

بسم الله الرحمن الرحيم

**Community College of Applied Science
and Technology**

Search on

Object Oriented Programming.

Student

Fota.Khader.Far

::::: Specialization :::::

Programming and Data Base.

Number

120061156.

Teacher

Rami Al Robi.

The contents of the search

- 1) java,C++,C# تاريخ البرمجة الشيئية في
- 2) Encapsulation الكبسلة
- 3) Inheritance الوراثة
- 4) Classes الفصائل
- 5) Operator Overloading

تاريخ البرمجة الشينية في java and C++.

سنتكلم هنا على انشاء هذه اللغات و دخولها الى عالم البرمجة الكائنية ،،،،

اولاً

سوف نتكلم على الجافا والس بلس بلس و كيفية دخولهما الى عالم الكائنية :....

الجافا : هي لغة برمجية اخترعت عام ١٩٩٥ على يد James Gosling و Henry McGilton (أسكتلندي) و فريق عمل مكون من الاف الأشخاص في شركة Sun سميت بالجافا بعد اجتماع James Gosling مع مجموعه من فريق العمل في أحد المقاهي فتوارد الى أذهانهم أسم Java وهي القهوة المشهورة فسميت اللغة بهذا الأسم ولو لاحظت شعار جافا لوجدت أنه يحمل شعار فنجان القهوة.

لماذا تم اختراع لغة جديدة الا تكفي اللغات الموجودة للبرمجة؟
كان الهدف عند اختراع جافا هو لغة قادرة على برمجة نظم لتشغيل جميع الأجهزة من كمبيوترات عملاقة (mainframes) الى الأجهزة الصغيرة مثل مشغلات MP3
لماذا الجافا مهمة؟

شركة صن Sun يصفون الجافا كالتالي : لغة بسيطة (Simple) و شينية (Object-Oriented) وموزعة (Distributed) و مفسرة (Interpreted) و قوية (Robust) و محمية (Secure) و ذات بنية متعادلة (Architecture Neutral) وقابلة للتغيير (Portable) و عالية الأداء (High-Performance) و متعددة الوظائف (Multithreaded) و لغة ديناميكية (Dynamic Language)

سنمر بالتفصيل على كل كلمة!

أولاً: لغة بسيطة (Simple)

جافا لغة بسيطة بالفعل فمصممين اللغة حاولوا جعلها لغة يتعلمها المبرمج بسرعة فلو كنت مبرمج لغة ++C/C ستجد أن معظم الأوامر متشابهة مصموا اللغة أزالوا بعض من الخصائص الموجودة في ++C/C عن لغة جافا لجعلها صغيرة و بسيطة فعلى سبيل المثال جافا لا تدعم goto و وضعوا بديل لها break و continue يمكن يكون أكبر تبسيط في اللغة هو أنها لا تحتوي على مؤشرات Pointer لأن أغلب الأخطاء في البرمجة بالمؤشرات فجافا اوتماتيكية تعامل المؤشرات بطريقتها الخاصة فتصنف الكائن و تلغي المؤشر الغير موجود.

شينية (Object-Oriented)

اللغة صممت أساساً على أنه تكون شينية على هذا الأساس أنت كمبرمج يجب عليك التركيز على البيانات و الطرق (Methods) (ال functions و ال procedure تسمى في جافا methods) في اللغات الشينية ال class مجموعة من البيانات و الطرق التي تعمل على هذه البيانات، البيانات و الطرق تعرف حالة الكائن (Object)

جافا أتت بمجموعة من ال Classes مرتبة في packages تستطيع استخدامه في برامجك مثل java.awt مجموعة من ال classes لتكوين النوافذ لبرنامج (graphical user interface) معظم الأشياء في جافا ممثلة ب classes ماعدى ال Character و Boolean و Numeric فال String عبارة عن Class

موزعة (Distributed)

عندما نقول لغة موزعه فمعناها أنها تقدم تدعم للشبكات جيد و مما أذهلني في الجافا قدرتها على تقديم دعم عالي جداً للشبكات فعلى سبيل المثال URL عبارة عن Class هو و مجموعة كلاسات في رزمة java.net تسهل عليك الأمر جداً فقراءة ملف في جهاز آخر في الشبكة كقراءة ملف في جهازك المحلي

