

بسم الله الرحمن الرحيم

المحاضرة الثانية :

مفهوم وفلسفة البرمجة هدفية الغرض (الكائنية) (OOP) Object Oriented :

- كما قلنا وتحدثنا سابقاً في المحاضرة الأولى من أنه لا بد من وجود معيار ومقياس لبرمجة وكتابة الكود من طرف المبرمجين ولنخرج حيل ذو خبرة عالية و مرونة وسهولة . وبسبب هذه الفلسفة ظهر ما يعرف ببرمجة الكائنات الموجهة (OOP) Object Oriented والتي تستند إلى إستخدام المكتبات Library و إستخدام الدوال Function والتي عرفناها قبل ذلك بالبرمجة التركيبية Structured Programming والتي ظهر مفهوم البرمجة بواسطة الكائنات الموجهة على إثرها .
- والآن ما معنى البرمجة الموجهة بالكائنات وكيف نستطيع الحكم على لغة بأنها OOP أو لا ؟

- قبل أن أبدأ أريد أن أسرد لك طرفة سريعة حدثت مع معلمي الأستاذ ((معن الجيتاوي)) فعندما كان يشتغل في شركة WinWord الموجودة هنا في الأردن والتي تعتبر رقم 1 في الشركات البرمجية والتي بواسطتها تم إدخال علم الحاسوب ونظم المعلومات الحاسوبية في المدارس والجامعات . ويصفته Project Manager في تلك الشركة جاءه شخص من شركة IBM الرائدة في عالم الحاسوب . وأراد أن يسأله ويختبر معلوماته وكونه ذو الخبرة العالية في السي ++ وهو في المقابلة فقام بسؤاله السؤال التالي :

هل لغة ++C تعتبر Full OOP أم لا ؟ قال له لا تعتبر بشكل كامل ولكنها تدعمها ، مع العلم أن الجميع يقولون بنعم ((تعتبر OOP بدعم كامل)) فتنبه هذا الشخص إلى ذكاء أستاذي ومعلمي فقال له إنتهت المقابلة بنجاح .

والسبب في ذلك ((لغة ال ++C)) ليست لغة برمجة موجهة بالكائنات التالي :

Encapsulation – 1

Abstraction – 2

Inheritance – 3

Polymorphizem – 4

ولنأتي الى الشروط الأربعة بالتفصيل الممل :

الشرط الأول : Encapsulation

ومعناه هو تجميع أشياء ووضعها كاملة بمكان ما . ومن إسمها تستطيع تمييز ذلك فمعناها الكبسولة ولتعريفها جزأين :

الجزء الأول : هي تعريف مكان (Class) ونضع فيه كل المتغيرات والدوال التي لها علاقة ببعضها ومن ثم نسند إليه إسماً

الجزء الثاني : هذا المكان لا يستطيع أحد الدخول إليه إلا بصلاحيات تحدد من قبل المبرمج فيوجد قسمان رئيسيان لها وهما ال Public وهو القسم العام وال Private وهو القسم الخاص .

الشرط الثاني : Abstraction

وهي إضافة دالة عامة Function داخل ال (Class) لا أقوم بتحديد التفاصيل لها إلا في قسم البرنامج الرئيسي .
فمثلاً :

نريد صنع عدة كائنات (Class) أو Modules في جامعة معينة فينتج من ذلك كائن للدكتور وكائن للطالب وكائن للمدير وكائن للعامل و ... الخ وأردنا أن تجري بحث فيه من خلال البرنامج الرئيسي فأتت مجموعتان أول مجموعة قامت بتعريف 100 متغير و 100 دالة لهذا الغرض فكتبت في كل كائن دالة للبحث في خاصية معينة فمثلاً دالة للبحث حسب الإسم ودالة أخرى للبحث الرقم ودالة أخرى للبحث الرقم و الإسم معاً وهنا تكون المجموعة الأولى قد أطالت من برنامجها وقللت من كفاءة البرنامج التي صنعته . وتأتي المجموعة الثانية فتستخدم 30 متغير و 30 دالة وتؤدي نفس الغرض فهي هنا استخدمت دالة خاصة بالبحث في كل كائن بدلاً من عدة دوال وهي هنا لم تحدد البحث بواسطة ماذا ولكنها عند تشغيل البرنامج تقوم وبناءاً على طلب المستخدم بالبحث عن أي خاصية يريد .

