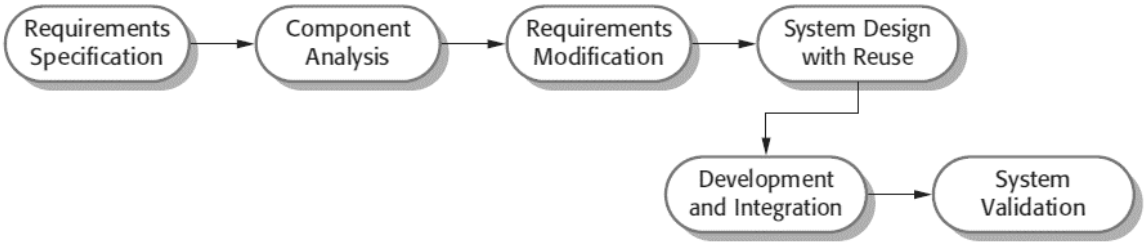
محاسن: الوقت القصير الذي يمكننا من الحصول على منتج ملموس

يمكن التعامل مع متطلبات متغيرة وتعديل المتطلبات واضافة خدمات جديدة

مساوئ: يمكن ان تتعارض الخدمات القديمة مع الخدمات الجديدة

يستخدم غالباً في بناء تطبيقات الويب.

3- Reuse-Oriented software engineering النموذج المقام بإعادة الاستخدام



البحث عنا هي مرحلة نقوم بها بالبحث عن مكونات جاهزة نضمنها في مشروعنا، حيث بعد القيام بعملية التحليل وتحديد المتطلبات والخدمات التي يقوم بها النظام نجري عملية بحث في السوق عن أنظمة قديمة تخدم الخدمات التي يقدمها نظامي فنقوم بتجميع هذه المكونات الجاهزة وربطها سوياً ليتشكل نظامنا، لكن أحياناً يكون لدينا خدمة موجودة في نظامنا لكنها خدمة غير موجودة بالسوق ولا توجد مكونات جاهزة تقدمها لي عندها نضطر لإجراء عملية تنجيز بأنفسنا لهذه الخدمة.

محاسن: السرعة في التنجيز – الوثوقية بسبب عمليات الاختبار.

مساوئ: احياناً لا تتوافر المكونات المناسبة – ربط المكونات مع بعضها قد يكون معقد.

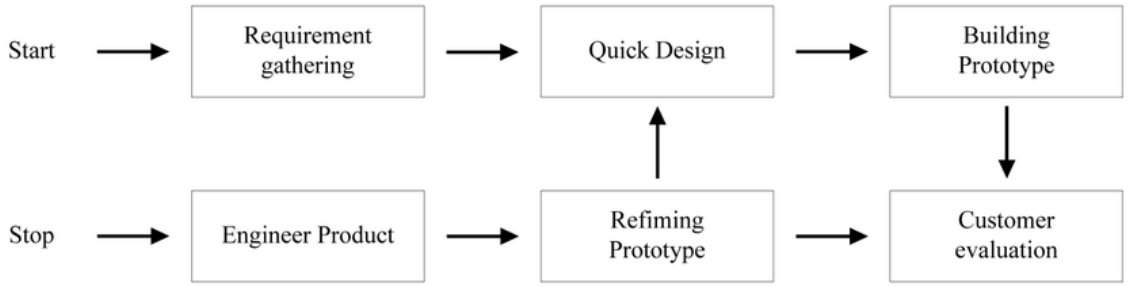
إعادة الاستخدام باستخدام مكونات جاهزة:

مثال: في تطبيقات الويب قد نستخدم قوالب جاهزة في تصميم الواجهات ولذلك تطبيقات الويب تطور باستخدام هذا النموذج او النموذج التزايدي.

إعادة الاستخدام باستخدام المعرفة:

أي القيام بإعادة استخدام الفكرة البرمجية وليس الكود.

4- Proto-type model النموذج الاولي



يستخدم هذا النموذج في حالتين:

1- تطوير نظام جديد ذو فكرة جديدة غير مطبق من قبل (غامض).

2- عندما يكون الزبون غير قادر عن التعبير عن متطلباته.

نقوم بتصميم رسوم أو واجهات للعرض والخرج ولكن هذه الواجهات غير فعالة، يؤدي ذلك إلى تحفيز الزبون وذلك لجمع المتطلبات منه، لأن رؤية أشياء ملموسة يختلف عن التحدث عنها بشكل مجهول.

يوجد نوعين لهذا النموذج:

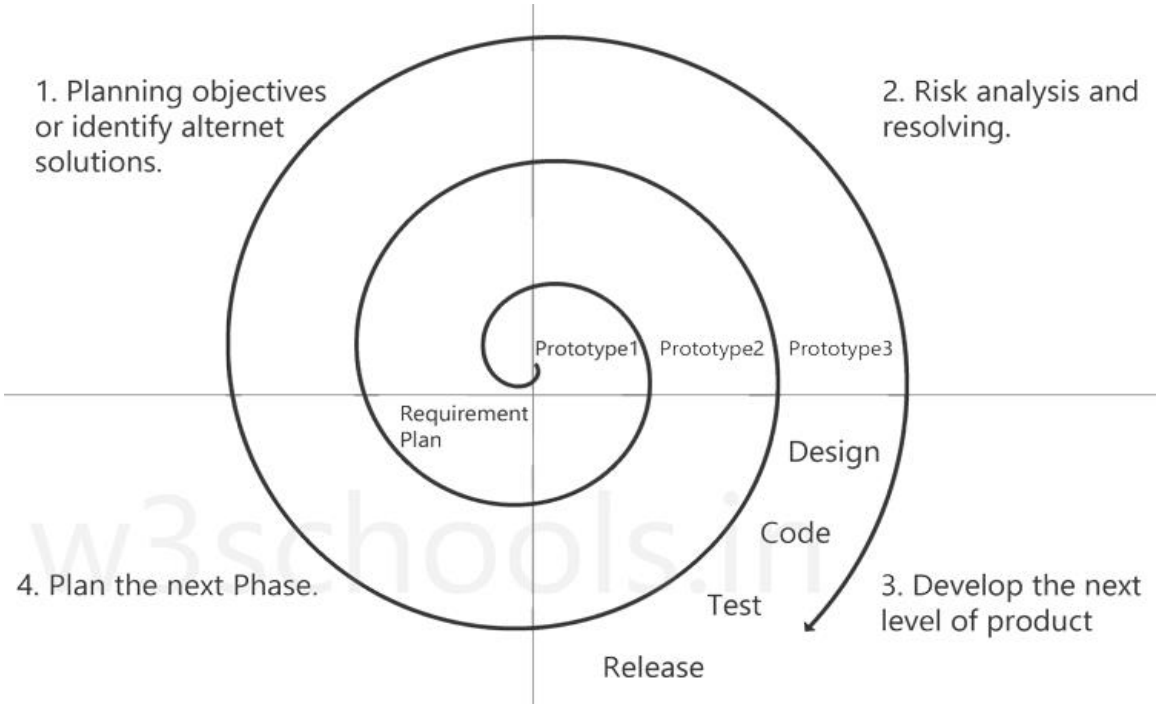
1- مهمل: الهدف منه فقط جمع المتطلبات من الزبون ثم رمي الواجهات والرسومات التي قمنا بتصميمها ومتابعة الأنشطة من تحليل وتصميم وتنجيز واختبار.

2- تطويري: الهدف منه تطوير الواجهات التي صممناها ونستخدمها في النظام الذي نطوره.

محاسن: انه الحل الوحيد إذا كان: نظام جديد تماماً – زبون غير قادر عن التعبير عن متطلباته.

مساوئ: مستهلك للوقت في عملية جمع المتطلبات – مستهلك للجهد – غير مضمون.

5- Spiral model النموذج الحلزوني



هذا النموذج مكون من أربع قطاعات أساسية وهي:

1- التخطيط 2- دراسة المخاطر 3- التطوير 4- التقييم

محاسن: يأخذ بعين الاعتبار التغييرات ودراسة المخاطر ويكون ذو جودة عالية بسبب عدد التكرارات التي يتم تنفيذها للحصول على هذا النظام ويكون قابل للصيانة.

مساوئ: صعب التطبيق بشكل كبير – يحتاج الى فريق ذو خبرات ومؤهلات عالية وهو يطبق في المشاريع الكبيرة مثل (نظام التحكم بالطائرة نظام ضخم جداً ويحوي على الكثير من المخاطر).

الإجرائية الموحدة The Unified process (RUP)

هذه الإجرائية تقوم بعمليات التحليل والتصميم غرضي التوجه بمساعدة لغة النمذجة الموحدة

Unified model language (UML).

الهدف من عملية النمذجة:

1- التوثيق.

2- التعامل مع المخططات يكون أسهل من الكتابة.

3- التوصيف: نوصف النظام على شكل مخططات وتوضع المخططات مع وثيقة المتطلبات.

4- فهم النظام.

ما هي الأمور التي تجعلنا نختار النموذج الإجرائي المناسب للنظام؟

1- المتطلبات:

إذا كانت واضحة والنظام واضح نختار الشلالي.

إذا كانت جديدة وغير معروفة نختار البدائي.

إذا أردنا السرعة نختار التزايدي.

إذا أردنا الجودة العالية نختار الإجرائية الموحدة.

2- فريق العمل:

إذا كان الفريق كبير نختار الإجرائية الموحدة.

إذا كان ذو خبرات عالية نختار الحلزوني.

إذا كان صغير نختار التزايدي.

إذا كان لا يملك الخبرة نختار الشلالي.

3- الزبون:

إذا كان لا يعرف التعبير عن متطلباته نختار البدائي.

إذا كان متقلباً ويغير متطلباته باستمرار نختار التزايدي.

Agile Software Development

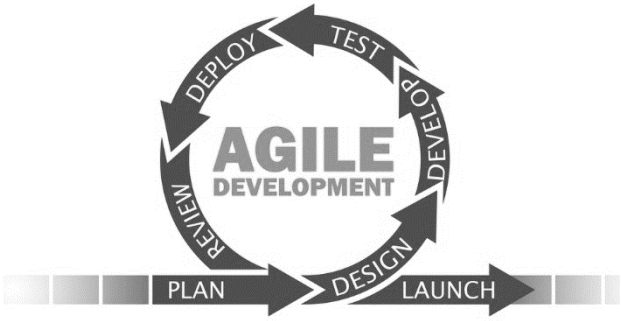
بهذا النوع من تطوير النظم لا يوجد لا يوجد لدينا تخطيط كبير وكذلك لا يوجد توثيق وبالتالي عند ظهور مشاكل معينة في النظام لا نستطيع حلها وذلك لان الأنشطة من (تحليل – تصميم – تنجيز – اختبار) متداخلة مع بعضها البعض ويتم تطوير النظام بشكل تزايدي وذلك لجعله يقدم الخدمات الأساسية وبعد ذلك تبدأ مرحلة التطوير وغالباً ما نقوم باستخدام مكونات جاهزة او أدوات تساعدنا في عملية التطوير.

متى يكون استخدام طريقة اجايل مناسباً للتطوير؟

- يجب ان يكون النظام صغير او متوسط حصراً أي ان النظم الكبيرة لا تطور ب اجايل.
- ان يكون فريق التطوير صغيراً ومتكاملاً بالخبرات ومتفاهماً ولا يتجاوز 7 اشخاص.

الهدف الأساسي من اجايل:

تخفيف التوثيق قدر الإمكان والاستجابة مع التغيرات بسرعة كبيرة عن طريق تقديم النظام على شكل نسخ حيث نجعله اولاً يقدم الخدمات الأساسية ثم تبدأ عليه مرحلة التطوير أي يتم تطويره بشكل تزايدي.



The principles of agile methods

1- Customer involvement

يجب ان يكون الزبون مع فريق العمل دائماً ويقوم بتقييم العمل اثناء عملية التطوير.

2- Incremental delivery

يجب ان يقدم النظام المطور على شكل نسخ.

3- people not process

الاجايل لا تؤكد على الأنشطة بل تؤكد على الناس ويجب على فريق التطوير ان يكون متكاملاً.

4- Embrace change

يجب ان تكون قادرة على التعامل مع تغييرات النظام.

5- Maintain simplicity

يجب ان تكون بسيطة.

إذا تحققت جميع الشروط السابقة نسمي الطريقة (اجايل).

Agile method == Agile team.

Agile team → small size and skilled people.

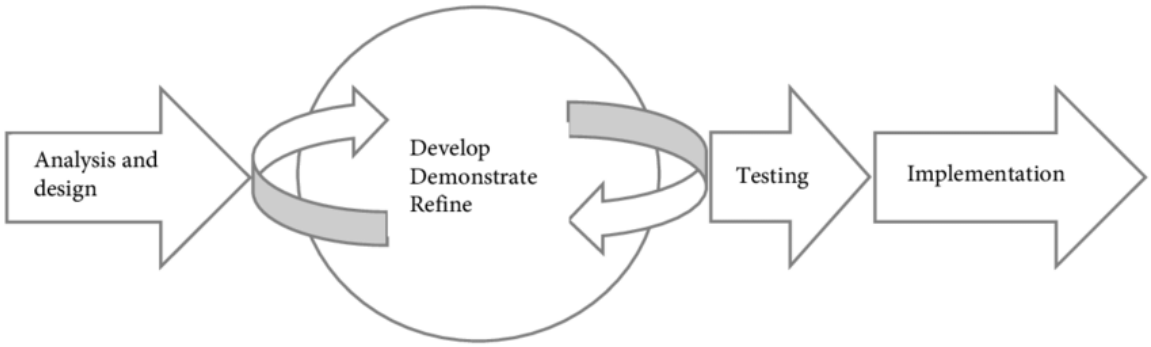
Agile Models

1- Rapid Application Development (RAD)

فكرة هذا النموذج تقوم على أساس القيام بمرحلة تحليل النظام بشكل كامل وبعد القيام بعملية التحليل يتم تقسيم النظام إلى مجموعات وأجزاء مثلاً:

- قسم للواجهات.
- قسم للخوارزميات اللازمة للنظام.
- قسم لمعمارية النظام.
- قسم لقاعدة البيانات الخاصة بالنظام.

يتم تطوير المهام بشكل متوازي مع بعضها البعض بعد ذلك يقوم الفريق بعملية تكامل لهذه المهام وعندها النظام أصبح جاهزاً.



2- Extreme programming (xp)

هي الطريقة الأساسية ب اجايل وسميت بهذا الاسم لأنها تركز على عمليات البرمجة وعملياً هي تهتم بتحويل القصص التي يقصها الزبون الى كود.

مرحلة تحويل المهام الى كود يعتمد على الطرق التالية معاً:

1- Pair programming:

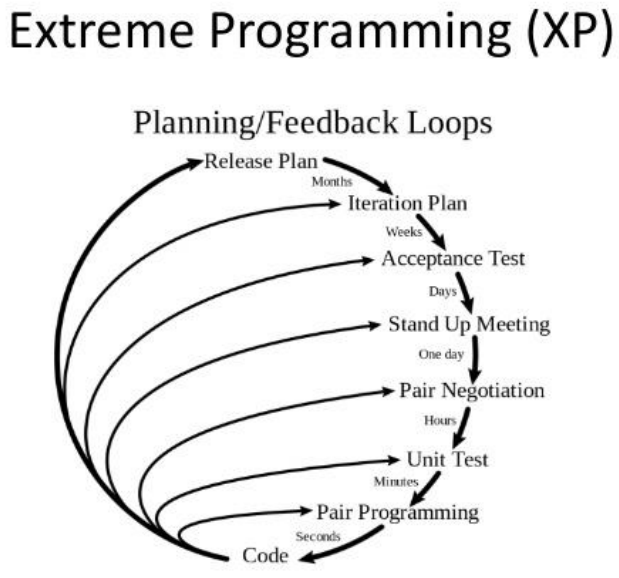
هذه الطريقة يقوم بها مبرمجين اثنين حيث يستلم أحدهم عملية كتابة الكود والأخر يقوم بعملية مرجعة للكود وبعد ذلك يتبادلان الأدوار.

2- Test-first development:

هذه الطريقة تقوم بإدخال بيانات على الكود واختباره إذا اعطى النتائج بشكل صحيح نقوم بتحويلها الى كود للنظام.

3- Refactoring:

بعد انهاء كل قصة نقوم بتحسين الكود وتنظيفه.



3- Scrum model

هي إطار اداري لتطوير النظم بشكل سريع.

يوجد بهذه الطريقة 3 مراحل أساسية:

1- Planning and architectural design:

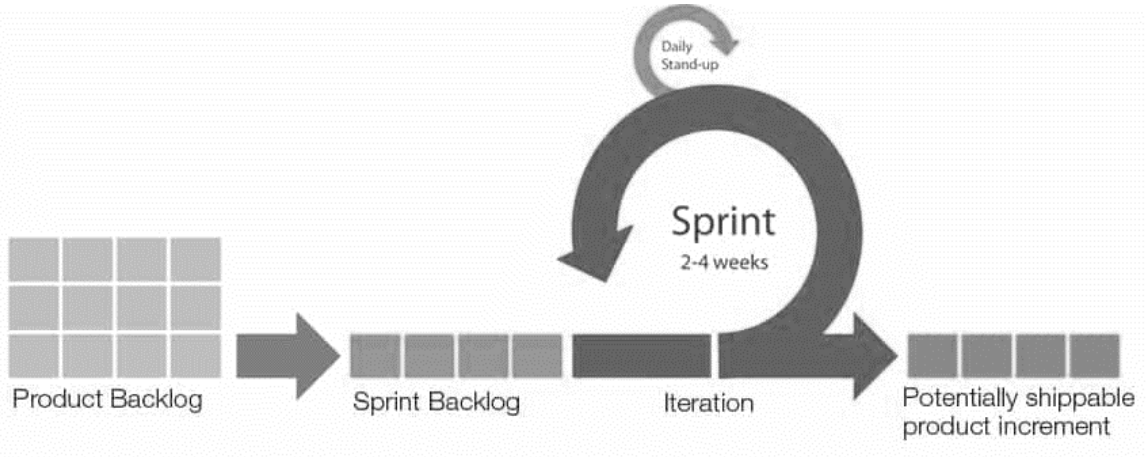
في هذه المرحلة يقوم الفريق بمقابلة الزبون وتجميع المتطلبات ووضع مخطط سريع للعمل ومعرفة ما هو مطلوب من النظام.