تكن قوة جافا في هذه النقطة في (Remote Method Invocation) هو عبارة عن API لتشغيل الطرق من كائن في الشبكة كما لو أنها في جهازك العملية نوعاً ما معقدة لكن سنحاول شرحها لاحقاً
مفسرة (Interpreted)

عندما نعمل compile لبرنامج جافا فإنه يعطينا byte-code يقرئه ال Java Virtual Machine لغات البرمجة الأخرى تعطيك Machine code أي Assembly ببساطة تلاحظ أنه عندما تريد أن تشغل برامج جافا يتطلب أن يكون في جهازك مشغل الجافا JVM طبعاً هذا له فوائد كثيرة بنستعرضها في الفقرة Portable

قوية (Robust)
عندما نطلق على برنامج أنه Robust فهذا معناه أنه لا يفشل عند محاولة تشغيله لكن عندما نطلق الشيء نفسه على لغة فمعناته أن اللغة تحد من وجود الأخطاء في البرامج باتخاذ الإجراءات اللازمة لجعل البرمجة بهذه اللغة لا يمكن أن تنتج برامج تحتوي أخطاء نعم ما زال ممكن أن يكون هناك أخطاء في برامج الجافا ولكنها محدودة جداً فمثلاً عندما قلنا في البداية أن جافا لا تحتوي مؤشرات فهو للحد من الأخطاء البرمجية.

محمية (Secure)
بلا شك أن جميعنا يعلم أن ال applets مبرمجة بالجافا و تحتاج الى وجود ال JVM لتشغيل في اليوم الواحد تمر بمواقع عدة بعضها يحتوي على applets فمن الممكن أن يكون هذا البرنامج برنامج خبيث ليضر بجهازك عند تصميم الجافا أخذ في الحسبان الحماية و وضع لها عدة طبقات لحماية الجهاز من البرامج الخبيثة و الأكواد الغير صحيحة فعلى سبيل المثال لا يمكنك أن تقرأ من الذاكرة مباشرة. ذات بنيان متعادل (Architecture Neutral) وقابلة للتغيير (Portable)
أجمل ميزة في الجافا أنها تشتغل على جميع الأجهزة لا تأخذ في الحسبان أي نظام تشغيل ذلك الجهاز يشغل كل ما عليك هو كتابة برنامج و بدون أي تغيير في البرنامج بإمكانه العمل على وندوز أو لينكس أو ماكنتوش!!! كيف يتم ذلك

كل هذا بفضل ال Byte-Code و JVM فجافا لا تحول البرنامج الى لغة الآلة إنما تحوله الى ما يسمى Byte-Code و لكي يشتغل البرنامج فيجب توفر JVM في الجهاز الذي بدوره يقرأ ال Byte-Code و يحوله الى ما يتناسب مع نظام التشغيل. مقولة مشهوره (Write Once, Run Anywhere) أكتب مره واحده وشغل في أي مكان مختصه بالجافا.

عالية الأداء (High-Performance)
إذا جينا نقارن سرعة الجافا مع C نلاحظ أن جافا أبطء بعشرين مره تقريباً لأنه تحول الى Byte-Code ثم الى لغة الآلة بينما C الى لغة الآلة مباشرة. لكن لو قارنا ما تقدمه من أداء بالنسبة الى اللغات الأخرى فنجد أنها في الوسط فمثلاً TCL لغة Portable و جيدة لعمل البرامج التمثيلية Prototyping و لكنها بطيئة جداً ال C سريعة و لكنها تفتقد لل Reliability و Portability

(سهولة التغير و الاعتمادية) بالنسبة لجافا فهي في الوسط فهي Reliability و Portability كما أنها ليست ببطء TCL أو UNIX shells

متعددة الوظائف (Multithreaded)

لو كان لديك عدة مهمات تريد عملها في نفس الوقت فأن جافا تتيح لك ذلك من خلال قدرتها على تشغيل أكثر من مهمة في نفس الوقت. كما أنها تتيح لك بكلمة **synchronized** التحكم بجزئية من البرنامج بحيث يعمل في وقت محدد.

لغة ديناميكية (Dynamic Language)
أي كلاس **class** في الجافا بإمكانك تحميله للمترجم في أي وقت.