وكمثال ثان :

نفترض أن عندنا شركة مشروبات غازية تقوم بإضافة علبه خامه ((قياسية)) فتأتي شركة أ وتطلب منه عبوة هنا تستخدم الشركة العبوة الخامه وتقوم بالتعديل وإضافة الإعلانات الخاصة بشركة أ على العلب المطلوبة وتأتي شركة ب وتطلب 300 عبوة فتصنع كما صنعت مع شركة أ . ولنتوقف عند هذا المثال قليلاً ونسأل أنفسنا : ماذا لو قمنا بالأصل بتصميم العلب للشركة أ وقامت الشركة ب بالطلب ؟؟ ستحدث خسارة في الوقت والجهد ، فهنا يكون معنى الـ Abstraction قد توضح لنا ورسم في مخيلتنا .

الشرط الثالث : Inheritance

لو سألنا أنفسنا ما معنى هذه الكلمة لأجبنا أنفسنا بالسرعة معناها التوارث ((طيب)) وماذا يعني ذلك ؟؟

العديد من المبرمجين يفهموا هذا المفهوم فهماً خاطئاً كالتالي : إذا كان عندنا أربع أو خمس كائنات تحتوي جميعها على خاصية موحدة أو خاصيتان أو عدة خواص فإننا نقوم بإنشاء كائن جديد نضع فيه الخواص المتشابهة في كل كائن ونقول لأنفسنا أن جميع الكائنات الأخرى هي عبارة عن Inheritance أي متوارثة من الكائن الأصلي ((الجديد)) وهذا بمفهومه المنطقي صحيح ولا غبار عليه . ولكن النظرة من الجانب العملي لها وجهة خاصة فعندما نقوم بتصنيف الخواص المشتركة ووضعها في كائن جديد لم يعد للكائن الجديد معنى . وإليك المثال التالي لتسهيل الفهم :

لنفترض أننا نمتلك الكائنات التالية : الطالب و الدكتور والعامل والسكرتير و ... الخ كل الكائنات السابقة لها خاصيتان متشابهتان وهما الاسم والرقم . فلو قمنا بفصل هذين الخاصيتين وجعلناها في كائن واحد (إنسان) ثم قمنا بإعطائه دور الأب وإعطاءه دور الابن ، فهنا يمكننا القول بأن الكائنات الأخرى هي Inheritance أي متوارثة من الكائن الأصلي ((الإنسان)) وهذا كما قلت منطقياً مقبول . ولكن !!! ماذا لو أخذنا الكائن الجديد على حدة وسألنا أنفسنا ماذا يمثل ؟؟؟ لا شيء وليس له معنى لأنه عندما كانت هذين الخاصيتين في كائن الطالب كانت هناك تفاصيل أكثر دقة تفسر الكائن وتدل على أنه كائن الطالب . وبعد أخذهما على حدة فلا يمكننا التمييز ما هو ؟؟؟ .

سؤال يجب طرحه هنا : ما الفرق بين الـ ((Class)) الكائن وبين الـ ((Object)) ؟؟؟

سؤال محير أليس كذلك !! إليك الفرق :

الـ Class هو شيء نظري غير موجود على الواقع فهو عبارة عن تعريفات لشيء معين نظرياً . الـ Object وهو الكائن النظري عند تطبيقه على أرض الواقع ليصبح عنصراً وكائناً عملياً .

الشرط الرابع : Polymorphizen

يمكن تقسيمه إلى نوعين وهما الـ Overload والـ Override . وإليك تبييناً لكلاهما : Overload : وهي عبارة عن دوال لهم نفس الاسم ولكن تختلف في عدد البارامتر ((الوسائط التي تأخذها)) أو نوعها

Operator Overload : فمثلاً نريد إشارة الزائد + في المعادلة التالية $2=1+1$ أن نحوله إلى حرف وصل ((&)) فيمكننا تعريف نفس الاسم ولكن أغير في النوع والكود لتصبح المعادلة التالية $1 \& 1 = 11$.

Override : وهي عبارة عن كتابة نفس الدالة الأصلية طبق الأصل عنها ولكن أغير في الكود فمثلاً نعرف الجملة التالية $X = 3$ ثم نقوم بعدها بتعريف الجملة التالية $X = 7$ فيقوم الحاسوب بحذف الأولى وتطبيق الثانية . هذا بالنسبة للمتغيرات أما بالنسبة للدوال فإذا عرفنا دالة في كائن معين ثم قمنا بإعادة كتابة الدالة ولكن غيرنا في الكود فيقوم الحاسوب بحذف الأولى و اعتماد الثانية .

إذن يحق لي كمبرمج إستعمال الدالة Function أكثر من مرة بإضافة تغييرات في النوع أو البارامتر أو بإعادة تعريفه .

إذن :

Overload : تغيير النوع أو عدد البارامتر لدالة معينة .

Override : تغيير الكود فقط ولا تغيير على النوع أو البارامتر .