الفصل السابع عشر
التصميم الجيد لواجهة المستخدم

سنتعرف في هذا الفصل على فلسفة "مركزية المستخدم"وكيف ننشئ واجهة تطبيق مبنية عليها. سنتعرف أيضا على المراحل التي يمر بها المنتج البرمجي. كذلك سنتعرض لبعض العناصر الأساسية التي تستخدم لبناء واجهة جيدة للمستخدم ومنها النماذج والقوائم وأدوات التحكم.و أخيرا سنتعرض لطريقة كتابة الكود وأهميتها في أغراض صيانةوتعديل الكود لاحقا.

بانتهاء هذا الفصل سنتعرف علي :

- *إنشاء البرنامج الذي يعمل كما يريد المستخدم.*
- *كيف تكون واجهة المستخدم بديهية واضحة.*
- *التصميم الجيد للنماذج والقوائم.*
- *بناء الانطباع الجيد عند المستخدم عن البرنامج.*
- *كيف تتجنب الأخطاء الشهيرة في كتابة كود البرنامج.*

قبل ظهور Visual Basic كانت برمجة Windows أمرا ليس بالهين ،كان عليك أن تبدأ بقراءة العديد من الصفحات التي تشرح طريقة عملها، ثم تبدأ في تعلم الأنماطوالدوال اللازمة لكتابة برنامج يعمل تحت بيئة Windrows بلغة C مثلا ، ثم يلي ذلك البرمجة الفعليةوالتي تمتلئ بالتكراروالاحتمالات الأخطاء، حيث كان برنامج بسيط لا يفعل إلا إظهار نافذة خالية يحتاج إلى قدر كبير من المجهود والكتابة والتصحيح.

وعندما ظهر Visual Basic أعفى المبرمج من كل هذه التفاصيل وأصبح عليه أن يركز فقط في وظيفة برنامجه، لا في وظيفة Windows،ولذا فإن تسهيل البرمجة رغم أنه رفع عن كاهل المبرمج كثير من العناء الضائع المكرر، إلا أنه كذلك يلقي على عاتقه الكثير من المسؤولية تجاه وظيفة برنامجهوسهولة استخدامهوتناسق واجهته.

إن البرامج الآن تبنى على فلسفة جديدة هي (مركزية المستخدم): اقتباسا من مركزية الشمس،وليست كالماضي على مركزية البرنامج، إننا ننتج البرنامج لنفيد المستخدمونسهل عليهونرفع إنتاجيته انطلاقا من خبرته السابقة في التعامل مع الكمبيوتروبرامجه، لا لنرهقه بتعلم كيفية استخدام برنامجناونقوم بغسل مخه من خبرته في التعامل مع بقية البرامج لنفرض عليه بدلاً منها أسلوب برنامجناوفلسفته في التصميم التي قد لا تناسبه.

إنشاء تطبيقات تعمل كما يريد المستخدم

إن أداة البرمجة، ولغة Visual Basic مثل أدوات الصانع أو الحرفي، تتطور مع الزمن لتسهيل عمل الصانع ولإتاحة الفرصة له ليتقن حرفته، مثلا المنشار للنجار تطور من الصورة اليدوية إلى الكهربائية إلى أن وصلنا إلى المنشار الذي يدار بالكمبيوتر. إلا أن الأداة الجيدة لن تهب الصانع المهارة ولا الخبرة،كما أن الحرفي غير الماهر الذي لا يستطيع استخدام المنشار اليدوي لن يستطيع بالتبعية استخدام المنشار الموجه بالكمبيوتر.

إن Visual Basic رغم قوته لن يجعل المبرمج أفضل، ولكنه يسهل عمل المبرمج الجيد، لذا فعليك أن تعلم كيف تكون مبرمجاًومطوراً جيداً للبرامج قبل أن تتعلم Visual Basic.لكي تحقق ذلك لابد أن تحدد بوضوح النقاط الآتية: ما هو البرنامج الذي تريده؟،

إلى من ستقدم هذا البرنامج ؟،وكيف ستقوم بإتمام هذا البرنامج ؟. وكلما تفهمت طبيعة المستخدم وطبائعه، كلما أثر ذلك بشدة على نجاح برنامجك.

يمكننا أن نقسم تطوير البرامج إلى المراحل الآتية:

مرحلة ما قبل الإنتاج

تهدف هذه المرحلة إلى تعريفوتحديد للمنتج المتوقع،وكذلك هدفهوخواصه بالتفصيل.وكذلك ما هي الخواص التي سيتم التعجيل بها،والأخرى التي ستؤجل إلى إصدارات مقبلة.