*****الجافا و السي بلس بلس*****

****تاريخ البرمجة الشيئية الكائنية في java C++ C
java**

اللغة صممت أساساً على أنه تكون شيئية على هذا الأساس أنت كمبرمج يجب عليك التركيز على البيانات والطرق (Methods) (ال **functions** و ال **procedure** تسمى في جافا **methods**) في اللغات الشيئية ال **class** مجموعة من البيانات و الطرق التي تعمل على هذه البيانات، البيانات و الطرق تعرف حالة الكائن (Object)

جافا أتت بمجموعه من ال **Classes** مرتبة في **packages** تستطيع استخدامه في برامجك مثل **java.awt** مجموعة

من ال **classes** لتكوين النوافذ لبرنامج (graphical user interface)

C++

المهم هذا جعل شخصا جميل جدا اسمه **Bjarne Stroustrup** في عام ١٩٨٨ ان يقوم باضافة اذكي ما توصل اليه العقل البشري حتي الان في هندسة تصميم البرامج انها البرمجة الموجة لكائن **Object Oriented programming** او **OOP** وكانت لغة **C++** تخيل لغة بقوة السي مع اضافة ال **OOP** لها انها ثورة اخري في لغات البرمجة بعد ثورة السي ومضي الوقت وتربعت السي بلس بلس علي عرش لغات البرمجة بدون منازع حتي عام ١٩٩٥ عندما ظهرت الجافا كانت الجافا هي ثورة اخري بعد **C++** فهي نسخة من السي بلس بلس مع التخلص من عيوبها. هل للغة **C++** عيوب ؟

C#

#C هي تطور حقيقي للجافا فالـ **Net FrameWork** هي تقنية المستقبل وهي اقوي بكثير من تقنية الجافا وذلك نتيجة تجربتي للتقنيتين وللعلم لكل محبي البرمجة كل شيء مع **#C** مجاني هذا شيء في منتهي الروعة فالـ **IDE** مجانية بعيدا عن الـ **Microsoft Visual Studio** هناك الـ **SharpDevelop** ولكن لابد ان يكون لديك الاول الـ **net FrameWork** ويمكنك تنزيلها مجانا من علي موقع ميكروسوفت كما يمكنك تنزيل الـ **SDK** ايضا مجانا.

::سي بلس بلس و السي شارب::

تركيب لغة السي الا انها مع مرور الوقت وكبر حجم البرامج أصبحت لغة السي صعبة التحكم فضخامة حجم البرامج مع وجود كيان واحد فقط مستقل يمكننا تقسيم البرنامج من خلاله وهو الدالة جعل البرامج الضخمة صعبة التحكم جدا فتخيل كثرة الدوال وهي اسفل بعضها مع اضطرارك للتصريح عنها اولا قبل الدالة (main) ثم تعريفك لها بعد جسد الدالة main ثم استدعائك لها داخل الدالة main لو كان البرنامج صغيرا فليس

هناك مشكلة ولكن اثناء كتابتك للبرنامج لا يمكنك ان تحدد كم سطرا ستكتب فانا علي سبيل المثال عندما بدأت احد مشروعاتي البرمجية بلغة C# لم اقم من امام الجهاز الا وانا كاتب اكثر من الف سطر من الاوامر البرمجية بلغة السي شارب مع الاخذ في الاعتبار ان هناك حوالي الفين سطر كتبهم المترجم بدلا مني وهم الاوامر الخاصة بتصميم الواجهة الرسومية اي ان حجم البرنامج الصغير هذا اكثر من ثلاثة الالاف كود تخيل لو كنت كتبت هذا البرنامج بلغة السي لكن التحكم فيه من الصعوبة بمكان مع كتابتي شخصا للثلاثة الالاف كود!!!