2- Sprint cycle:

يتم تطوير النظام بشكل تزايدي ويقود هذه المرحلة رئيس مهمته الأساسية هي عزل فريق العمل عن العالم الخارجي ويكون هو صلة الوصل بين الزبون وفريق التطوير.

3- Project closure:

sوهي المرحلة النهائية ويكون المشروع جاهزاً للتسليم.



Requirements engineering

هندسة المتطلبات تعتبر المرحلة الأساسية في مرحلة تحليل النظام

يكون دخل هذه المرحلة دراسة الجدوى من النظام أي إذا كان غير مجدي فلا نطوره.

يكون خرج هذه المرحلة وثيقة المتطلبات وهي الوثيقة التي تضم المتطلبات بشكلين:

- شكل مبسط للمستخدم – شكل مفصل للمطور.

تسمى هذه الوثيقة

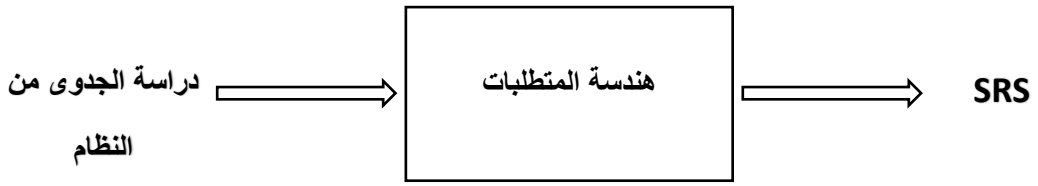
Software Requirement Specification (**SRS**).

وهي عبارة عن مستند يوصف النظام ويضم عدد من الوثائق والمخططات المطلوبة وهذه الوثيقة تلعب ثلاثة أدوار وهي:

1- تكون بمثابة وثيقة لتوقيع عقد مع الزبون.

2- تكون دخل لمرحلة التصميم وهي نقطة البداية التي يبدأ بها المصممون لتصميم النظام اعتماداً عليها.

3- تكون مرجع أساسي في عمليات الاختبار.



Requirement

المتطلب هو الشيء المطلوب من النظام أي مجموعة الوظائف المطلوب تقديمها من النظام بعد تطويره بالإضافة الى القيود على هذه المتطلبات.

أنواع المتطلبات:

متطلبات وظيفية Functional requirement

هي الوظائف الأساسية التي يقدمها النظام مثل (خدمة الحجز – العرض – التسجيل – وغيرها..).

متطلبات غير وظيفية Nonfunctional requirement

هي عبارة عن قيود على المتطلبات وهي تعد بمثابة متطلبات مثل:

1- متطلبات تتعلق بالجودة:

Security - reliability – performance – availability.

2- متطلبات متعلقة بالوقت أي إعطاء فترة محددة من الزمن لتطوير النظام.

كيف نقوم بجمع المتطلبات؟

نقوم بإحضار المتطلبات من أصحاب المصلحة وهم اشخاص او مؤسسات تمثل بأشخاص يتأثرون بشكل مباشر او غير مباشر بالنظام ويتم تقسيمهم الى:

Stack holders:

1- Developers

2- Users

3- Sponsors

المسؤولون الذين يمولون النظام.

Stack holders VS Actors:

Stack holders

هم المهتمون وهم من يستخدمون النظام وهم دائماً اشخاص.

Actors

هو من يفعل النظام او يفعل خدمة من النظام وهو ليس بالضرورة ان يكون شخص يمكن ان يكون:

Hardware – Software – Time...

ترتيب العمل في هندسة المتطلبات يكون كالتالي:

1- نحدد المهتمون (أصحاب المصلحة)

2- نقوم بعملية جمع المتطلبات.

3- نقوم بعملية تحليل لهذه المتطلبات.

4- نقوم بعملية التحقق او إدارة المتطلبات.

سنشرح كل عملية من العمليات السابقة.

1- تحديد المهتمون: نحدد المتأثرون بالنظام.

2- جمع المتطلبات: يتم جمع المتطلبات عبر تقنيات وهي:

- المقابلة.

- الاستبيان.

- الملاحظة.

- دراسة الوثائق: عندما يكون هناك نظام يدوي موجود او نظام معلوماتي لكنه قديم ويرغب بتطويره عندها يقوم محلل النظام بجمع الوثائق والقوائم الخاصة بهذا النظام ودراستها.

مثال جمع متطلبات تطبيق بي اوردر.

Exercise Bee Order:

- Actor Identification.

- Requirement Gathering.

Actors:

1. Primary Actors:

1. **Customers.**
2. **Admin.**
3. **Restaurant manager.**
4. **Restaurant employee.**
5. **Driver.**

2. Secondary Actors:

1. **Call Center.**
2. **Billing System.**
3. **Notification System.**

Requirements Gathering:

- Update menu(add/remove/edit).

- Approve order.

- Add location.

- Payment.

- Login/Logout.

- Register.

- Offers.

- Make Order.

- View restaurant list.

- Order delivery.

- Track order.

- Search.

- Filter.

- Ban customer.

- Calculate cost.

- Assign best restaurant.

- Recommender cart.

- View order history.

- Rate restaurant.

- Rate app.

- Feedback.

- Dashboard.

- Current order.

- Ticket.

- View menu.

- View restaurant.

- View food specification.

- Profile/edit.

- Select service.

- Add/remove/edit restaurant.

- Add/remove/edit customer.

- Add/remove/edit order.

3- تحليل المتطلبات:

Requirement analysis

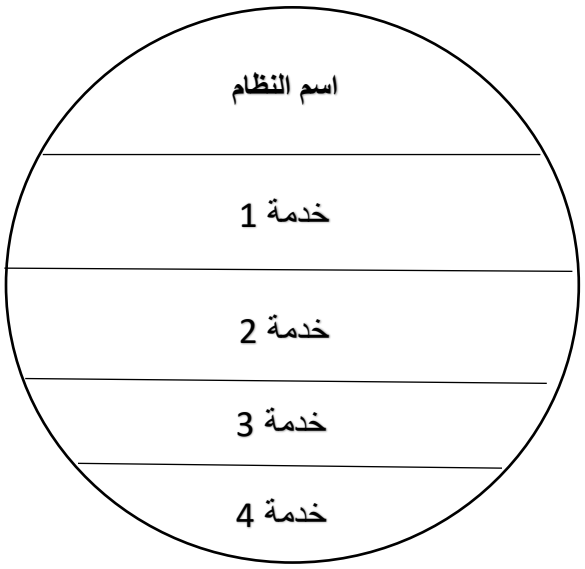
في هذه المرحلة نستخدم أدوات تساعدنا في عمليات تحليل المعطيات وفي عمليات تطوير البرمجيات.

في هذه المرحلة يقوم المحلل بوضع مخططات كثيرة ومنها:

Activity diagram – Use case diagram – Class diagram – Context diagram…

1- Context diagram مخطط السياق

هو المخطط الأول الذي يتم بناءه عند مرحلة تحليل المتطلبات حيث يعرفنا هذا المخطط على سياق النظام ويعطينا الخدمات الأساسية التي يقدمها النظام بشكل مجرد.



Unified Modeling Language (UML)

- A standard language for specifying, visualizing, constructing, and documenting the artifacts software system.

لغة قياسية لتحديد وتصور وبناء وتوثيق عناصر أنظمة البرمجيات.

- Pictorial language used to make software blueprints.

اللغة التصويرية المستخدمة لعمل مخططات البرامج.

- It is not programming language but tools can be used to generate code in various language using UML diagram.

ليست لغة برمجة ولكن يمكن استخدام الأدوات لإنشاء كود بلغات مختلفة باستخدام مخططاتها.

- it has direct relation with object-oriented analysis and design.

لها علاقة مباشرة بالتحليل والتصميم غرضي التوجه.

UML Kinds:

1. Structure diagram:

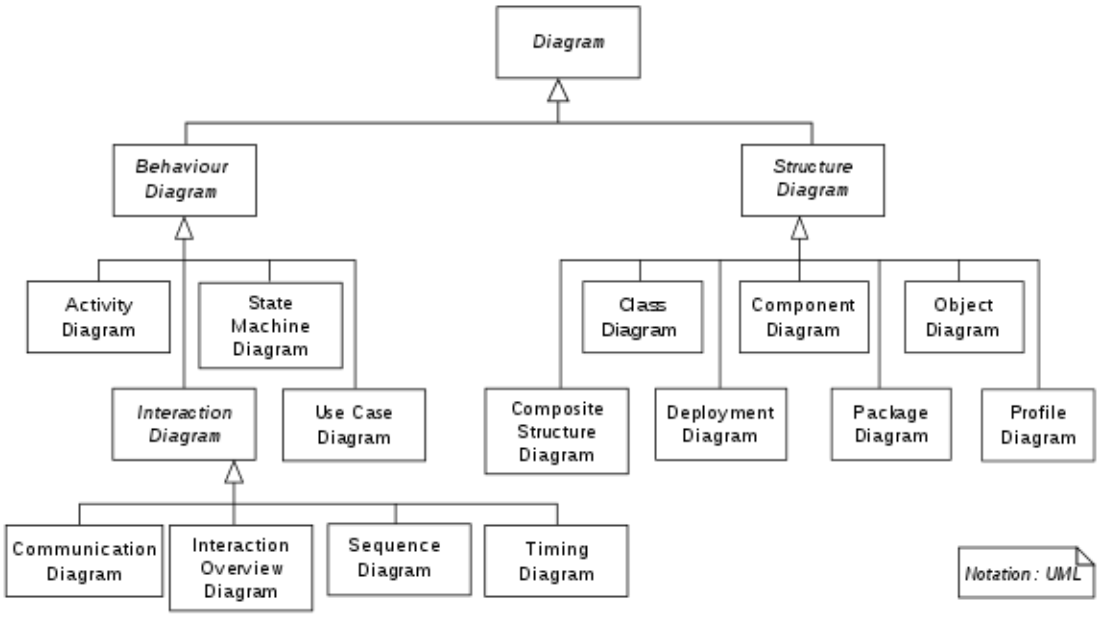
- Show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other.

إظهار البنية الثابتة للنظام وأجزائه على مستويات التجريد والتنفيذ المختلفة وكيفية ارتباطها ببعضها البعض.

2. Behavior diagram:

- show the dynamic behavior of the objects in a system, which can describe as a series of changes to the system over time.

إظهار السلوك الديناميكي للكائنات في النظام، والذي يمكن وصفه بأنه سلسلة من التغييرات على النظام بمرور الوقت.

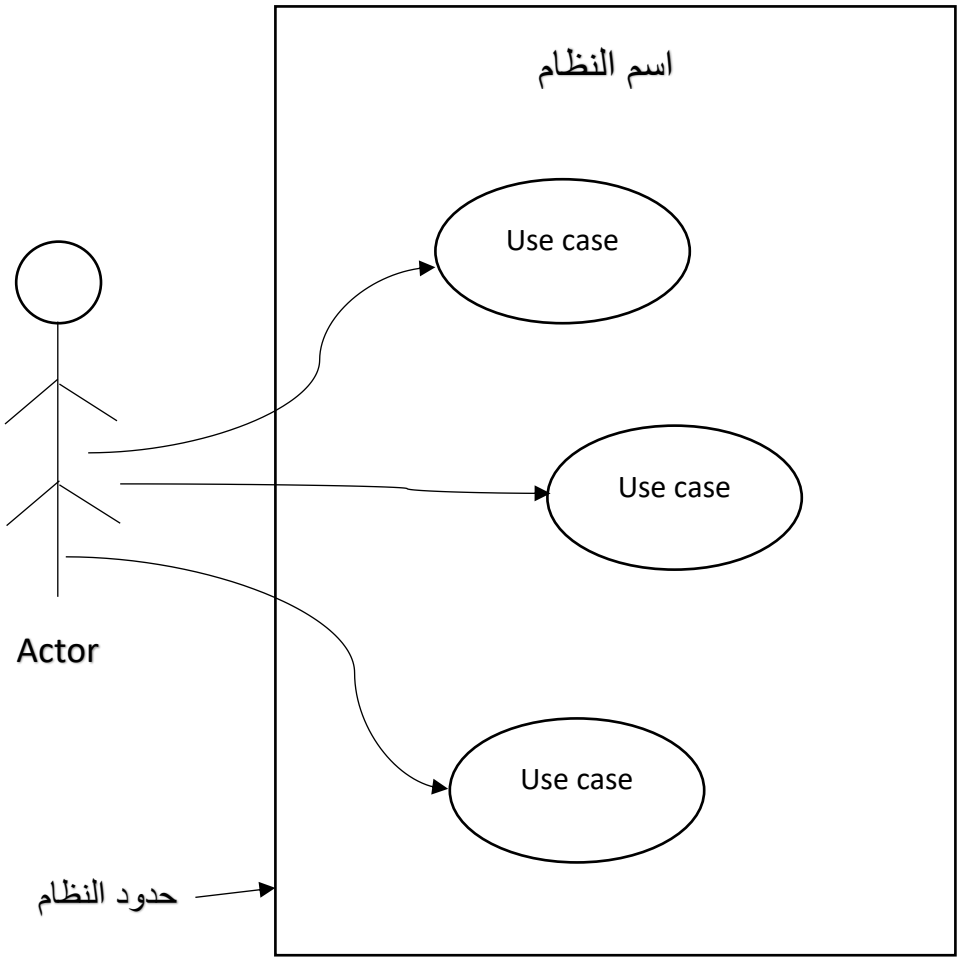


مخطط الحالات Use case diagram

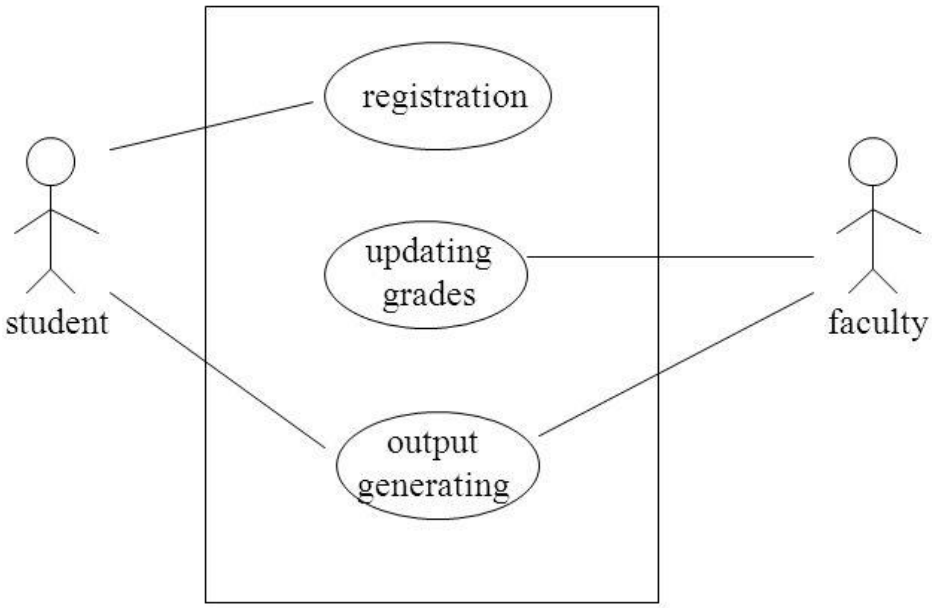
وهو المخطط الثاني الذي يقوم محلل النظام ببنائه وهو سهل الفهم لأنه ينظر للنظام بشكل مجرد.