أيضا في هذه المرحلة يتم رسم ملامح للمستخدم أي يتم التعرف على إمكاناتهوتفضيلاته، مثلا هل المستخدم متمرس في استعمال **Windows** أم أنه لا يعرف ما هو الكمبيوتر (بعض البرامج تستهدف هؤلاء).

من الأشياء التي تحدد أيضا في هذه المرحلة مستقبل البرنامج ، هل يستهدف قطر أو لغة معينة أم أنه سيكون منتج عالمي يستهدف نشره في أقطار عديدةوبلغات مختلفة.

أيضا يجب تحديد معدل استخدام البرنامج لأن هذا سيحدد السرعة المطلوبة لاستجابته، البرنامج الذي يستخدم مئات المرات في اليوم من عشرات المستخدمين (كبرامج قواعد البيانات) لا يمكن أن تكون سرعته كبرنامج حفظ نسخة احتياطية يتم تشغيله مرة كل شهر. مثال على ذلك برنامج يعمل على أحد صفحات الإنترنت، لن يكون مقبولا بالمرة أن يأخذ وقتا طويلا ليظهر للمستخدم وإلا سيهرب إلى صفحة أخرى غير هذه التي تضيع له وقته.

مرحلة الإنتاج

في هذه المرحلة يتم كتابة الكود الفعلي للبرنامج، قبل ذلك يجب تحديد أية لغة سيتم كتابة هذا الكود بها ! ، نظرا لأن كتابنا عن **Visual Basic** فليس لنا خيار إلا أنه في المشاريع الحقيقية قد تكون لغة معينة أنسب للتطبيق لاعتبارات عديدة.

يأتي بعد ذلك أمر هام في المشاريع الكبيرة عامة ،وهو التحكم في الكود. إن الكود الذي يقوم بكتابته العشرات من المبرمجين متفرقين على أجهزة مختلفة على شبكة محلية ، لابد من وسيلة ما لربطه ومتابعة تجديده أولا بأول لإتمام الترجمة الكاملة للبرنامج.

يأتي في هذه المرحلة أيضا ما يسمى باختبار النظاموهو اختبار البرنامج من ناحية الأخطاء، صعوبة الاستخداموالتوافق مع الخصائص. عملية الاختبار هذه قد تتم للمشروع ككل، عن طريق أفراد منفصلين عن التطوير يقومون بالاختبار ثم يعودون بنتائجهم للفريق الأول ليقوم بتصحيح الأخطاء، هذه الطريقة تطيل وقت الإخراج النهائي للمشروع الكامل، بينما تسهل المراحل الأولى للإنتاج ولكن نتائجها مضمونة نوعا ما نظرا لاختبار البرنامج كاملا.

خلافاً لذلك قد ندمج عملية الاختبار في عملية التطوير، بحيث يرافق أفراد الاختبار أفراد التطوير خطوة بخطوة، يختبروا جزئيات البرنامج الصغيرة أولا بأول، بالطبع إخراج جزئية من البرنامج سيطول إلا أن أخطاء المنتج النهائي ستكون أقل احتمالا.

أياً كانت الطريقة فإن المسؤولية الأولى للاختبار تقع على عاتق المبرمج نفسه.

مرحلة ما بعد الإنتاج

في هذه المرحلة تقوم بإنشاء وثائق للمشروع، وتحضر البرنامج للنشر، ثم تقيم عملية التطوير برمتها. تقوم هنا بإنشاء الوثائق المطبوعةوالإليكترونية لمساعدة المستخدموشرح البرنامج. أيضا تقرر وتجهز وسيلة توزيع ونشر البرنامج.وتعرض نسخة تجريبية للمستخدم ما أمكن.

التوثيق

بعد وصول المنتج لصورة مقبولةوقليلة الأخطاء نوعا ما، يمكننا البدء في كتابة وثائق شارحة للبرنامج،والأهم من ذلك الوثائق الفنية التي تشرح كيفية عمل البرنامجوطريقة تعديله، حتى لو كنت أنت الذي ستقوم بتعديل الكود في المستقبل، فإنك لا تضمن ذاكرتك أو حتى توفيرا للوقت الذي ستأخذه لفهم برنامجك من جديد. للأسف يقوم الكثير من المبرمجين بتجاهل هذه الخطوة مما يعد أحد العيوب في المبرمج خاصة لو كان سيعمل في شركة لإنتاج البرامج.