المهم هذا جعل شخصا جميل جدا اسمه Bjarne Stroustrup في عام ١٩٨٨ ان يقوم باضافة اذكي ما توصل اليه العقل البشري حتي الان في هندسة تصميم البرامج انها البرمجة الموجة لكانن Object Oriented programming او OOP وكانت لغة ++C تخيل لغة بقوة السي مع اضافة الـ OOP لها انها ثورة اخري في لغات البرمجة بعد ثورة السي ومضي الوقت وتربعت السي بلس بلس علي عرش لغات البرمجة بدون منازع حتي عام ١٩٩٥ عندما ظهرت الجافا كانت الجافا هي ثورة اخري بعد ++C فهي نسخة من السي بلس بلس مع التخلص من عيوبها. هل للغة ++C عيوب ؟ -نعم لها عيوب مثلثة...

اولا: اكثر ما يصيب بالاحباط فيها انها ليست Full OOP ماذا يعني هذا؟ يعني انك عندما تبدأ كتابة البرامج بلغة السي بلس بلس لا تدري هل هي تدعم الـ OOP ام لا فربما تظنها السي!!!

هذا شيء يدعو للأسف فعلا فانت عندما تكتب اول برنامج بلغة السي بلس بلس تستخدم الـ OOP دون ان تدري تخيل!!

فانت تستخدم cingout ولكن لا تعرف شيءا عنهما . ليس هذا فقط فلغة ++C تفرض عليك التحكم المباشر في الذاكرة.

وهل هذا شيء انها سمة من سمات القوة؟!

-نعم هي سمة من سمات القوة والصعوبة في نفس الوقت.

المهم جاءت الجافا بعكس ذلك تماما . فتخلصت من التحكم المباشر او ما تسمى الادارة اليدوية للذاكرة وجعلت المترجم الخاص بها يقوم بذلك بدلا منك وعندما تدخل علي الجافا ستعرف فورا انها لغة تدعم الـ OOP بل لا تدعمها فقط بل هي الاساس والباقي فروع فانت لا يمكنك كتابة امر واحد بالجافا دون ان تستخدم الـ oop .

دعنا نتكلم عن السي شارب ...

اذا كنت ميرمج بالجافا فانت تعرف الجافا جيدا ولا تحتاج مني للكلام الكثير عنها اما اذا كنت لا تعرف الجافا فيمكنني بكل ثقة ان اقول لك ان C# هي تطور حقيقي للجافا فـ<Net FrameWork هي تقنية المستقبل وهي اقوي بكثير من تقنية الجافا وذلك نتيجة تجربتي للتقنيتين وللعلم لكل محبي البرمجة كل شيء مع C# مجاني هذا شيء في منتهى الروعة فالمترجم او الـ IDE مجانية بعيدا عن الـ Microsoft Visual Studio هناك SharpDevelop يمكنك تنزيل بيئة التطوير هذه من الموقع

التالي WWW.sharpdevelop.net ولكن لابد ان يكون لديك الاول <net Framework ويمكنك تنزيلها مجانا من علي موقع ميكروسوفت كما يمكنك تنزيل الـ SDK ايضا مجانا...

الى هنا ينتهي حديثي عن تاريخ اهم اللغات شهرة و عالمية وبدايتها

وكيفية ظهور البرمجة الشيئية فيها #C,C++,java

.....

.....:الكبسلة:.....

سنعرف ما هي الكبسلة و ماذا نقصد به..

كبسلة Encapsulation

لكي نفهم فكرة الكبسلة ممكن ان نأخذ مثال وليكن صندوق جهاز الحاسوب box والذي يتكون من اللوحة الأم motherboard.. والتي بدورها تتكون من دوائر وما الى ذلك .. ويتكون الجهاز ايضا من Power والذي هو وحده مكونه من أجزاء داخلية .. ومن ثم هناك معالج والذي عبارة عن جزء أو وحدة مكون من عدة أجزاء .. وبالنسبة سلاحظ ان جهاز الحاسوب مكون من تجمع أجزاء وهذه الاجزاء كل منها مكون من أجزاء ووحدات داخلية .. ومن المثال نستطيع ان نقول ان الكبسلة خاصية ان اي كائن يتكون من وحدات داخلية تتكامل مع بعضها لتكوين الكائن الأكبر ولاحظ انك من مبدأ الكبسلة تستطيع ان تحقق مبدأ اخفاء البيانات

اخفاء البيانات Data Hiding

وهي خاصية كان نقول مثلا انك تستطيع استخدام أي جهاز جوال دون الحاجة لمعرفة طريقة الصنع والتفاصيل الفنية لجهاز الجوال الذي تستخدمه .. فقط يكفيك انك تستطيع استخدامه والتعامل معه.

