المحاضرة الثانية
المنحى الكائني

د. فادي تركاوي

- في هذه المحاضرة سوف نلقي نظرة على مفهوم المنحى الكائني Object Orientation (OO) .
لقد تمّ تصميم لغة النمذجة الموحّدة UML بحيث تدعم المنحى الكائني ، لذلك سنقوم بتعريف مفاهيمه قبل التعمق في لغة UML ، و استكشاف المزايا التي قد يوفرّها.
- سوف نستعمل جملة المنحى الكائني للتعبير عن المنحى الكائني للتصميم أو/ و المنحى الكائني للبرمجة.

البرمجة الهيكلة (أحيانا يُسمّى وظائفي Functional)

- في البرمجة المهيكلة Structured Programming، الطريقة العامة المتّبعة هي النظر إلى المسألة، ثم تصميم مجموعة من الوظائف functions التي يمكنها انجاز المهام المطلوبة لحلها. إذا تضخّمت هذه الوظائف ، يتم تجزئتها حتى تصير صغيرة بالحدّ الذي يتيسّر فيه مناولتها و فهمها. هذه العملية تدعى التفكيك الوظائفي functional decomposition.
- ستحتاج معظم الوظائف إلى بيانات من نوع ما لتعمل عليها. البيانات في الأنظمة الوظائفية عادة ما يحتفظ بها في قاعدة بيانات من نوع ما (أو قد يحتفظ بها في الذاكرة كمتغيّرات عامة global variables).
- لنأخذ مثالا بسيطا، تخيّل منظومة لإدارة معهد، هذه المنظومة تحتفظ ببيانات الطلبة و المدرّبين في المعهد إضافة للمعلومات حول الدورات المتوفرة، كذلك تقوم المنظومة بتتبع كل طالب و المقررات التي التحق بها.

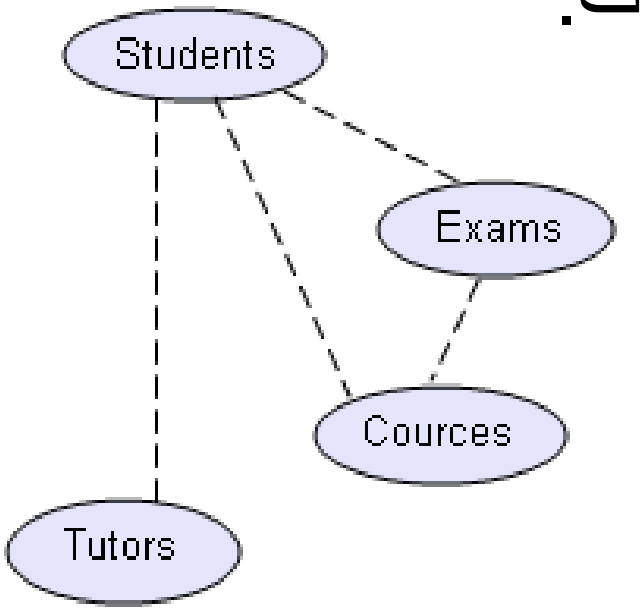
Function الوظائف

- التصميم الوظائفي المحتمل سيتضمّن كتابة الوظائف functions التالية:

add_student إضافة طالب
enter_for_exam دخول امتحان
check_exam_marks فحص علامات امتحان
issue_certificate إصدار شهادة
expel_student طرد طالب