Use case:

هو اسم الخدمة الذي الوظيفية التي يقدمها النظام وكل واحدة تعبر عن خدمة وظيفية واحدة وفقط وظيفية (الخدمات الغير وظيفية لا نعبر عنها في هذا المخطط بل نكتفي بكتابتها على جنب)



Example of Use Case Diagram

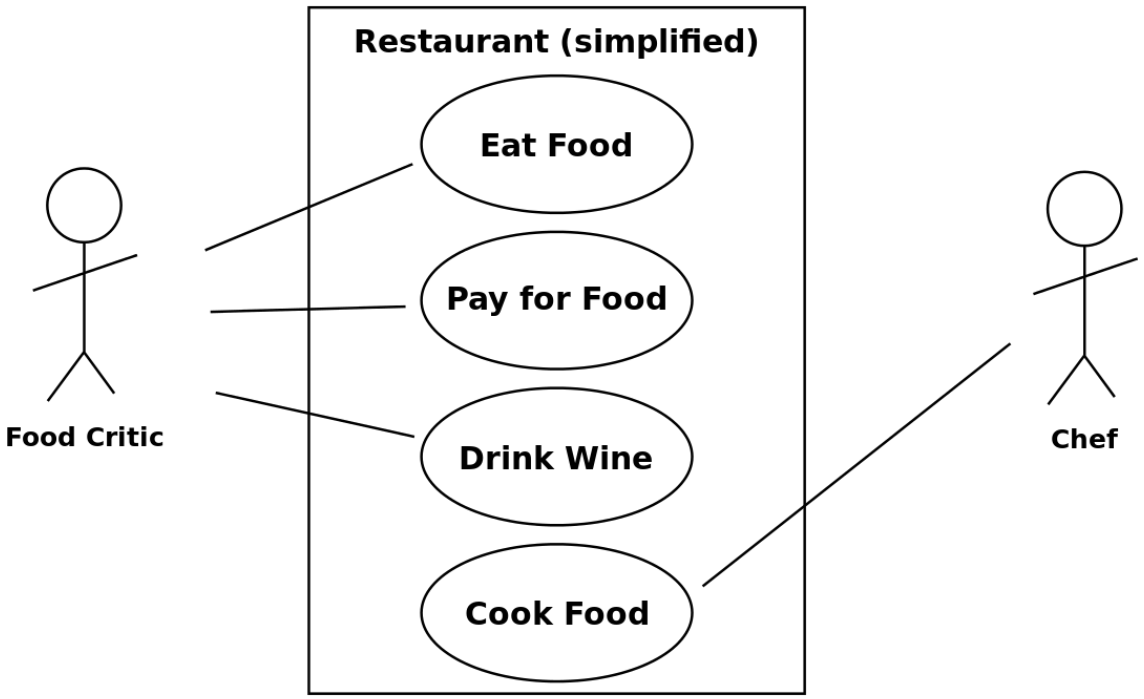


في هذا النظام يوجد فاعلين هما الطالب والكلية

ويوجد ثلاث خدمات:

- **خدمة التسجيل.**
- **خدمة تحديث العلامات.**
- **وإخراج العلامات.**

Example of Use Case Diagram



في هذا النظام يوجد فاعلين هما الزبون والشيف

ويوجد أربع خدمات او وظائف:

- اكل الطعام.

- دفع ثمن الطعام.

- شرب النبيذ.

- طبخ الطعام.

Use Case Diagram Guidelines:

- **Include:** relationship between Use Case

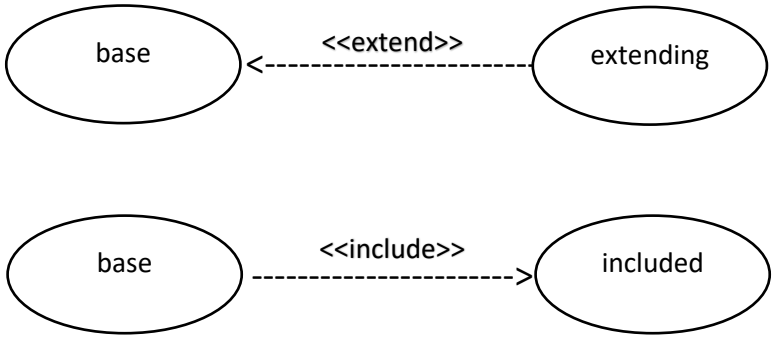
- (one Use Case must call another; e.g., Logout Use Case includes Login Use Case).

علاقة بين خدمتين تعني لا يمكن تنفيذ خدمة دون استدعاء خدمة.

- **Extend:** relationship between Use Case

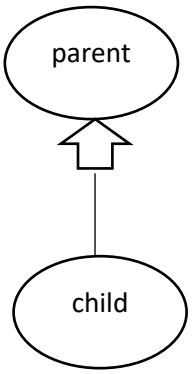
- (one UC call another under certain condition).

علاقة بين خدمتين تعني توسعة لخدمة في خدمة أخرى.



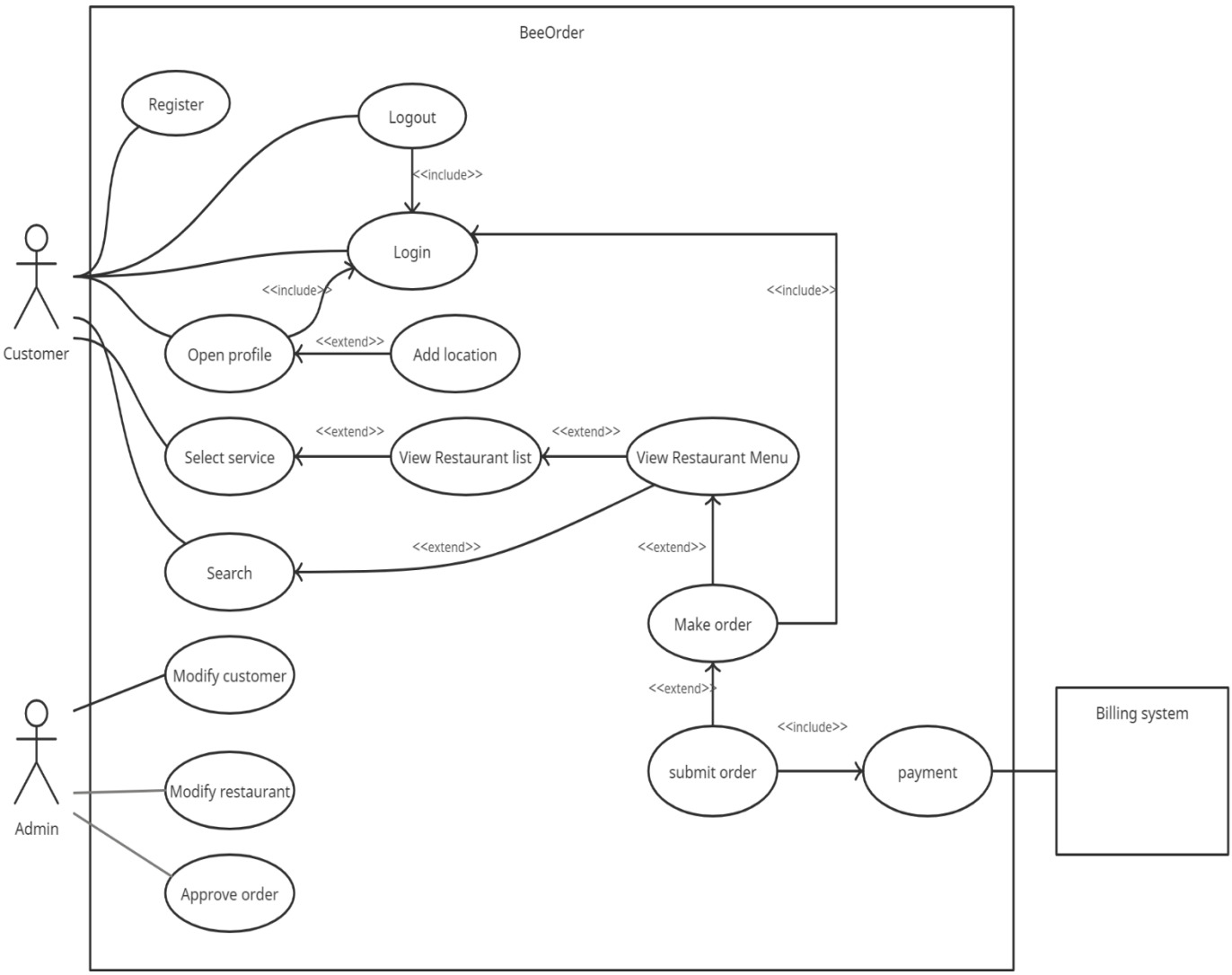
Generalization relationship:

- **Use Case Generalization:** shows that one use case provides all the functionality of the more general use case and some additional function.

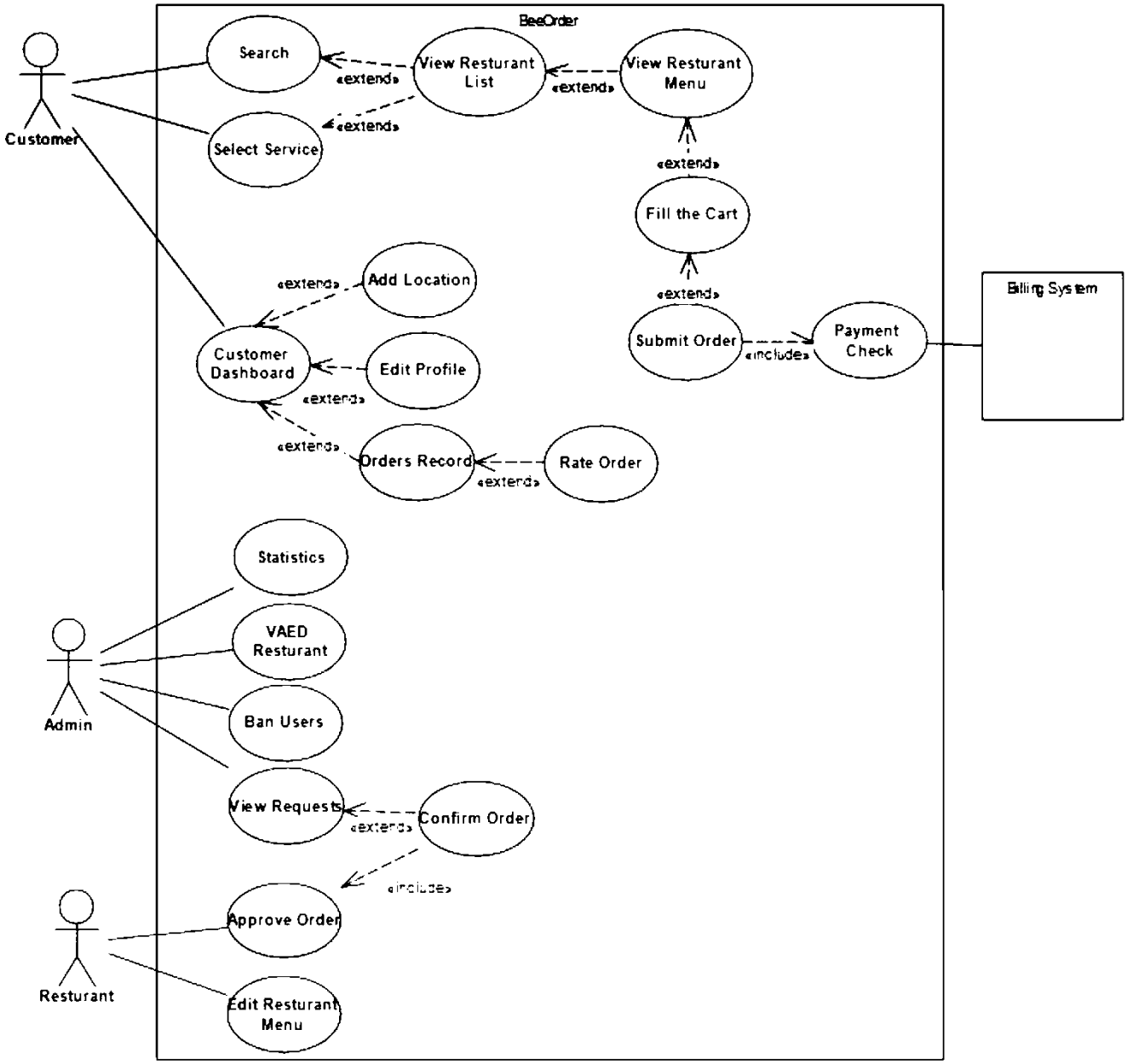


- **Actor Generalization:** shows that one actor can participate in all the associations with use cases that the more general actor can plus some additional use cases.

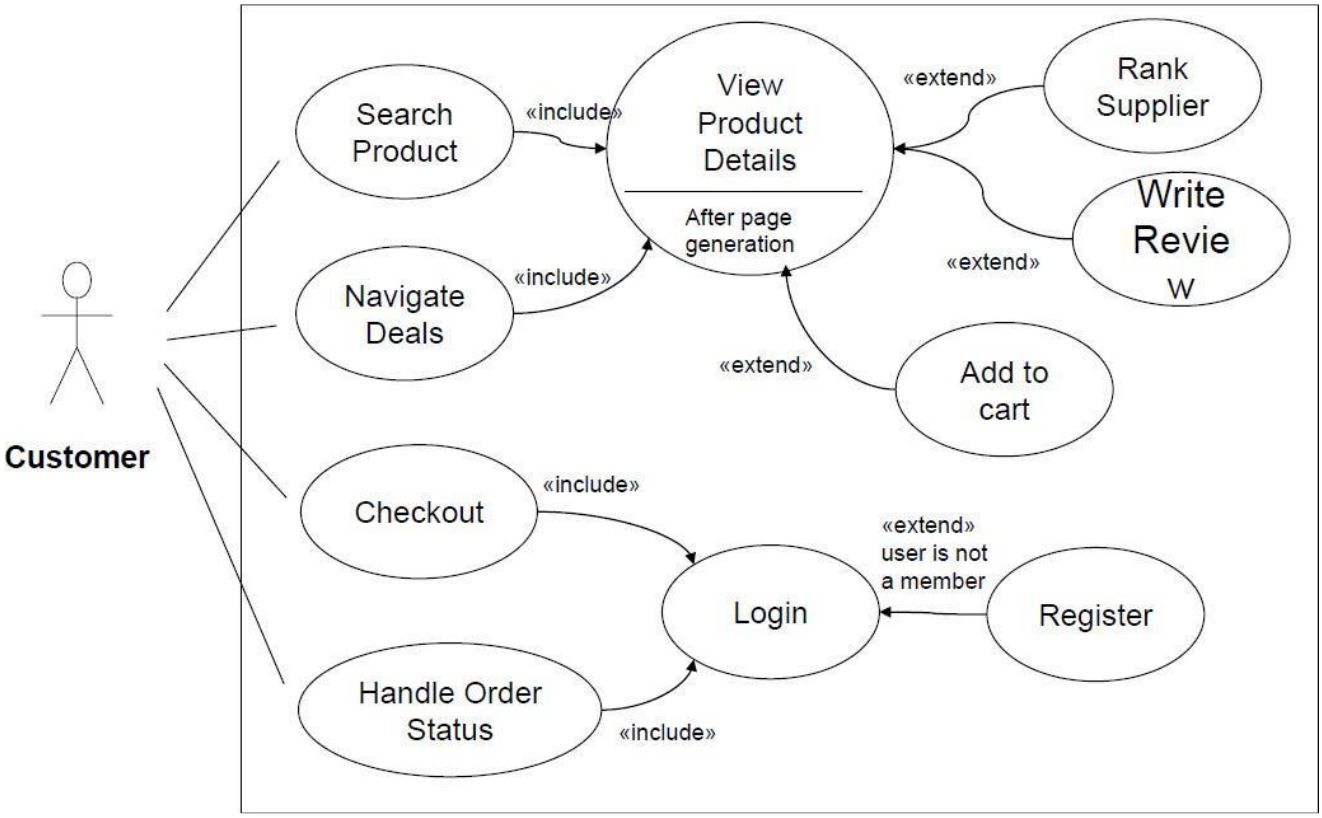
Bee Order Use Case Diagram:



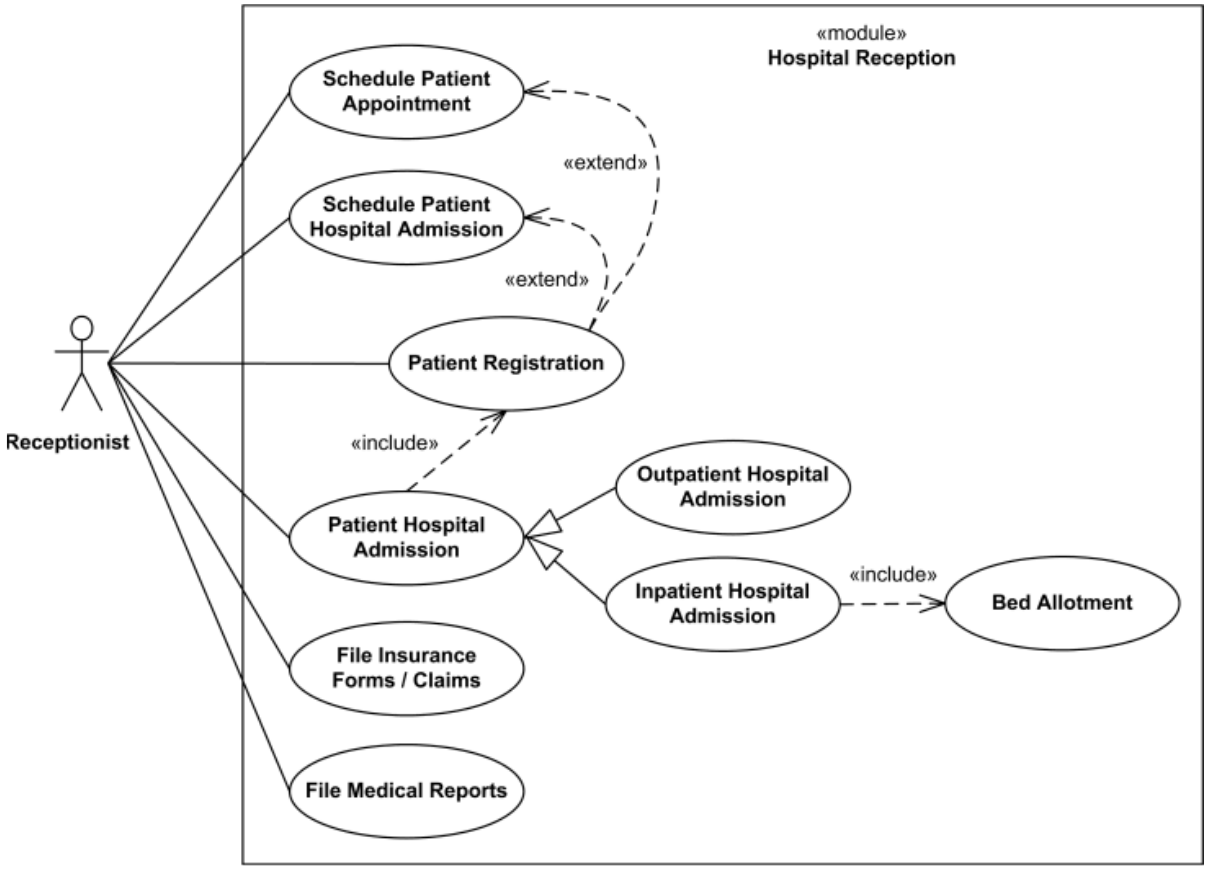
Bee Order Use Case Diagram 2:



Use Case Diagram Example:



Use Case Diagram Example 2:



Use Case Specification مخطط التوصيف

في هذا المخطط نمثل كل خدمة في جدول كالتالي:

ID / Title	اسم الخدمة او رقمها

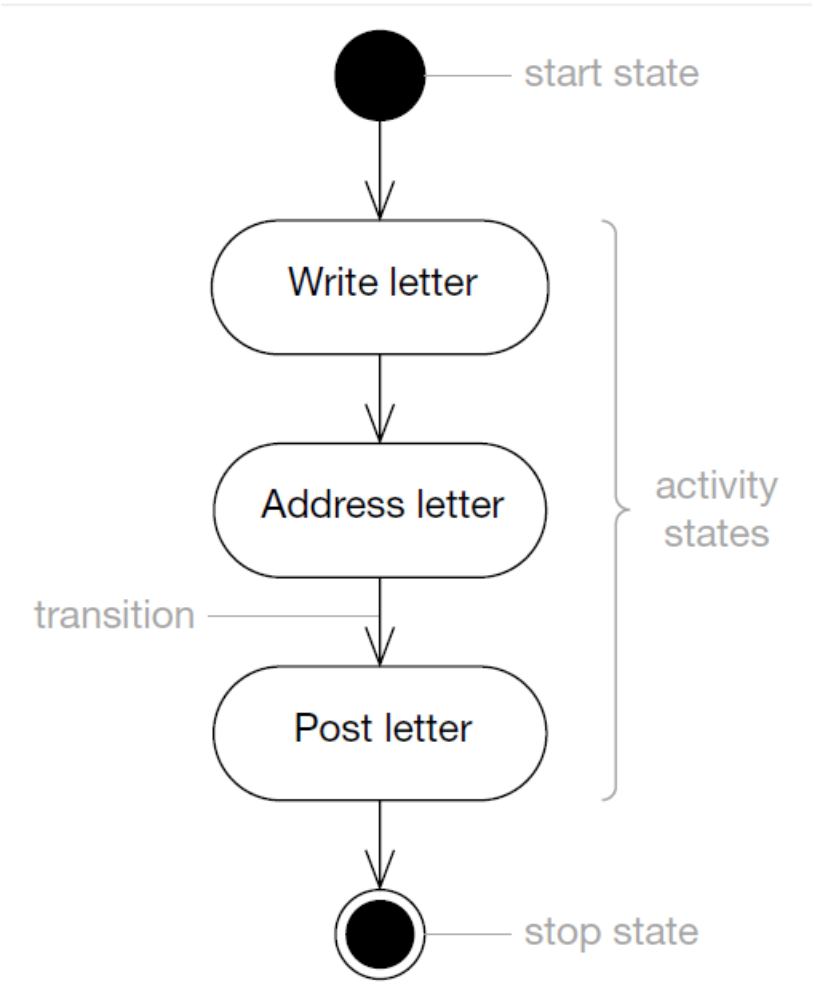
Actor	الممثل	
Description	وصف الخدمة	
Pre-condition	الشرط المسبق للخدمة	
Post-condition	ماذا يحصل في النظام بعد تنفيذ الخدمة	
Flow of event	**Actor**	**System**
Critical Scenario	**Scenario**	**System response**

Bee Order Use Case Specification make order:

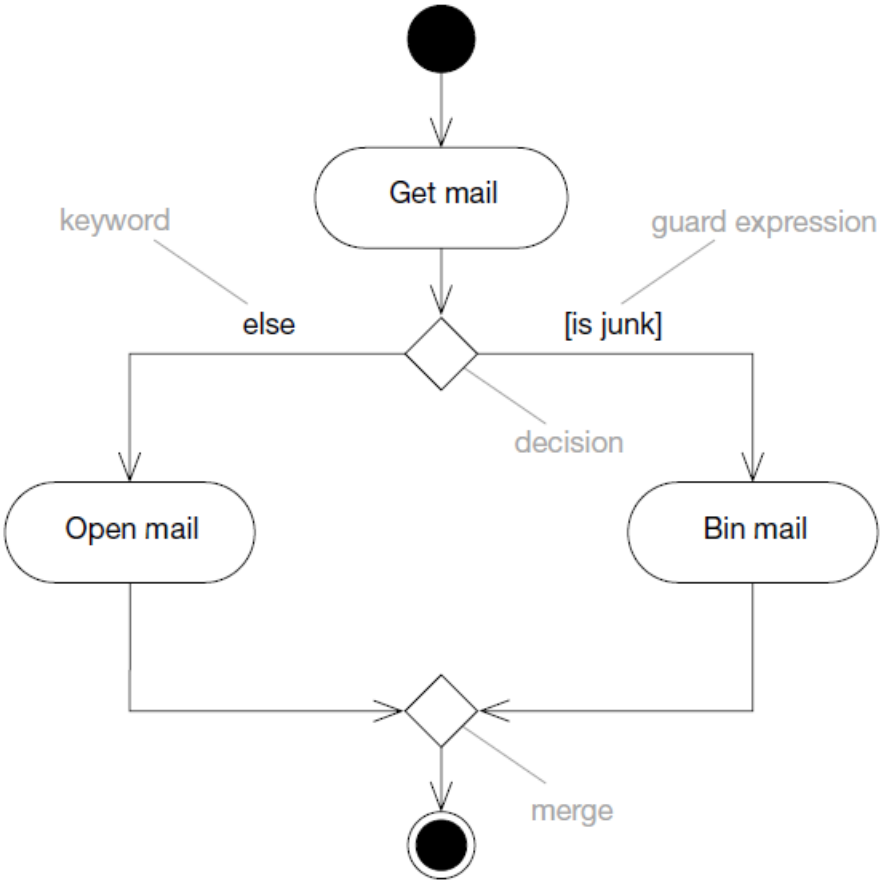
ID / Title	Make Order	
Actor	Customer	
Description	Let the user choose food	
Pre-condition	Logged in – View restaurant list	
Post-condition	Filled Cart	
Flow of event	**Actor**	**System**
	1) choose food type	
		2) Fetch and show food page
		3) show add food form
	4) Fill the form	
	5) click submit form button	
		6) Update Cart
		7) show confirmation message
	8) Repeat from 1 to 7	
	9) Submit order	
		10) Open Submit order form
Critical Scenario	**Scenario**	**System response**
	1) Missing required fields	Show msg “pls fill all the required information”
	2) Exit without saving	Show msg “are you sure you want to exit without saving”
	3) Fill an error field	Show msg “please fill in the correct values and types”
	4) Submit an empty cart	Hide submit button

Activity Diagram:

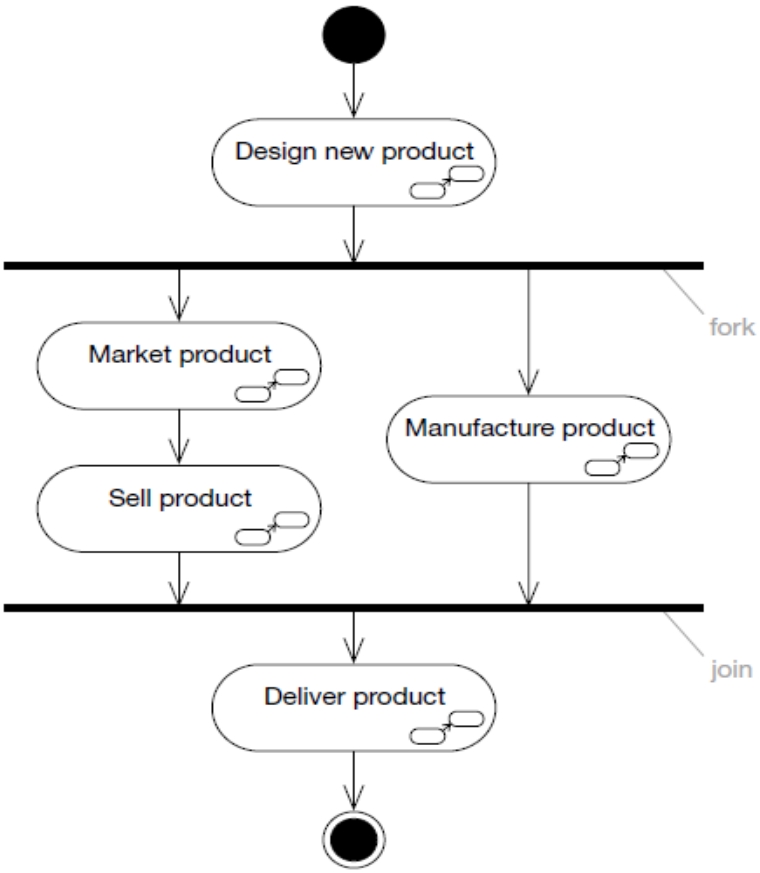
نمثله لكل خدمة وهو يبين طريق الخدمة من البداية الى النهاية وعادة نشتقه من تدفق الحدث من مخطط التوصيف.



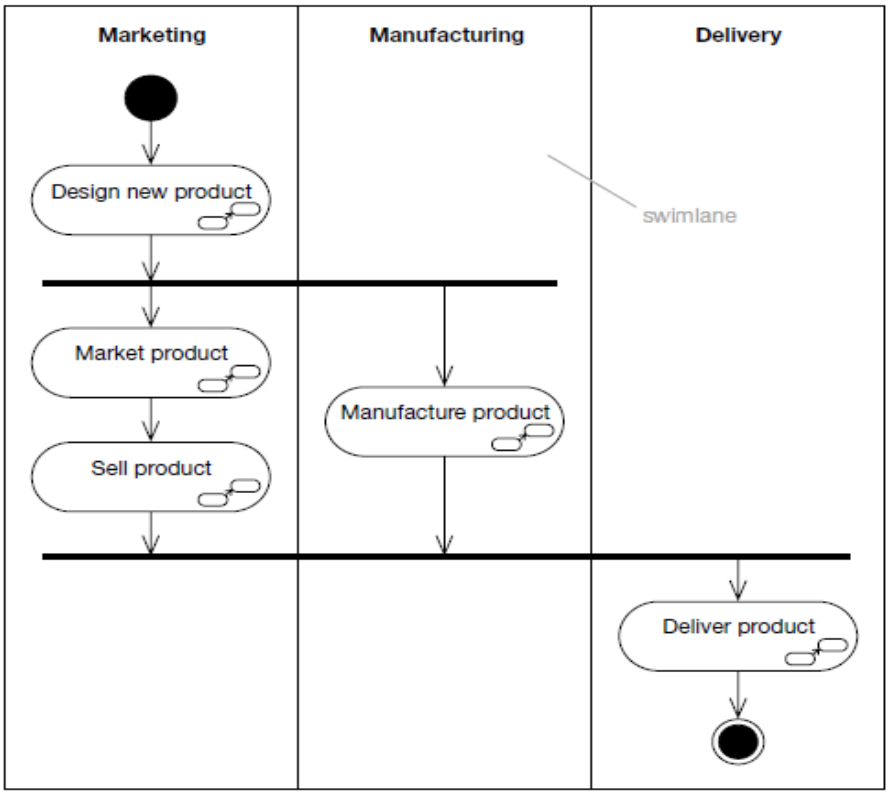
Activity diagram elements (decisions and merge)



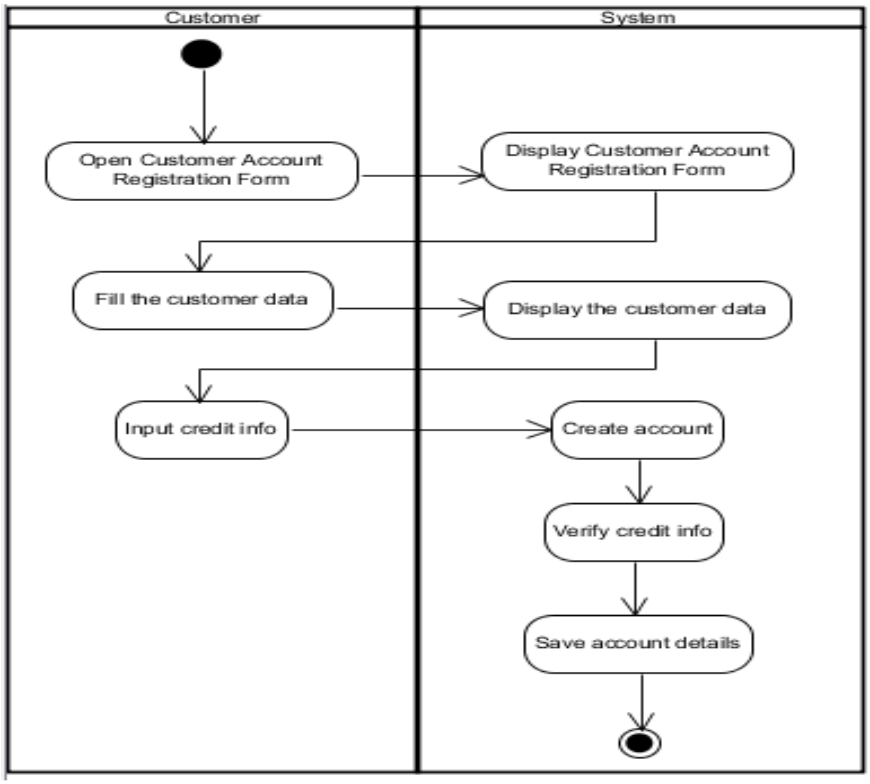
Activity diagram elements (Forks and Joins)



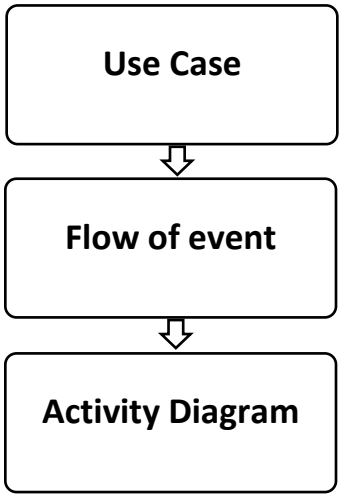
Activity diagram elements Swim lanes



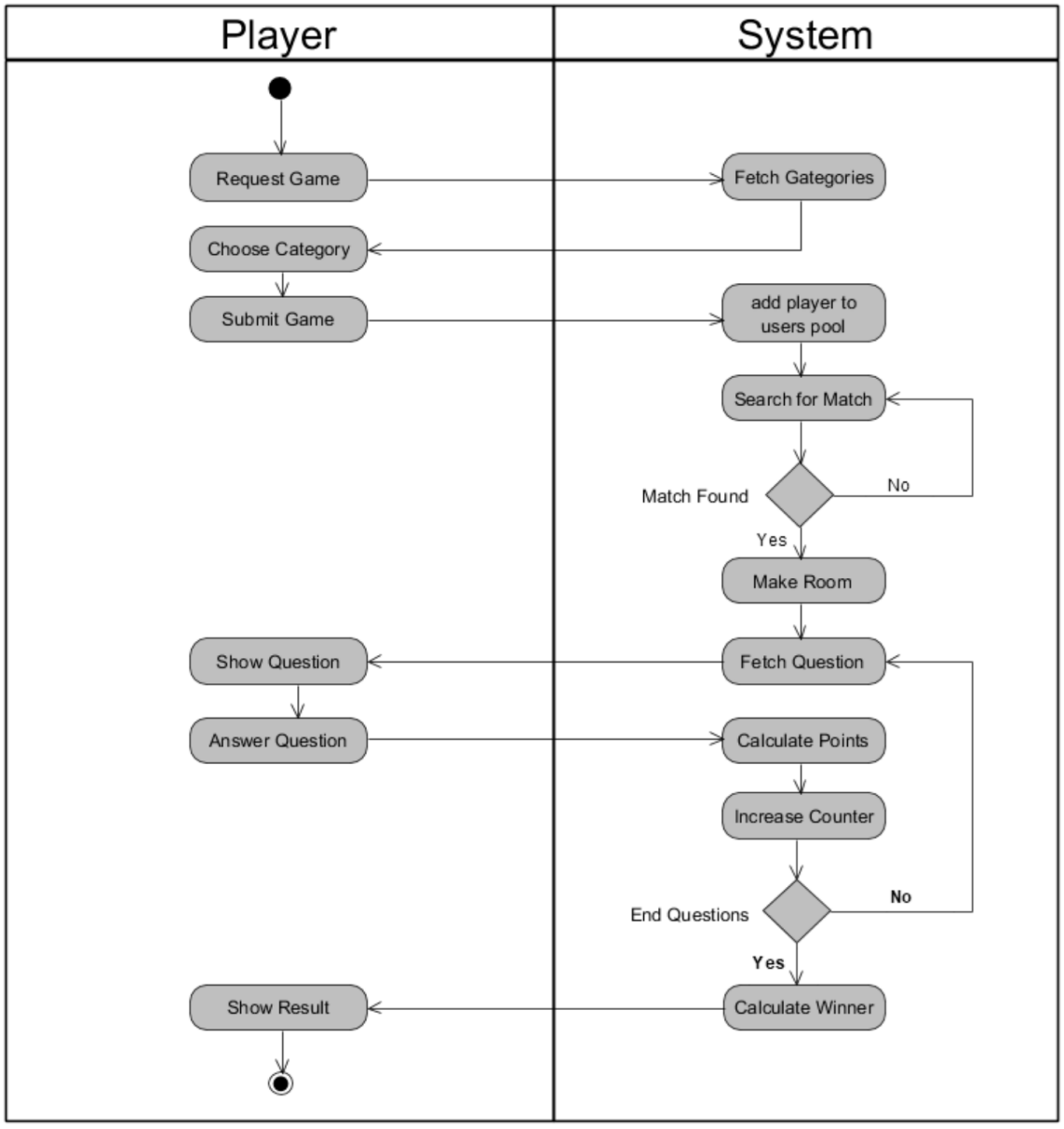
Example create customer account use case