الكائن يجمع في مضمونه بياناته (خصائصه)، والكود القادر على معالجتها (المناهج).

تجريد البيانات: هيكل كائن ما لا يظهر من الخارج، شكله يتركب من رسائل غير اصطلاحية، واستقبال لأي رسالة ينجم عنه تنفيذ مجموعة من المناهج.

تجريد الإجراءات: إذا ما نظرنا من الخارج (بالنسبة لمستخدم الكائن)، المستخدم لا يملك أدنى معلومة حول النشاط الداخلي المطبق، فمثلا: هو لا يدري ما إذا كان العمل المطلوب، يحتاج إلى تنفيذ بعض أو كل المناهج أم لا، أو سيقوم بخلق كائن مؤقت... الخ.

حسب القيم القانونية النموذجية للكائن، خدمات الكائن ليست إصطلاحية (لا يمكن تحقيقها) إلا من خلال رسائل، والتي هي مركبة من:

إسم
قائمة بارامترات الدخول
قائمة بارامترات الخروج

قائمة الرسائل التي يقدر بفضلها الكائن من الاستجابة تمثل واجهته interface، إنها القسم العام public من الكائن. أما كل ما يخص المعالجة، فيجب أن يبقى مخفيا عن المستخدم النهائي: وهذا ما يمثل القسم الخاص private من الكائن، تماما، كل الكائنات المنتمية لنفس الصنف تتمتع بذات الواجهة، بالمقابل كائنين ينتميان لصنفين مختلفين، يقدران على تقديم نفس الواجهة. من وجهات نظر مختلفة، الواجهة يمكن تمثيلها على شكل خاصية متميزة للصنف.

تطبيقيا، في معظم لغات البرمجة الحديثة كائنية التوجه oop، الواجهة تمثل قائمة المناهج التي يمكن النفاذ إليها من قبل المستخدم.

من المستحسن إخفاء تفاصيل معالجة الكائنات عن أعين المستخدم، بحيث يسمح ذلك بتغيير مثلا الهيكل الداخلي لبيانات الصنف (تبديل جدول بحزمة رموز) من دون اللجوء إلى تعديل كود المستخدم. أو كمثال آخر: لنعتبر أن صنفا ما يشكل لنا نقطة بإحداثيتين (بعدين)، والتي تعطينا منهاجين يحددان على التوالي موقع النقطة من محوري الفواصل والتراتب (السطر والعمود)، وبالتالي ليس من الداعي إعلام المستخدم ما إذا كانت النقطة ممثلة داخل الصنف على شكل قطبي أو خطي.

كل اللغات الكائنية التوجه لا تشترط الالتزام بمبدأ الكبسلة (الإخفاء)، إذن هو من خصوصيات وطريقة تفكير المبرمج.

الى هنا ينتهي مفهوم الكبسلة لدينا و نعرف ان الكبسلة هي من اختصاص المبرمج....

.....

..... Inheritance الوراثة

تعتمد فكرتها على امكانيه اعاده استخدام كود البرمجة لبناء اجزاء جديدة من اكواد سابقة دون الحاجة لان تعيد الكتابة من الصفر وبالطبع الكود الجديد سيحتوي تفاصيل الكود القديم ..
وكمثال لذلك .. اي اصدار لبرنامج حاسوب فانه يكون يحتوي على تفاصيل مشابهة للاصدار السابق لان مبرمج البرنامج لن يعيد برمجة البرنامج انما سيطور عليه مع زيادة تفاصيل ..
ويمكننا ان نأخذ مثال اكثر قربا وهو السيارة فالسيارة لموديل هذا العام لن تختلف عن موديل العام السابق بالخصائص انما ستزيد الشركة على الموديل السنه الماضيه المزيد من التفاصيل الجديدة والمميزات فبالطبع لن تعيد الشركة المصنعه التصميم وتنفيذ أجزاء السيارة بالكامل ولكنها ستقوم بالتعديل والتطوير على الموديلات السابقة واخرجه بشكل جديد وبالتالي فان الوراثة ستوفر الكثير من الوقت والجهد ..