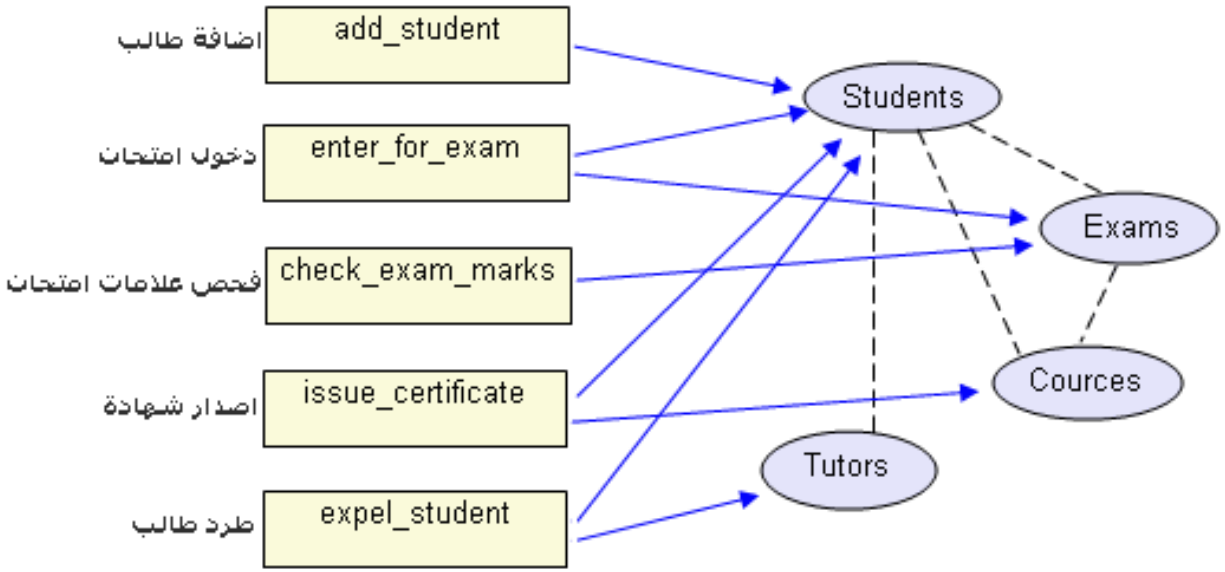
نموذج البيانات Data Model

- سوف نحتاج أيضا إلى نموذج بيانات data model ليمثّل هذه الوظائف. نحتاج لتخزين معلومات عن الطلبة، و المدربين و الامتحانات و المقررات، لذا يجب علينا تصميم مخطط قاعدة بيانات database schema للاحتفاظ بهذه البيانات.



- الآن بدأ واضحا أن الوظائف functions التي حدّدناها سابقا سوف تعتمد على هذه المصفوفة من البيانات. مثلا، وظيفة "add_student" (إضافة طالب) ستقوم بتغيير محتويات "Students" (طلبة)، و وظيفة "issue_certificate" (إصدار شهادة) تحتاج إلى الوصول إلى بيانات طالب "Student" (لمعرفة تفاصيل الطالب الذي يحتاج للشهادة) و ستحتاج الوظيفة أيضا إلي بيانات الامتحانات "Exam" .

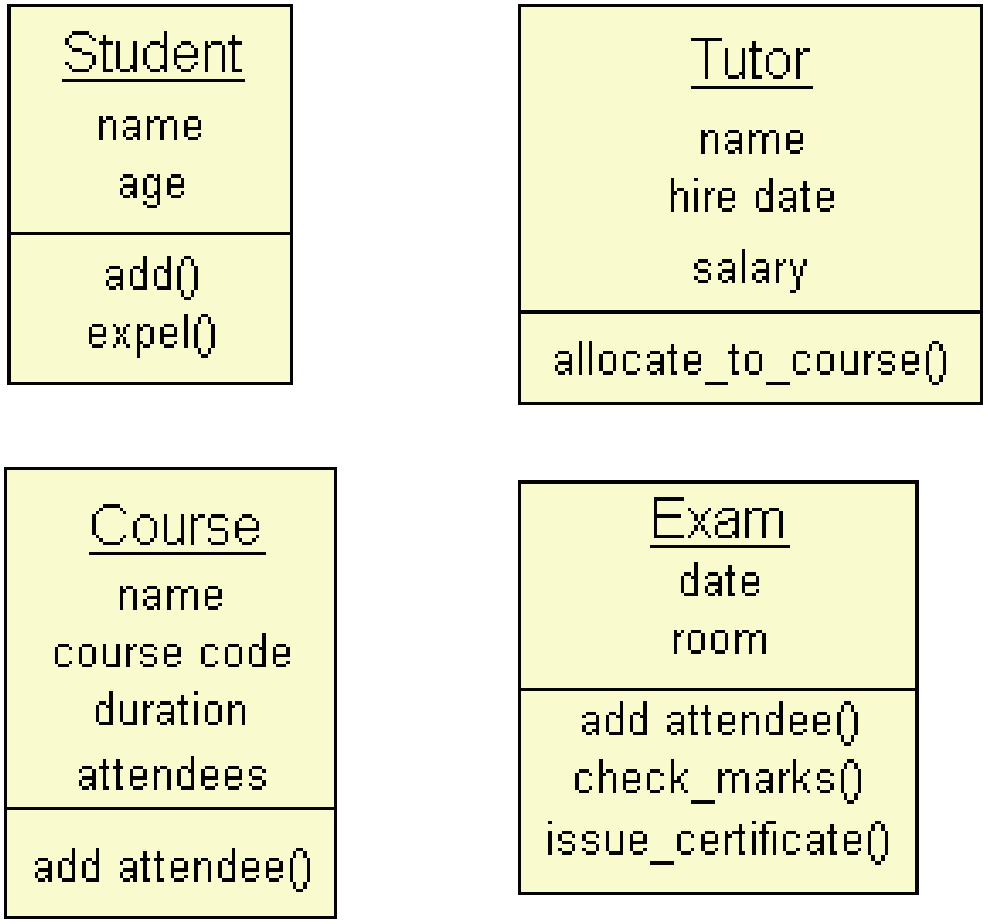
- المخطط diagram التالي عبارة عن رسم لكل الوظائف، مجتمعة رفق البيانات، و رسمت الخطوط فيها حيثما وجدت اعتمادية