التوزيع

بعد تجهيز الوثائقوالملفات التنفيذية للبرنامج يأتي تحديد على أي وسيط ستنشر برنامجك : أقراص مرنة (آخذة في الانقراض شيئا فشيئا) أم على أقراص مدمجة (ليزر) أو أقراص **DVD**، أم مباشرة من الإنترنت.

عليك أولاً تجهيز ملفات التثبيت لبرنامجك، تذكر أنه مهما كان نجاح برنامجك فإن جزء التثبيت له أهمية كبيرة، إنه واجهة عملك، التثبيت السهل يعجب المستخدمينويلفت أنظارهم. يمكنك هنا استخدام أي من برامج إنشاء ملفات التثبيت المتاحة (مثل **Setup Wizard** الذي يأتي مع **Visual Basic**)ولكن أيا كان البرنامج الذي ستستخدمه عليك باختبار التثبيت على أكبر قدر من الأجهزة تستطيع الوصول إليه. تذكر أنه مهما كان نجاح برنامجك فإنه سيكون فاشلا في عين المستخدم إن لم تتم عملية التثبيت على جهازه بنجاح.

اختيار الوسيط أحد عوامل نجاح التثبيت، إن البرامج الكبيرة من غير المناسب أن توضع على أقراص مرنة (ستلحق بمصير الديناصورات عما قريب) سواء في عددها أو في الزمن الذي سيأخذه التثبيت. بعد اختيارك للوسيط قم بكتابة تعليمات استخدامه بصورة واضحةوبسيطة عليه، مثلا عنون الأقراصورقّمها، اكتب تعليمات التثبيت على علبة القرصوهكذا. أيضا حاول أن تجمع كل ما تريد تثبيته في برنامج واحد قدر الإمكان، إن الملف الواحد للتثبيت دائماً هو أنجح الخيارات. إن كنت ستستخدم اسطوانات مدمجة، قد يكون من المفيد وضع ملفات التشغيل التلقائي **Autorun.Inf** على الدليل الرئيسي لتسهيل التثبيت على المستخدم.

التقييموالتخطيط للإصدارات المقبلة

ليس هناك مشروع برمجي لا يتطور، ليس فقط لأننا نتبين مع الوقت صفات جديدة من المفيد إضافتها، بل قد تكون معروفة حتى قبل الإصدار الحالي، ولكن نظراً للارتباط بجداول زمنية، قد يحول عامل الزمن دون إتمامنا (اختياريا) لبعض المزايا في البرنامج. هذا جانب، الجانب الآخر أنه مع الخبرة المتزايدة من مشروع لمشروع نستطيع اكتشاف الأخطاء التي كلفتنا الأموالوالوقت، لنقوم بتجنبها فيما يقبل من المشاريع.

في الحقيقة إن تقييم برنامجك لا يتم من خلالك وحدكومن وجهة النظر السابقة فقط، بل إن هناك الاستخدام الفعلي للبرنامج من قبل المستعملينوهذا هو التقييم الفعليوالحقيقي للبرنامج.

إنشاء الواجهات الرسومية المتناسقة والفعالة

لقد وضع Windows بعض الأشكال القياسية للواجهات الرسوميةوالتي تلقاها المستخدم بالقبول، ذلك لتناسق هذه الواجهاتوخضوعها نوعا ما لنظام محدد، إن التناسق هو أحد أهم الأشياء في الواجهة الرسومية،وفقدانه يضيع فائدتها بالمرة.

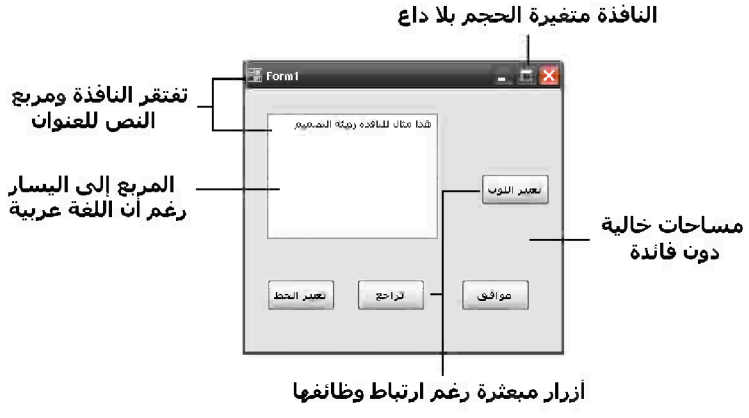
لا يعني خضوعك للأشكال القياسية الحجر على حريتك في الإبداع وتقديم الجديد إلى الواجهات الرسومية، المهم أن تكون سهلة الاستخدام ومحببة للمستخدم،وهذا هو المقياس في النجاح.