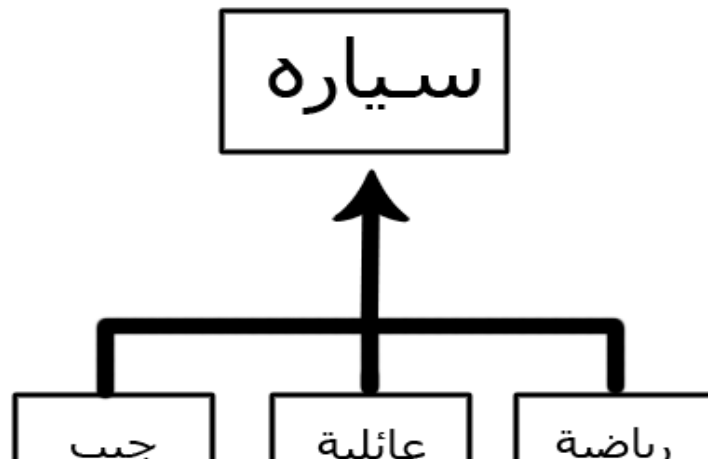
الوراثة بين مجموعة فئات

الفئة:

الفئة هي مجموعة من الكائنات المتشابهة .

الوراثة

تختلف الوراثة بين الفئات عن الوراثة بين الكائنات بشكل جذري فلا يمكننا القول بأن الابن هو أبيه وأيضاً لا يملك الابن جميع صفات أبيه في حين أنه يمكننا القول بأن "الحمضيات" هي "أشجار مثمرة" إذ أن فئة الحمضيات تمتلك جميع صفات فئة الأشجار المثمرة وتزيد عليها بعض الصفات الإضافية فعند تعريف فئة "مستطيلات" في أحد برامج الرسم فإنها تزود بطريقة لرسم المستطيل وإذا كان رسم المربعات يتم بشكل متكرر فربما تعرف الفئة "مربعات" على أنها فئة ابن للفئة "مستطيلات" تتمتع بصفة إضافية هي تساوي الطول والعرض وبمجرد حدوث ذلك فإن هذه الفئة ستتمكن من رسم المربعات دون أن تحتوي على إجراءات خاصة بها لرسم المربع إذ أنها ستسأل عن طول ضلع للمربع ومن ثم ترسم المربع على اعتباره مستطيل تساوي طوله مع عرضه.



عملية الوراثة سهله، فانت على سبيل المثال، ورثت صفة الحواجب العريضه من الاب، و الانف المعتدل من الام وغير ذلك، ولكن لديك اشياء وخواص خاصه بك لا توجد في الاب و الام اليس ذلك صحيحا ؟ نفس الشيء هنا، لنضرب مثالا على ذلك ، لدينا شركة سيارات، تقوم ببيع سيارات على عدة انواع (رياضيه ، عائليه ، جيب) ، كل منهم لديهم خواص متماثله ، كل سياره لها لون، كل سياره لها عجلات، وغيرها، ولكن كل نوع لديه بعض الخصائص لا توجد في الاخرى، الرياضيه مقعدان فقط، الجيب ٧ مقاعد ، العائليه ٥ مقاعد. في مبدا الكائنات نقوم بعملها كالتالي، ننشئ تصنيف اسمه سياره، وثلاث تصنيفات اخرى ونجعلها تورث الصفات من التصنيف سياره.

انشاء التصنيفات (Class) ، الخواص (Attribute) ، العمليات (Operations) :

تكلنا كثيرا في درس اليوم، لنأتي الان الى تطبيقاتنا ، ونأخذ امثله حيه لكيفة انشاء التصنيفات والخواص والمعاملات. يمكننا انشاء تصنيفات باستخدام العبارة class لاحظ المثال التالي:

```
1 <?php
2
3 class car
4 {
5
6 }
7
8 ?>
9
```

هذا مثال على php

تصنيفنا هذا يحتاج الى خواص ، لنقم الان بإنشاء خاصيتين ، وهي اسم السياره ولونها ، ونقوم بإنشاء الخواص باستخدام العبارة var ، لاحظ التالي :

```
1 <?php
2
3 class car
4 {
5     var $carName;
6     var $carColor;
7 }
8
9 ?>
```

php

.....class الكلاس:.....