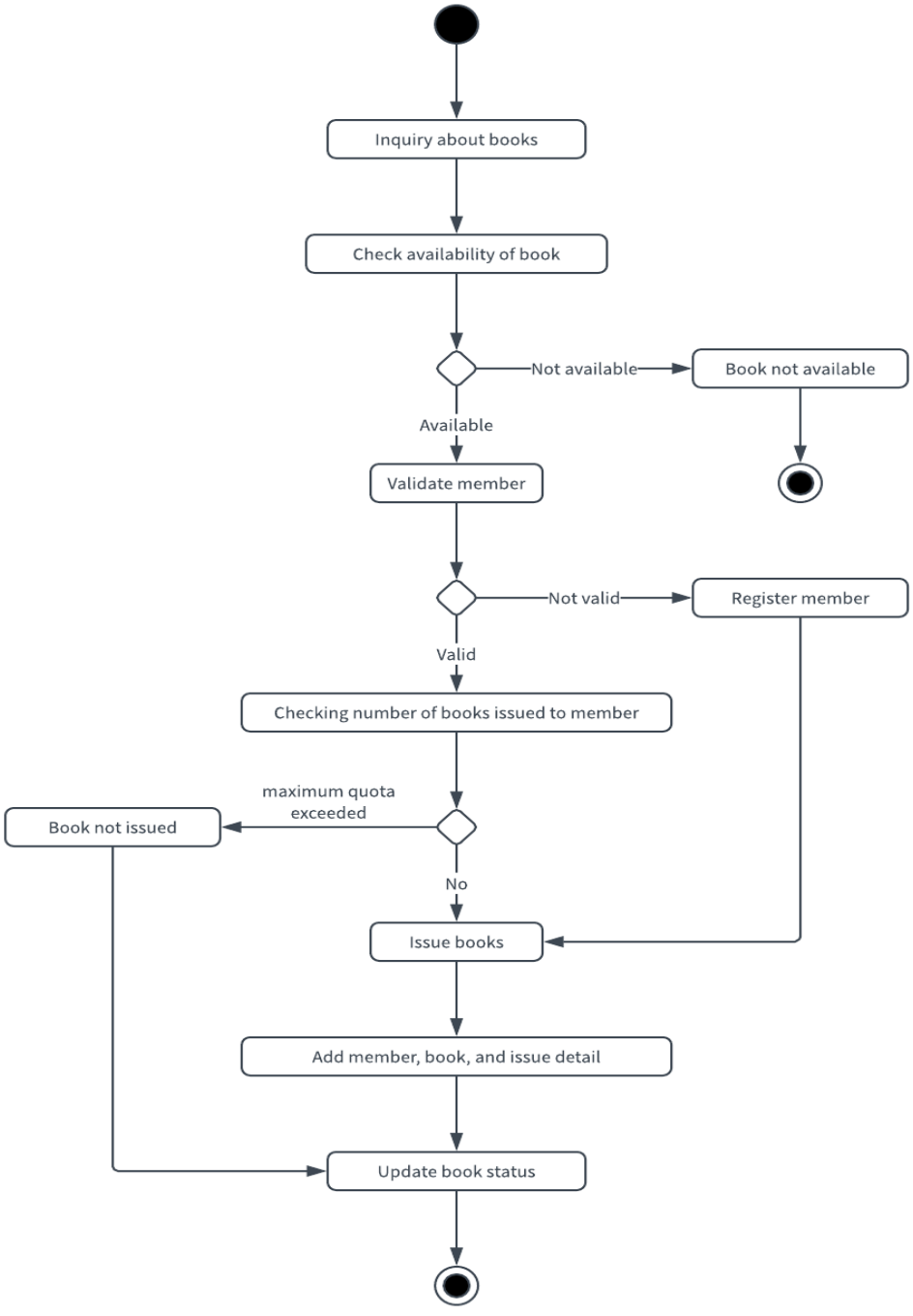
Activity Diagram from use case specification:



Example Quiz Up



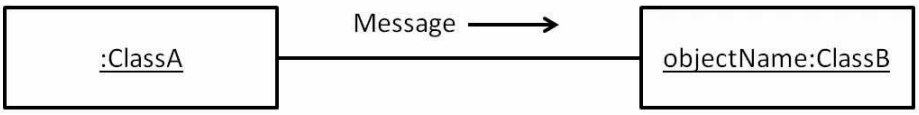
Example Library management system



المخطط التفاعلي Collaboration Diagram

يعبر عن صورة لحظية من خدمة معينة

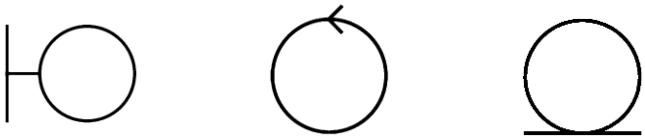
يعبر عن الرسائل التي ستحدث داخل النظام وتفاعلها وترتيبها وتفاعل الكائنات مع بعضها.



Collaboration Diagram Elements & Kinds:

- Object
- Links
- Messages

Collaboration Diagram object kinds:



Boundary Controller Entity

- **Boundary:**

لها علاقة بالواجهة او بالعرض

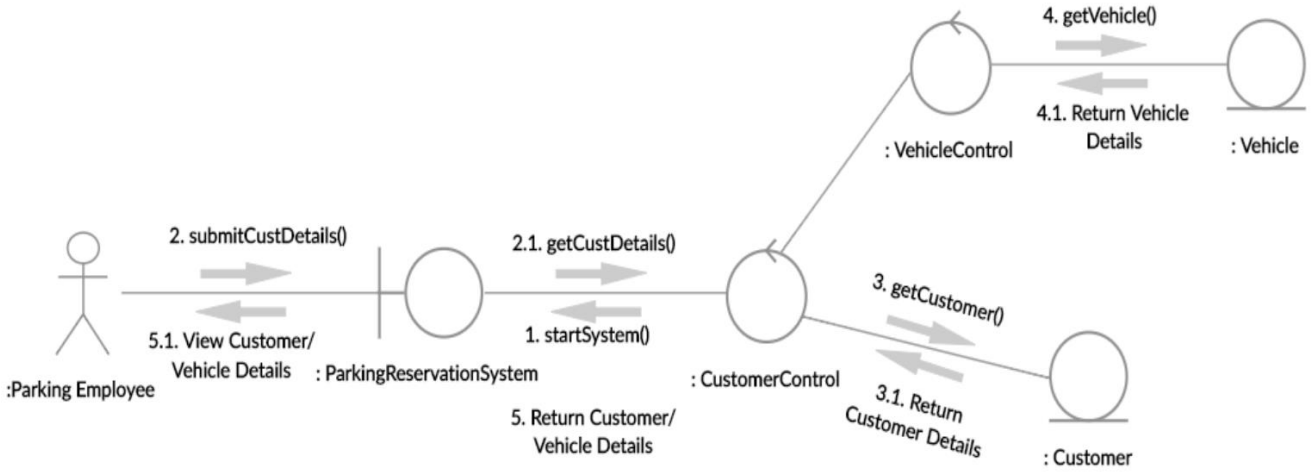
- **Controller:**

لها علاقة بالتحكم

- **Entity:**

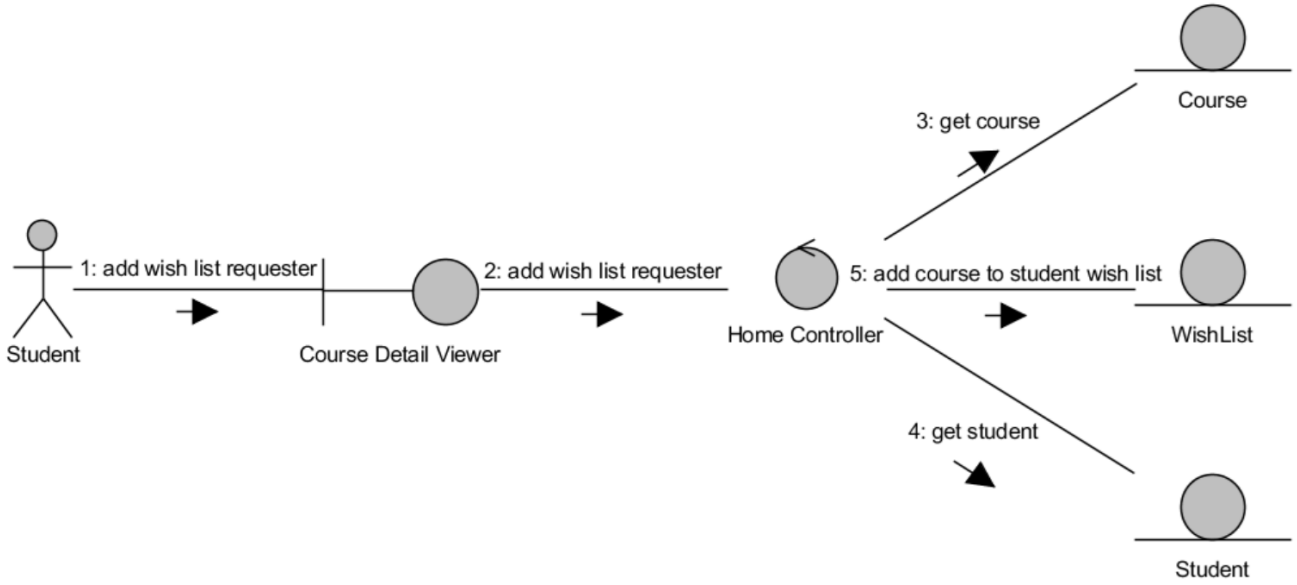
لها علاقة بقاعدة البيانات

Collaboration Diagram Parking System Example

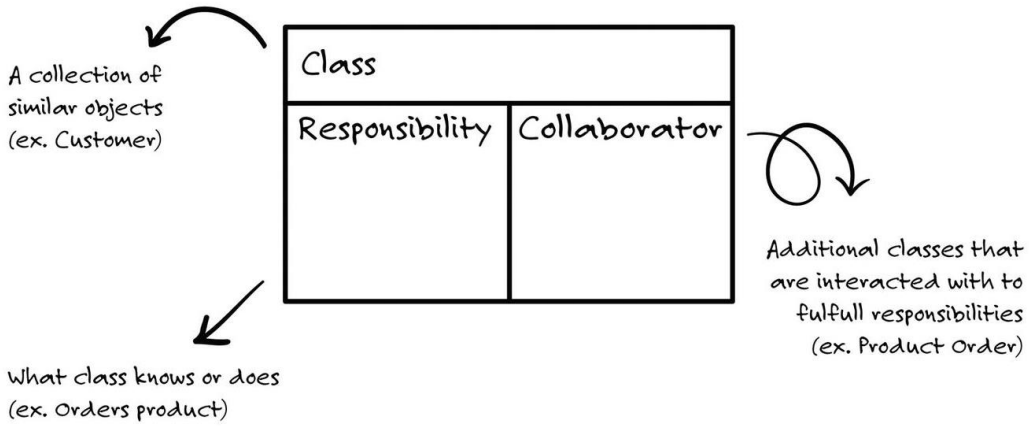


From Use Case to Collaboration

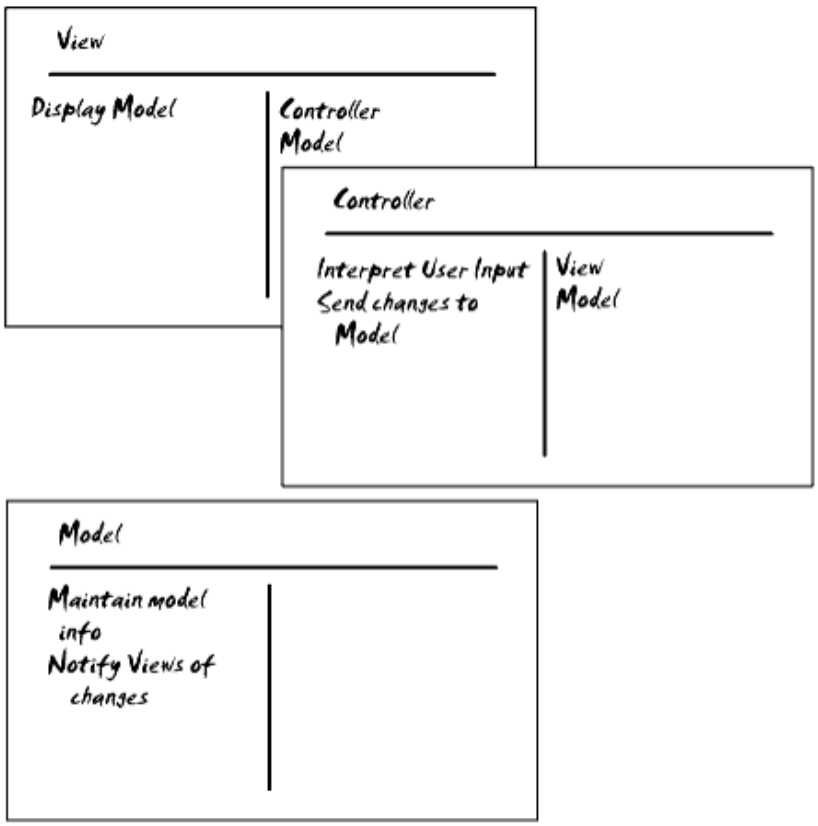
ID / Title	Add to Wish list	
Actor	Student	
Description	Allow student to add a course to his wish list	
Pre-condition	Logged in – Course details page is open	
Post-condition	Add course to student wish list	
Flow of event	**Actor**	**System**
	1) Click wish list button	
		2) Add course to student
Critical Scenario	**Scenario**	**System response**
	None	



Class Responsibility Collaboration Cards (CRC CARDS)

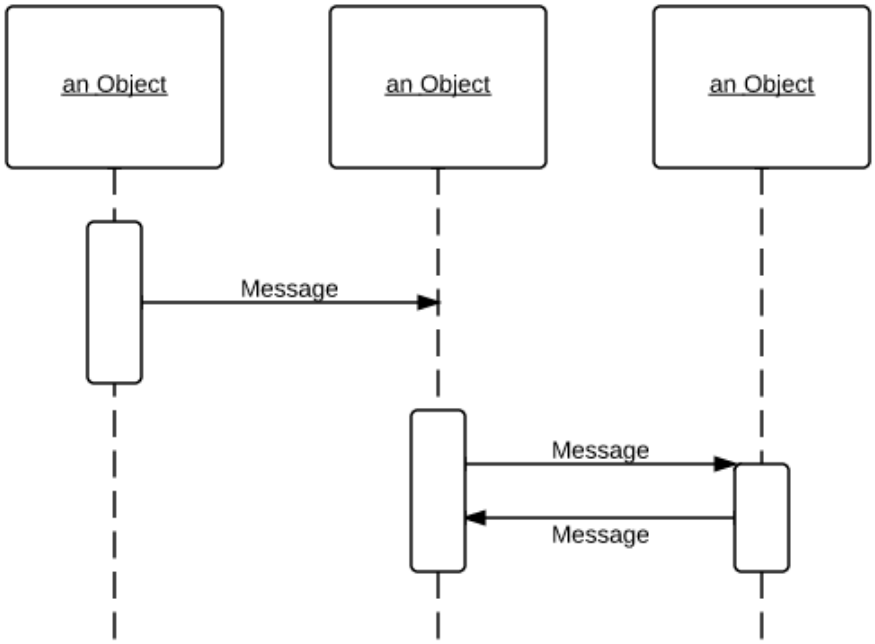


Example:

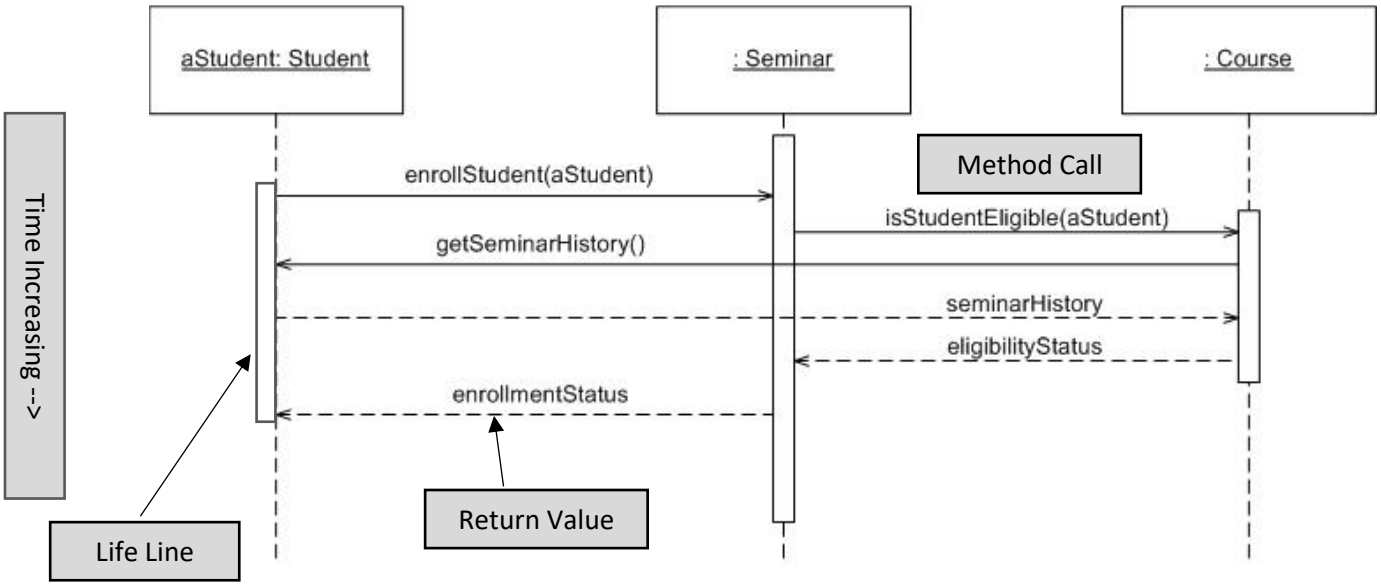


Sequence Diagram

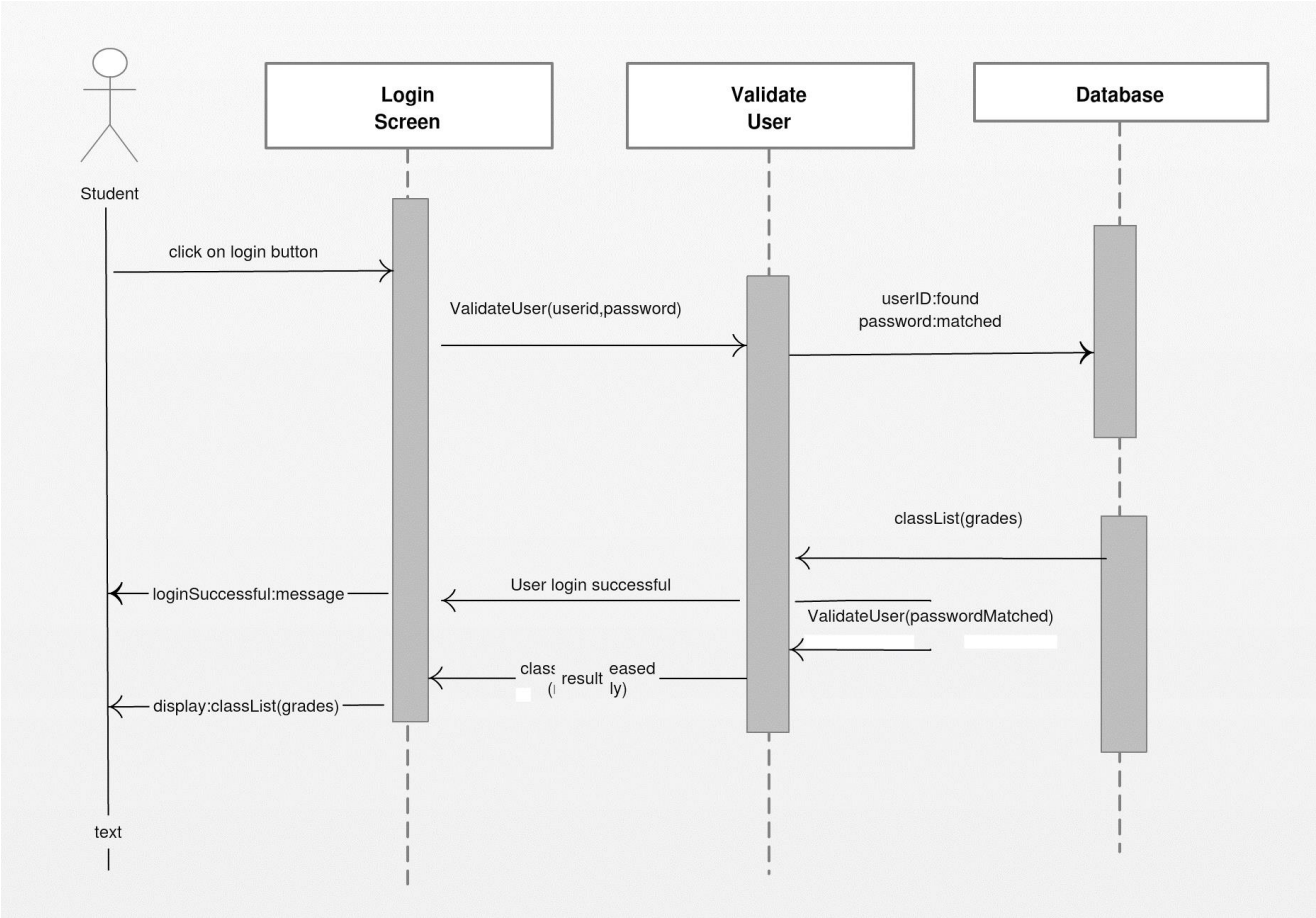
يوضح تفاعل الكائنات مع بعضها ضمن خطة زمنية.



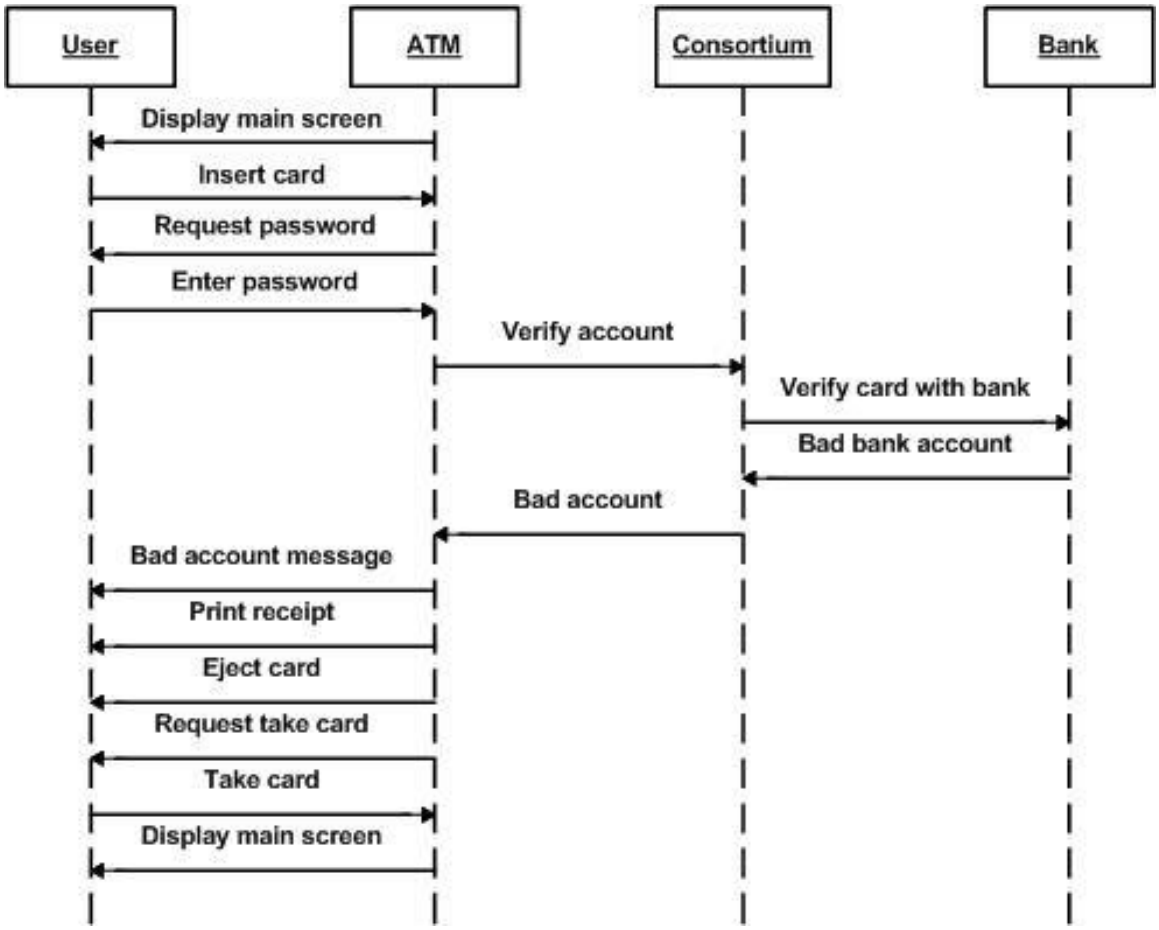
Sequence Diagram Notation:



Sequence Diagram Example



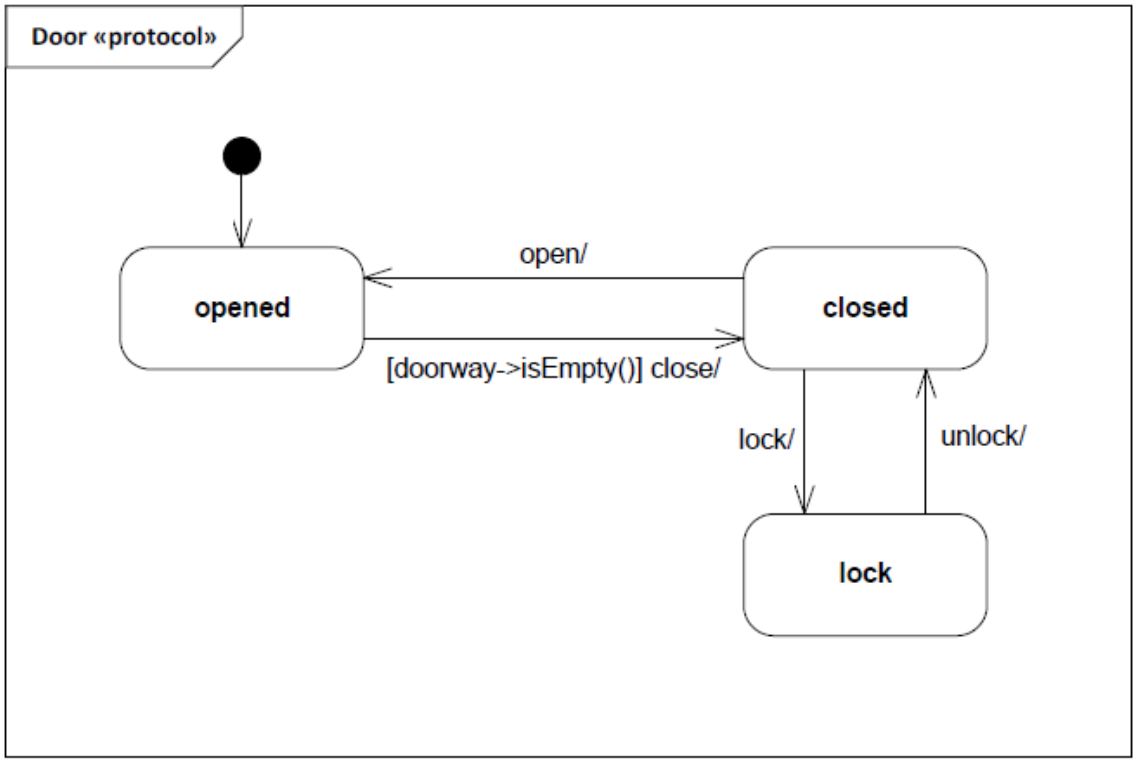
ATM Sequence Diagram



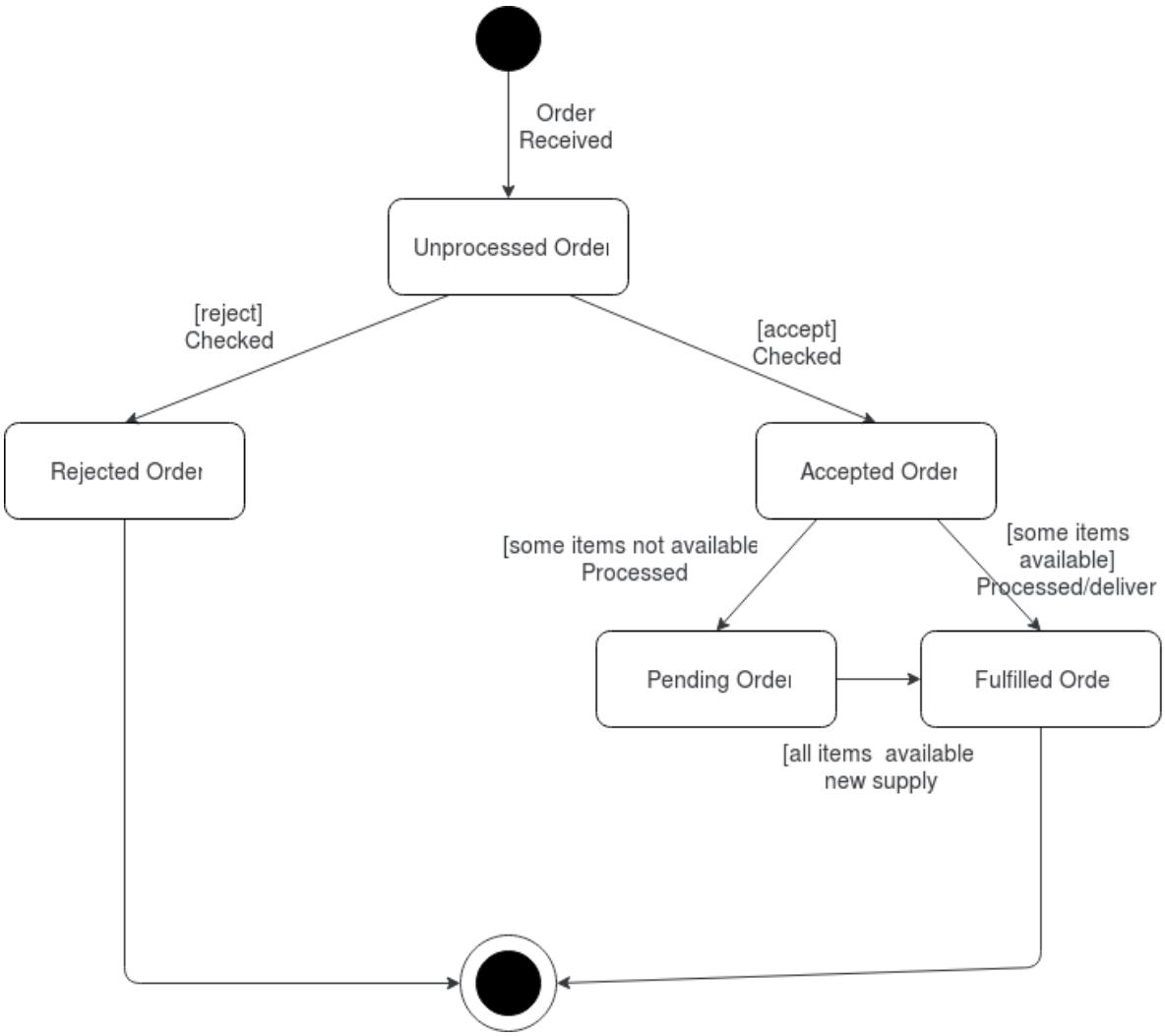
State Machine Diagram

يظهر مخطط الحالة تفاعل مكونات الكائن الواحد مع بعضها (دورة حياة الاوبجكت)

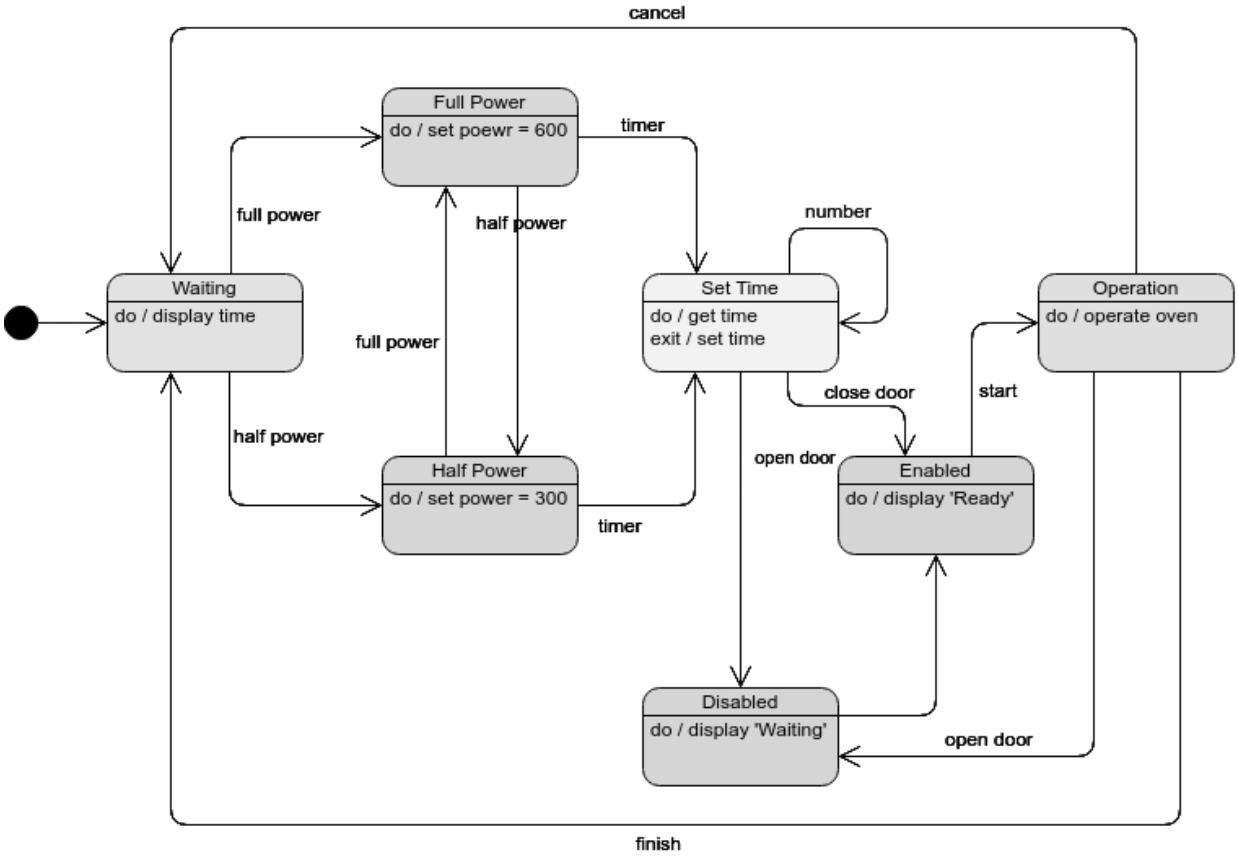
Example:



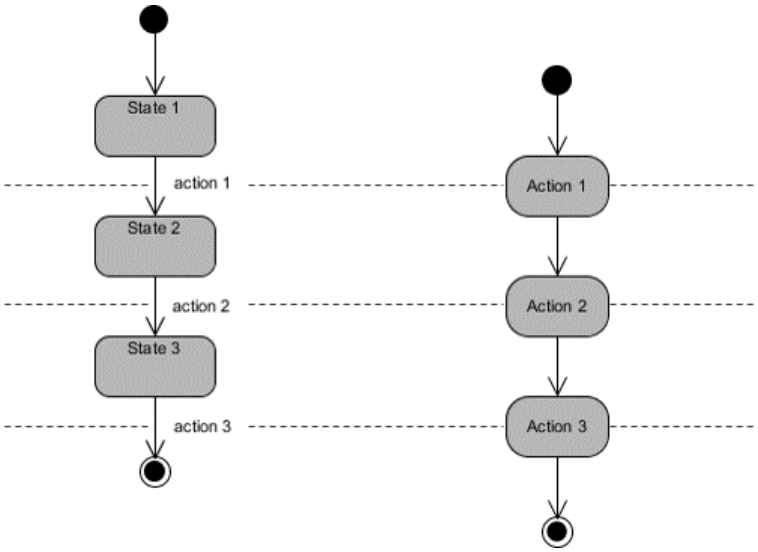
State Machine Diagram Example



State Machine Diagram Example microwave



Activity Diagram VS State Diagram



Architecture design التصميم المعماري

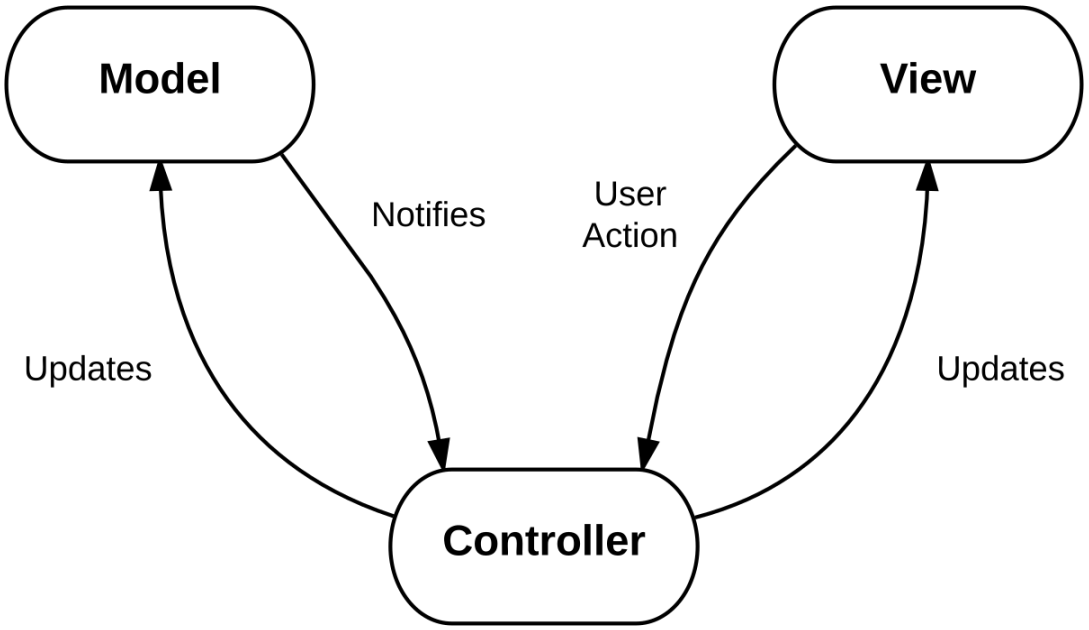
هو كيفية تقسيم النظام من الخارج وليس الكود وعادة يقسم النظام الى عدة طبقات وكل طبقة تهتم بوظائف معينة ويوجد لدينا أنماط عديدة للتصميم.

Architecture pattern:

هي مجموعة من الأنماط الجاهزة التي يتم اكتشافها اثناء تطوير البرمجيات فحسب صفات معينة خاصة بتطبيقنا نختار النمط المعماري المناسب له ومنها:

- **Model-View-Controller (MVC):**

يتم تقسيم النظام لثلاث طبقات طبقة تهتم بالواجهات وطبقة تهتم بالتعامل مع قواعد البيانات وطبقة تكون صلة الوصل بينهما وهذا النمط يكثر استخدامه في بناء تطبيقات الويب.



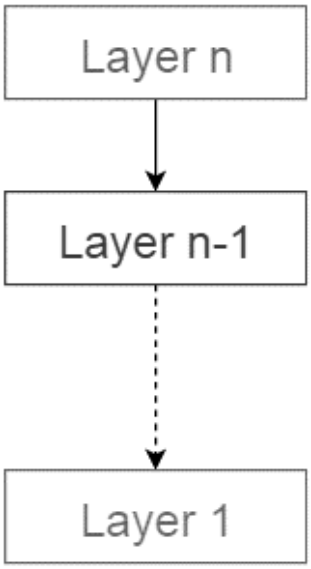
Layered pattern

The most commonly found 4 layers of a general information system are as follows.

- **Presentation layer** (also known as **UI layer**)
- **Application layer** (also known as **service layer**)
- **Business logic layer** (also known as **domain layer**)
- **Data access layer** (also known as **persistence layer**)

Usage

- General desktop applications.
- E commerce web applications.

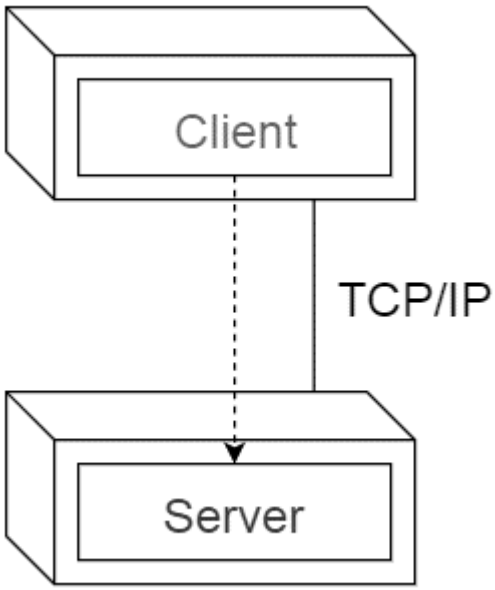


Client-server pattern

This pattern consists of two parties; a **server** and multiple **clients**. The server component will provide services to multiple client components.

Usage

- Online applications such as email, document sharing and banking.



Class Diagram

يعد مخطط الفئة(الكلاس) من المخططات البنيوية التي تصف بنية النظام من خلال اظهار فئات النظام وخصائصها وعملياتها والعلاقات بينها.

يتكون من:

- مجموعة من الفئات.

- مجموعة من العلاقات بين الفئات.

Class Notation:

1. **Class Name:**
- The name of the class appears in the first partition.

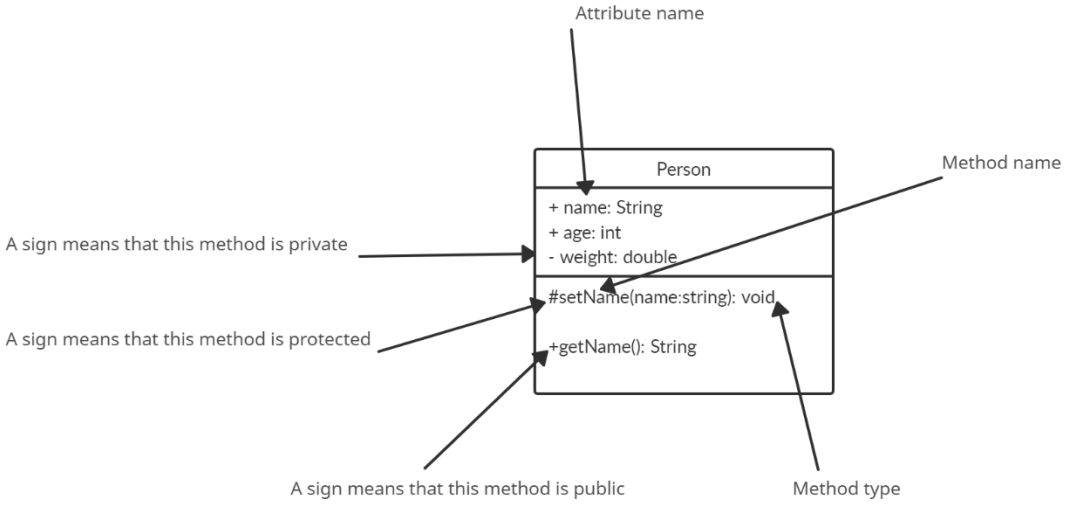
2. **Class Attributes:**
- Attributes are shown in the second partition.

3. **Class Operations (Methods):**
- Operations are shown in the third partition. They are servicing the class provides.

Visibility of Attributes and Operations:

- \+ denotes public attributes or operations.
- \- denotes private attributes or operations.
- # denotes protected attributes or operations.
- ~ denotes package attributes or operations.

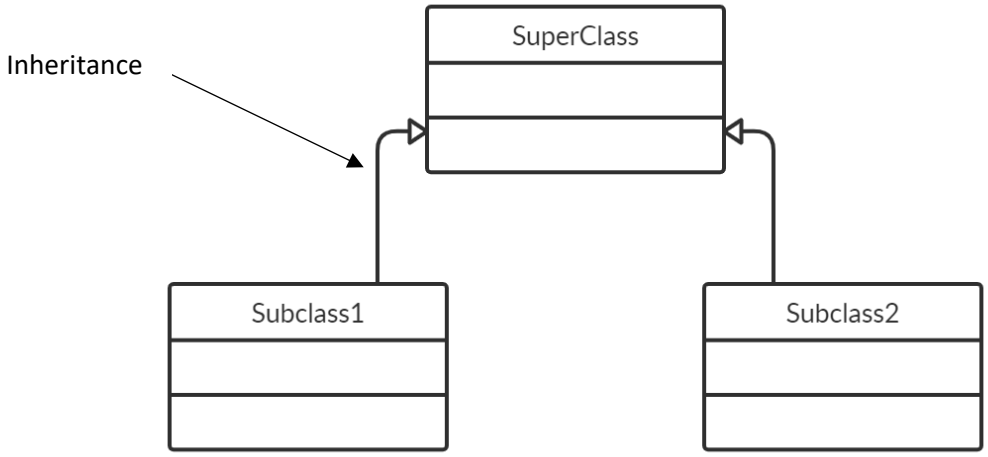
Access Right	public (+)	private (-)	protected (#)	Package (~)
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same package



Class Relationships:

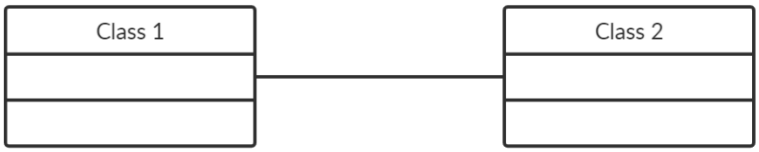
1- Inheritance (Generalization):

- SubClass1 and SubClass2 are specializations of Super Class.



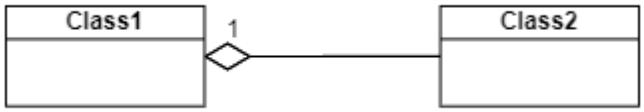
2- Simple Association:

- There is an association between Class1 and Class2



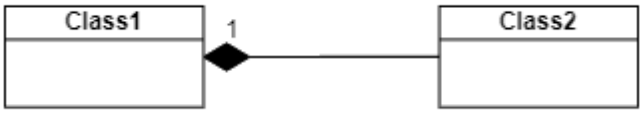
3- Aggregation:

- Aggregation is a weak Association
- Class2 is part of Class1



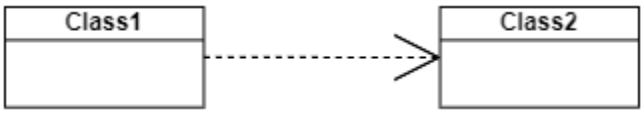
4- Composition:

- Composition is a strong Association.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.

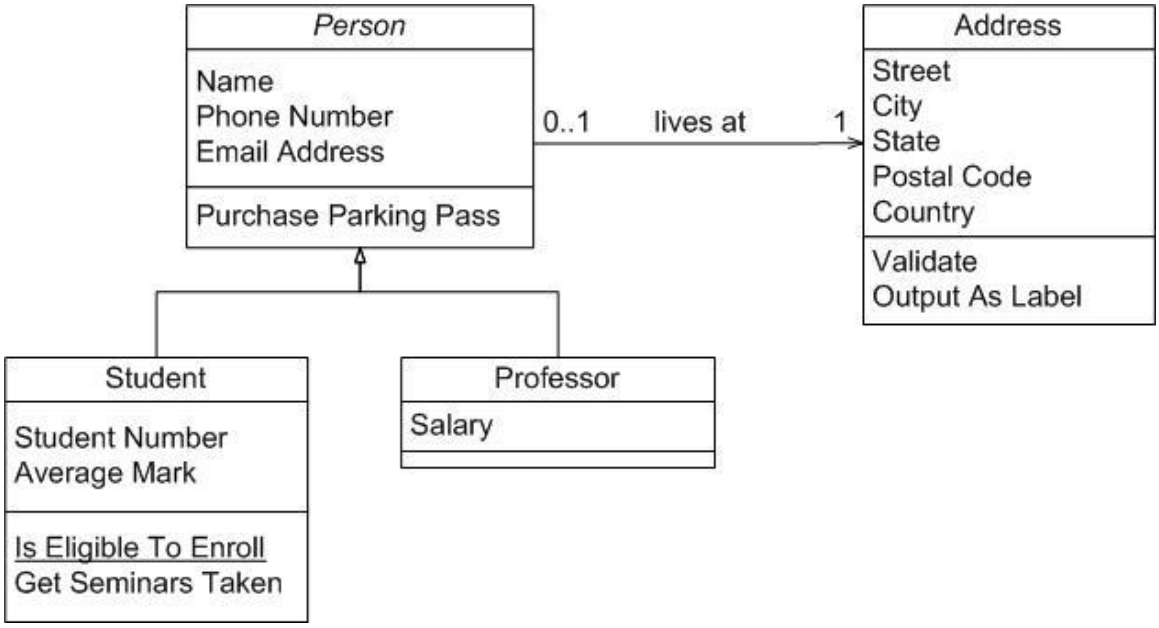


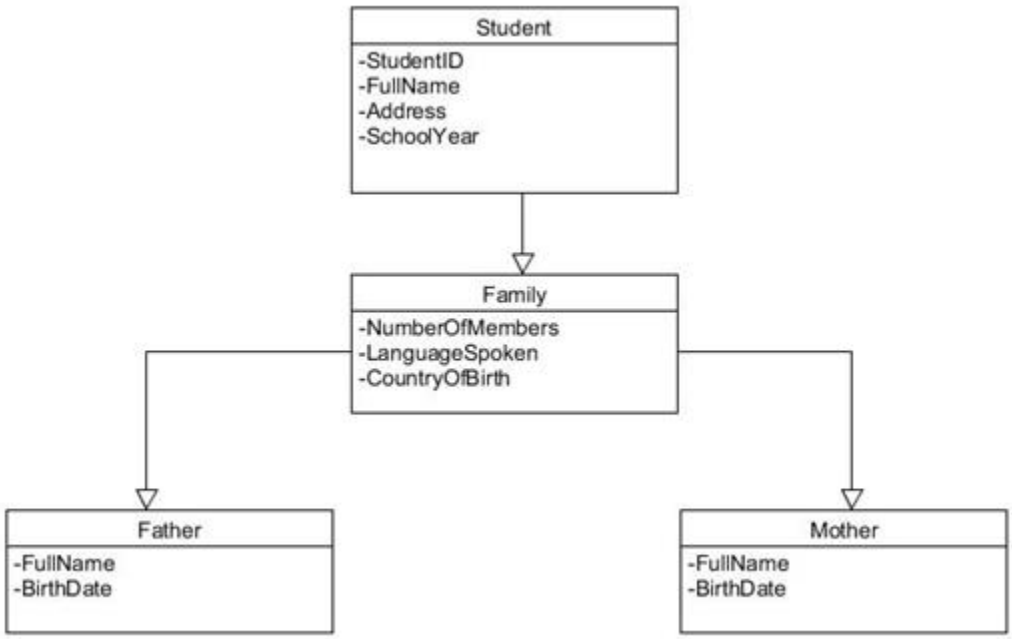
5- Dependency:

- Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2



Class Diagram Examples





Dependency Management إدارة الاعتمادية

- **What is dependency management?**

- A simple idea - as interdependencies increase, features like: reusability, flexibility, and maintainability decrease.
- Dependency management is controlling interdependencies.