- المشكلة مع هذه المقاربة؛ أن المسألة التي نتعامل معها إذا ما **تعقدت أكثر فإن صعوبة المحافظة على المنظومة** و صيانتها ستزداد. فإذا أخذنا المثال أعلاه، ماذا سيحدث لو تغيّرت المتطلبات requirements بطريقة تؤدّي إلى تغيير أسلوب مناولة بيانات الطالب Student.
- كمثال، لنتخيّل أن منظومتنا تعمل على أكمل ما يكون، لكننا اكتشفنا أن تخزين تاريخ ميلاد الطالب على شكل عدد ذو خانتين كي يمثل السنة كانت فكرة سيّئة، الحل المبدئي هنا هو أن نقوم بتغيير حقل تاريخ الميلاد في جدول الطلبة Students من خانتين إلى أربع خانات لرقم السنة.

أسلوب المنحى الكائني

- يحاول المنحى الكائني "OO" التقليل من تأثير هذه المشكلة عن طريق الجمع بين البيانات data و الوظائف functions ذات العلاقة في قالب module واحد.
- بالنظر إلى الشكل السابق ، يبدو واضحا وجود علاقة بين البيانات و الوظائف؛ فمثلا وظيفتا:
  add_student و expel_student مرتبطتان بقوة ببيانات Student الطالب.
- الشكل التالي يبيّن التجميعات الكلّية للبيانات و الوظائف ذات العلاقة على شكل قوالب:


Student
name
age
add()
expel()
Tutor
name
hire date
salary
allocate_to_course()
Course
name
course code
duration
attendees
add attendee()
Exam
date
room
add attendee()
check_marks()
issue_certificate()

- بعض النقاط الجديرة بالملاحظة حول نظام القولبة الجديد في البرمجة:
- أكثر من تمثل لنفس القالب يمكن استحضاره عند تشغيل البرنامج. في منظومة المعهد، سيكون هناك تمثيل لقالب Student لكل طالب يتبع المعهد يجرى التعامل معه في المنظومة. و كل تمثل سيكون له بياناته الخاصة. (بالطبع لكل تمثل اسما مختلفا).
- القوالب يمكنها "التخاطب" مع قوالب أخرى عن طريق استدعاء وظائفها. مثلا، عندما يتم استدعاء وظيفية "add" (إضافة) في Student (الطالب) ، سيتم خلق تمثلا جديدا لقالب Student، و بعدها سيتم استدعاء وظيفة "add_attendee" (إضافة مشترك) من التمثل المناسب لقالب "Course" دورة.

التغليف Encapsulation

- الأمر الأساسي هنا، أنه لا يتم السماح بقراءة أو تغيير أي عنصر في البيانات إلا من قبل التمثيّل الذي تتبعه هذه البيانات . فمثلا، التمثل الخاص بقالب المدرّب Tutor لا يمكنه تحديث أو قراءة بيانات "age" العمر داخل قالب Student الطالب. هذا المفهوم يسمّى بالتغليف Encapsulation ، الذي يسهم في جعل هيكل المنظومة أكثر متانة، و يجنّبها الحالة التي تعرّضنا لها سابقا، حيث التغيير البسيط في جزء من البيانات قد يؤدّي إلى تغييرات متتابعة و أكثر عمقا.
- بواسطة هذا التغليف، يمكن للمبرمج الذي يتعامل مع قالب Student مثلا أن يقوم بتغيير بيانات القالب بكل أمان و دون الخشية من وجود قوالب أخرى مرتبطة أو تعتمد على هذه البيانات. قد يحتاج المبرمج إلى تحديث الاجرائيات داخل القالب، و لكن التأثير سيبقى محصورا داخل قالب واحد معزول عن القوالب الأخرى.

الكائنات Objects

- هذه التجميعات من البيانات و الوظائف المرتبطة بأنها قوالب "modules". عموما إذا نظرنا إلى خصائص هذه القوالب سنجد مرادفات لها في العالم الحقيقي.**الكائنات** Objects في عالم الواقع يمكن تمييزها بشيئين : كل كائن في عالم الواقع لديه **بيانات** data و **سلوك** behaviour . فمثلا جهاز التلفاز هو كائن و يعالج البيانات بطريقة تجعلها تنضبط في قناة محددة، يتم تحديد معدّل المسح إلى قيمة معيّنة، كذلك معدّل التباين و شدّة الإضاءة و هكذا. التلفاز أيضا يمكنه أن "يقوم" بأشياء، التلفاز يمكنه التشغيل أو الإقفال، القنوات يمكن تغييرها، و هكذا.