العلاقة بين Classes:

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 بتصفية 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 ... إلخ
كل الكائنات السابقة لها خاصيتان متشابهتان وهما الاسم والرقم .
فلو قمنا بفصل هذين الخاصيتين وجعلناها في كائن واحد ثم قمنا بإعطاءه دور الأب وإعطاء الكائنات التالية دور الإبن فهنا يمكننا القول بأن الكائنات الأخرى هي Inheritance أي متوارثة من الكائن الأصلي ((الجديد)) 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 حده فلا يمكننا التمييز ما هو ؟؟ . !!

classes and objects:

You may find it not easy to understand the class and object story but I will try to do my best explaining it. Actually the class and object concept is related to each other and some beginners don't care about understanding it clear so I think they will have a hard times learning C.#

Object-Oriented concept takes the most of its functionality from the real-life concepts. For example, I will discuss the concept of Classes and Objects of the world first and then you will understand the computer's Classes and Objects before I even write anything about it.

مثال عام على ال class

```
class Person
{
    public int Age;
    public string HairColor;
}
```

This is our simple class which contains 2 variables. Don't worry about public keyword now because we will talk about it later . Now we will instantiate this class (that is, When you create an object of this class).

```
static void Main(string[] args)
{
    Person Michael = new Person();
    Person Mary = new Person();

    // Specify some values for the instance variables
    Michael.Age = 20;
    Michael.HairColor = "Brown";
    Mary.Age = 25;
    Mary.HairColor = "Black";
    // print the console's screen some of the variable's values
    Console.WriteLine("Michael's age = {0}, and Mary's age = {1}", Michael.Age,
        Mary.Age);
    Console.ReadLine();
}
```

.....Operator Overloading.....

Operator overloading allows programmers to build types which feel as natural to use as simple types (int, long, etc.). C# implements a stricter version of operator overloading in C++, but it allows classes such as the quintessential example of operator-overloading, the complex number class, to work well.

In C#, the == operator is a non-virtual (operators can't be virtual) method of the object class which compares by reference. When you build a class, you may define your own == operator. If you are using your class with collections, then you should implement the IComparable interface. This interface has one method to implement, called the CompareTo (object) method, which should return positive, negative, or 0 if \"this\" is greater, less than, or the same value as the object. You may choose to define <, <=, >=, > methods if you want users of your class to have a nicer syntax. The numeric types (int, long, etc) implement the IComparable interface.

Here's a simple example of how to deal with equality and comparisons:

```
public class Score : IComparable
```

```
{  
    int value;
```

Operator Overloading مثال على

```
    public Score (int score) {  
        value = score;  
    }  
}
```

```
    public static bool operator == (Score x, Score y) {  
        return x.value == y.value;  
    }  
}
```

```
    public static bool operator != (Score x, Score y) {  
        return x.value != y.value;  
    }  
}
```

```
    public int CompareTo (object o) {  
        return value - ((Score)o).value;  
    }  
}
```

```
Score a = new Score (5);
```

```
Score b = new Score (5);
Object c = a;
Object d = b;
To compare a and b by reference:
```

```
System.Console.WriteLine ((object)a == (object)b; // false
To compare a and b by value:
```

```
System.Console.WriteLine (a == b); // true
To compare c and d by reference:
```

```
System.Console.WriteLine (c == d); // false
To compare c and d by value:
```

```
System.Console.WriteLine (((IComparable)c).CompareTo (d) == 0); // true You could also
add the <, <=, >=, > operators to the score class.
C# ensures at compile time that operators which are logically paired (!= and
==, > and <, >= and <=), must both be defined
```