تعني زيادة الاعتمادية المتبادلة بين الفئات وتتحكم في الترابط بينها.

ميزاتها:

- إعادة الاستخدام – المرونة – قابلية الصيانة.

- **What bearing does DM have on software?**

- Coupling and cohesion are the eternal concerns of software development.
- One can say that OO is just a set of tools and techniques for Dependency Management.

تأثير إدارة الاعتمادية على البرمجيات:

- الاقتران والتماسك هما الاهتمامات الأبدية لتطوير البرمجيات.
- يمكن للمرء أن يقول إن البرمجة كائنية التوجه هي مجرد مجموعة من الأدوات والتقنيات لإدارة التبعية.

What is the benefit of good DM?

- **Not Rigid**
  - Modules can be changed independently.
  - Changes are focused into a small set of modules.
  - The cost of a change can be estimated.
  - Changes can be planned with more reliability.

غير جامد:

- يمكن تغيير الوحدات بشكل مستقل.
- تتركز التغييرات في مجموعة صغيرة من الوحدات.
- يمكن تقدير تكلفة التغيير.
- يمكن تخطيط التغييرات بمزيد من الموثوقية.

- **Not Fragile**
  - Limited propagation means fewer changes: less risk of introducing defects.
  - New defects are likely to be in the same conceptual area.
  - Risks are more predictable, Credibility remains high.

غير هشة:

- مخاطر اقل لحدوث عيوب.
- المخاطر أكثر قابلية للتنبؤ.
- المصداقية عالية.

- **Reusable**
  - Modules can be separated: modules with desirable qualities do not depend on those with undesirable qualities.

إعادة الاستخدام:

- يمكن فصل الوحدات: لا تعتمد الوحدات ذات الصفات المرغوبة.

- **Low viscosity**

قليل اللزوجة:

- من السهل إجراء التغييرات.
- النظام أكثر قابلية للتغيير.

Class Design Principles مبادئ تصميم الفئات

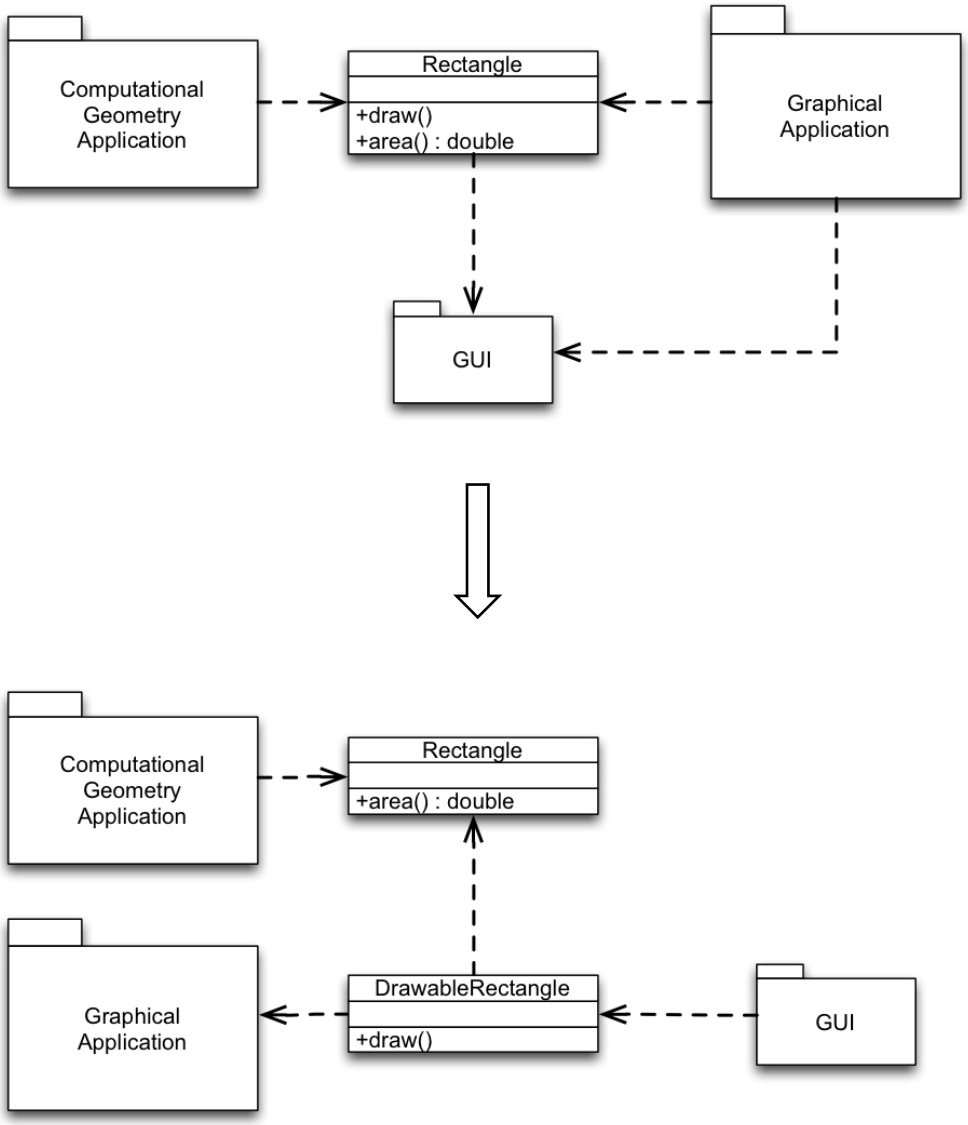
SOLID:

- **S**RP: The Single Responsibility Principle.
- **O**CP: The Open/Closed Principle.
- **L**SP: The Liskov Substitution Principle.
- **I**SP: The Interface Segregation Principle.
- **D**IP: The Dependency Inversion Principle.

The Single Responsibility Principle:

- A class should have one, and only one, reason to change.

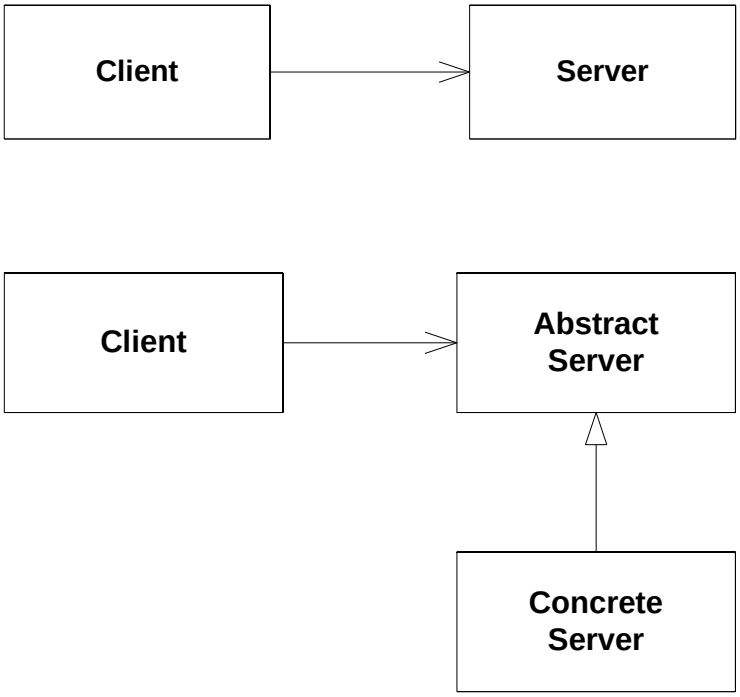
مبدأ المسؤولية الفردية:
يجب ان يكون للفئة(الكلاس) سبب واحد للتغيير.



Open/Closed Principle:

- A principle which states that we should add new functionality by adding new code, not by editing old code.
- Defines a lot of the value of OO programming
- Abstraction is the key

- إضافة الوظائف الجديدة يكون بإضافة كود جديد وليس التعديل على القديم.

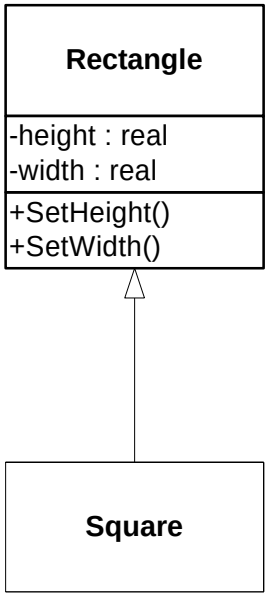


Liskov Substitution Principle:

- All derived classes must be substitutable for their base classes.

- يجب أن تكون جميع الفئات المشتقة قابلة للاستبدال بفئاتها الأساسية.
- هو توسعة لمبدأ المفتوح المغلق ويعتمد على الفئات المجردة.

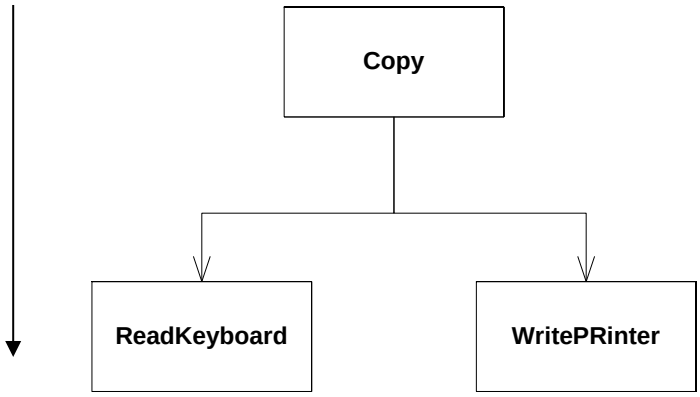
```
void Square::SetWidth(double w)
{
     width = w;
     height = w;
}
void Square::SetHeight(double h)
{
     width = h;
   height = h;
}
```

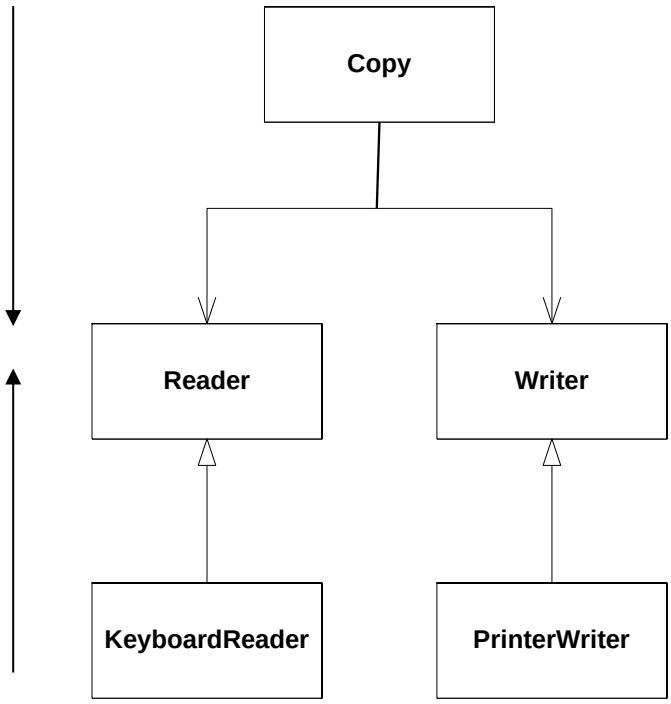


Dependency Inversion Principle:

- Details should depend on abstractions. Abstractions should not depend on details.

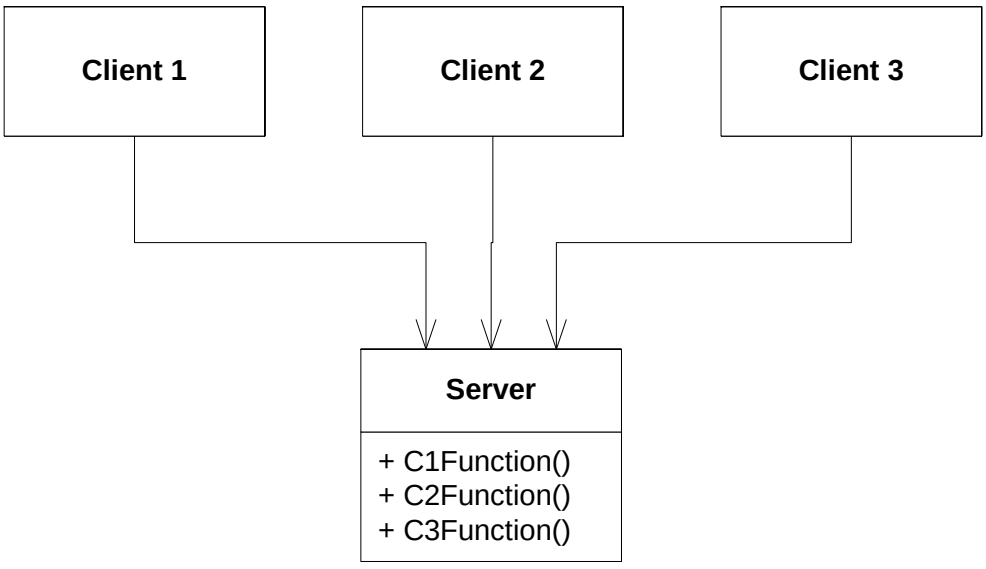
يجب أن تعتمد التفاصيل على الأفكار المجردة اي يجب ألا تعتمد التجريدات على التفاصيل

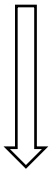


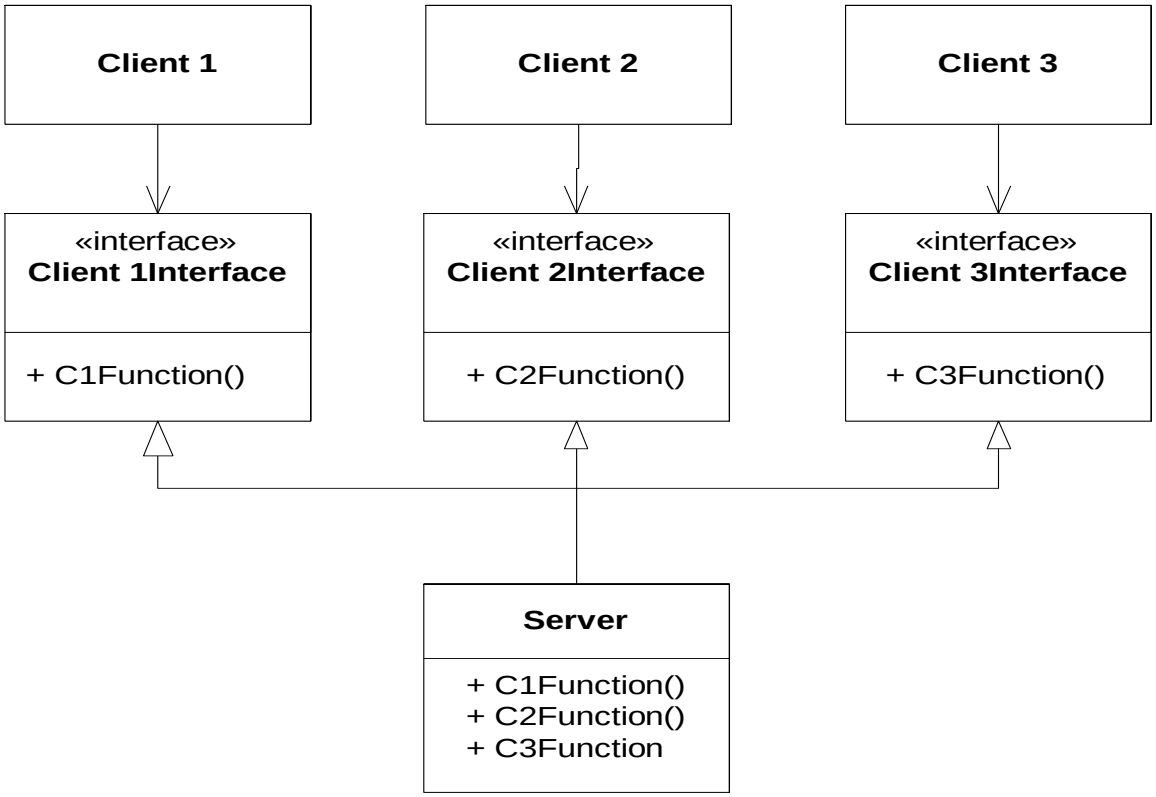


Interface Segregation Principle:

- Sometimes class methods have various groupings.
- These classes are used for different purposes.
- Not all users rely upon all methods.
- This lack of cohesion can cause serious dependency problems.
- These problems can be refactored away.






Client 1
Client 2
Client 3
«interface»
Client 1Interface
+ C1Function()
«interface»
Client 2Interface
+ C2Function()
«interface»
Client 3Interface
+ C3Function()
Server
+ C1Function()
+ C2Function()
+ C3Function

ATM UI Example