نوضح فيما يلى أهم المفاهيم التى تساعدك فى إنشاء الواجهات الرسومية المتناسقة والفعالة.

التصميم الجيد للنماذج

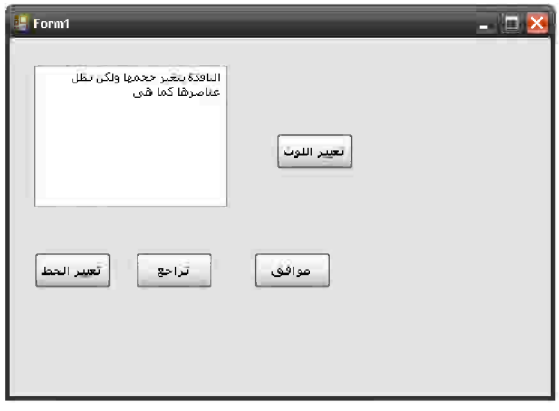
رغم سهولة إنشاء نوافذ النماذج في Visual Basic إلا أن إتقان ذلك بصورة جيدة ليس باليسير.ولإنشاء نموذججيد عليك بفهم وظيفة النموذج،واستخدامه وعلاقته بأجزاء البرنامج الأخرى.

سننطلق من النموذج المبين في الشكل ١٧-١لشرح معنى النموذج جيد التصميم، يستخدم هذا النموذج لإدخال بعض الملاحظات مع إمكانية تغيير اللون أو الخط ويحتوي على كثير من عيوب التصميم التي سنوضحها فيما يلي:



شكل ١٧-١مثال لنموذج معيب رغم أن وظيفته قد تكون خالية من العيوب

- أول ما نلاحظه أن نافذة هذا النموذج قابلة لتغيير حجمها، رغم أن هذا من ناحية ليس مهما لوظيفتها، لأن تثبيت الحجم لم يتم أخذه فى الاعتبار عند كتابة الكود وطبعاً العناصر لن يتغير حجمها تبعاً لتغير حجم نافذة النموذج. فلو قام المستخدم بتغيير الحجم مثلا سيجد النافذة كما في الشكل ١٧-٢.



شكل ١٧-٢ النافذة السابقة عند تغيير حجمها لا تغير من حجم عناصرها

كما ترى، ليس هناك داع لتغيير حجم النافذة طالما يمكن تأدية الوظائف في الحجم الأول، يمكننا إلغاء خاصية تغيير الحجم بتحويل خاصية النموذج **FormBorderStyle** إلى القيمة **FixedSingle** أو **FixedDialog** بدلاً من **Sizable**.

- الافتقار إلى العنونة، فمربع النص بلا عنوان،وكأن المصمم يعتمد على معرفة المستخدم لوظيفته بالتخمين أو البديهية،وهذا غير مقبول، لابد من وجود عنوان لكافة تفاصيل النافذة.
- الفراغات الزائدة، والأماكن العشوائية للعناصر الموجودة،وعدم تناسق مظهرها.
- عدم التنظيموالتبويب لعناصر النموذج تبعا للوظيفة لتسهيل الاستخدام، فتجد زر "تغيير الخط" منفصل عن زر "تغيير اللون" رغم تشابههما الوظيفي مما قد يسبب تشتيت للمستخدم.
- اتجاه النافذة من اليسار إلى اليمين رغم أن اللغة عربية، لابد لذلك من تحريك مربع النص إلى اليمينوالأزرار إلى اليسار.

انظر إلى النافذة في الشكل ١٧-٣ حيث تم تلافي جميع هذه العيوب. مع زيادة أخرى هي إدراج الإطار لفصل الوحدات الوظيفية للنافذة، أيضا تغيير عنوان النافذة إلى كلمات معبرة.

شكل ١٧-٣ النافذة السابقة بعد معالجة عيوب التصميم فيها.

تصميم القوائم

تعتبر القوائم من أهم مكونات برنامج Windows، لقد كانت فلسفة تصميم Windows أن تكون القوائم هي الأداة الرئيسية لإصدار الأوامر ،وبقية الوسائل كشرائط الأدوات عناصر إضافيةوليست رئيسية ، أي كل أمر يمكن للبرنامج تنفيذه لابد من وجوده في قائمة الأوامر.