مثال آخر على

Operator Overloading

يقوم هذا البرنامج بأخذ الرقمين و القسمة على نفسها و اعطاء القيمة النهائية لهم

```
g System;
```

```
public class Fraction
{
    private int d; //البسط
    private int n; // = 1; //المقام

    public Fraction()
    {
        N = 1;
    }

    public int D
    {
        set { d = value; }
        get { return d; }
    }

    public int N
    {
        set
        {
            if (value == 0)
            {
                n = 1;
            }
        }
    }

    Console.WriteLine("Numerator Can't Zero!!!");
}
```

```

        }
        else
            n = value;
    }
    get { return n; }
}

public void Print()
{
    Console.WriteLine(D + "/" + N);
}

public void Read()
{
    D = int.Parse(Console.ReadLine());
    N = int.Parse(Console.ReadLine());
}

static public Fraction operator*(Fraction f1, Fraction f2)
{
    Fraction result = new Fraction();

    result.D = f1.D * f2.D;
    result.N = f1.N * f2.N;

    return result;
}

static public Fraction operator+(Fraction f1, Fraction f2)
{
    Fraction result = new Fraction();

    //(d1 * n2)+(d2 * n1)
    result.D = (f1.D * f2.N) + (f2.D * f1.N);

    //n1 * n2
    result.N = f1.N * f2.N;

    return result;
}

static public Fraction operator-(Fraction f1, Fraction f2)
{
    Fraction result = new Fraction();

    result.D = (f1.D * f2.N) - (f2.D * f1.N);
    result.N = f1.N * f2.N;

    return result;
}
}

```

```

using System;           وهذه الكلاسس الاخر للتشغيل

public class Program
{
    static void Main(string[] args)
    {
        Fraction f1 = new Fraction(), f2 = new Fraction();
        Fraction f3;

        f1.Read();
        f1.Print();

        f2.Read();
        f2.Print();

        f3 = f1 * f2;
        f3.Print();

        f3 = f1 + f2;
        f3.Print();
    }
}

```

::جدول رائع يوجد فيه اغلب المفردات البرمجية و خاصة الكائنية::

المصطلح الإنجليزي	المترادف العربي	الشرح	Method	منهاج
Object	كائن	هو مجموعة من الدوال والبيانات التي تعمل في إطار موحد مشكلة شيئا متناسقا.	Attributes	هو دالة من الدوال التابعة لكائن ما، ومن بينها المشيد والمهم.
Instance	مثيل	هو صورة منفصلة من الصور التي تم تشكيلها من صنف ما.	Constructor	مختلف البيانات التابعة لكائن ما، سواء المحمية أو العامة أو الخاصة.
Instanciation	خلق مثيل	عملية تشكيل كائن أو كائنات عديدة من نفس الصنف.	Destructor	هو المنهاج الذي يقوم بهيئة الخصائص والبيانات التابعة للكائن عند أول عملية لخلق الكائن.
Heritage	ورثة	عملية اشتقاق أصناف جديدة من أصناف أكثر عمومية لتحتوي بذلك على مميزاتها، مع القدرة على إضافة مميزات جديدة وعلاقتها هي: الصنف المشتق هو إصدار خاص من الصنف القاعدي	Overload	هو المنهاج الذي يقوم بتعطيم الكائن بعد نهاية العمل، لتحرير الذاكرة المحجوزة.
Encapsulation	كبسلة	إخفاء بعض البيانات عن المستخدم لحماية الكائن من تعديلات غير مرغوبة.	Interface	إجراء تصنيف للكائنات المشابهة (في أي صفة) لاستخلاص عدة قوالب جامعة ومتسلسلة.
Aggregation	تركيب	هو أسلوب يسمح بتحديد مركبات كل صنف (أو كائن) وعددها، أو بمعنى آخر، العلاقة بين الأصناف، لتركيب صنف جامع.	Modeling	هو المنهاج الذي لا يمتلك مشقات
Polymorphism	تعدد الصفات	هو أسلوب يسمح لصنف ما من أن يتخذ عدة أشكال، ومع نفس الأسماء يمكن طلب مناهج مختلفة تحمل نفس الاسم، ولكن تختلف في تركيبها.	Page	علاقة تربط بين الأصناف ولا تكون محددة كالتركيب
Class	صنف	هو قالب يتم من خلاله خلق كائنات متشابهة أو مختلفة.	Association	الشرارة
Method	منهاج	هو دالة من الدوال التابعة لكائن ما، ومن بينها المشيد		

بهذا انتهى من هذا البحث البرمجي

References

- 1) www.csharp1.com
- 2) www.microsoft.com/msdn
- 3) www.csharphelp.com
- 4) Book (البرمجة الشيئية / ترجمة تواتي عمر)
- 5) A Programmer's Introduction to C#