والقوائم أيضا ينبغي أن تتمتع بالتناسقوالإيجازوالتنظيم. لقد ناقشنا بعضا من ذلك في فصل إنشاء القوائم وتحدثنا عن القوائم القياسية،والآن نكمل حديثنا بشئ من التفصيل بسرد عدد من الخطوط العامة التي تفيد في التصميم الجيد للقوائم.

- اتبع قدر استطاعتك النظام القياسي للقوائم (File , Edit ,View, ...) أو نظائرها بالعربية (ملف، تحرير، عرض .....).
- قم بتجميع عناصر القوائم منطقياوتقليل القوائم الرئيسية قدر المستطاع.
- استخدم عند الحاجة الفاصل Separator بين العناصر المختلفة في القائمة الواحدة.

- لا تكرر نفس الأمر في قائمتين مختلفتين،ولا تستخدم أسماء متشابهة لأمرين مختلفين.
- لا تستخدم عناصر القائمة الرئيسية دون قوائم فرعية تحتها.
- استخدم النقاط (...) بعد اسم القائمة التي تفتح مربع حوار عند اختيارها لتمييزها عن غيرها، فمثلا قائمة تستخدم في فتح ملف عن طريق مربع الحوار " فتح " يجب أن تظهر هكذا "فتح ..." نقوم فيما يلى بتوضيح هذه النقاط بتفصيل أكثر.

اتباع نظام القوائم القياسي

لقد استقر نظام واجهة **Windows** في ذهن المستخدم، حتى صار الخروج عن الأشكال القياسية مدعاة للالتباسوالأخطاء. مثلا اعتدنا على القوائم أن تأتي بالترتيبملف، تحرير، ...وهكذا.

فليس من المقبول أن يأتي برنامجويخالف ذلك حتى لو كان لديه سبب ما، انظر على سبيل المثال للنموذج الموجود بشكل ١٧-٤ والذى لا يخضع الترتيب فيه للشكل القياسي،مما يتسبب فى أخطاء أثناء الاستخدام فضلا عن الضيق الذي يسببه للمستخدم.

شكل ١٧-٤ قوائم غير مرتبة بالشكل القياسي.

تجميع القوائم بطريقة منطقية

الشكل ١٧-٥ التالى عالج المشكلة السابقة، إلا أنه يبين نوعا آخر من عيوب تصميم القوائموهو الجمع بين عناصر غير متشابهة في قائمة واحدة. فالقائمة "أدوات" تحتوي على "خيارات"و"طباعة" علماً بأن الطباعة توضع في قائمة "ملف" قياسيا.

شكل ١٧-٥ قوائم مجتمعة علما بأنها مختلفة في الوظيفة.

استخدام الفاصل

يقوم الفاصل بتقسيم القائمة الواحدة إلى مجموعات منفصلةويسهل على المستخدم البحث عن الأوامر، مثلا قمنا في الشكل ١٧-٦ بمعالجة المشكلة الموجودة في شكل ١٧-٥ بنقل محتويات القائمة "أدوات" كلها إلى قائمة ملف،واستخدمنا الفواصل لتوضيح الفارق بين طبيعة الأوامر.

شكل ١٧-٦ استخدام الفواصل لتوضيح الاختلاف في طبيعة أوامر القائمة الواحدة

تجنب القوائم المكررة

لقد عالج الشكل ١٧-٦السابق مشكلة الوضع الخاطئ لعناصر بعض القوائم إلا أنه أنشأ مشكلة أكبر، وهي تكرار العناصر في أكثر من قائمة،لأنه أبقى على قائمة "أدوات" وذلك من شأنه إحداث اضطراب لدى المستخدم. لذلك يفضل إزالة العناصر التي بلا فائدة. كما يظهر في شكل ١٧-٧.

شكل ١٧-٧ النافذة بعد إزالة القائمة "أدوات" المكررة

تجنب القوائم الرئيسية بدون قوائم فرعية تحتها

يطلق على القوائم الرئيسية التي لا تحتوي على قوائم فرعية القوائم اليتيمة،وهي تشغل مساحة هامة على شريط القوائم الرئيسي،والمستخدم يتوقع دائما قائمة منسدلة عند نقر قائمة رئيسية، ولا يتوقع مربع حوار كالذي يظهر عند ضغط قائمة "عن"،والتي يفضل إزالتها لتظهر النافذة بقوائم مختصرة كالشكل ١٧-٨.

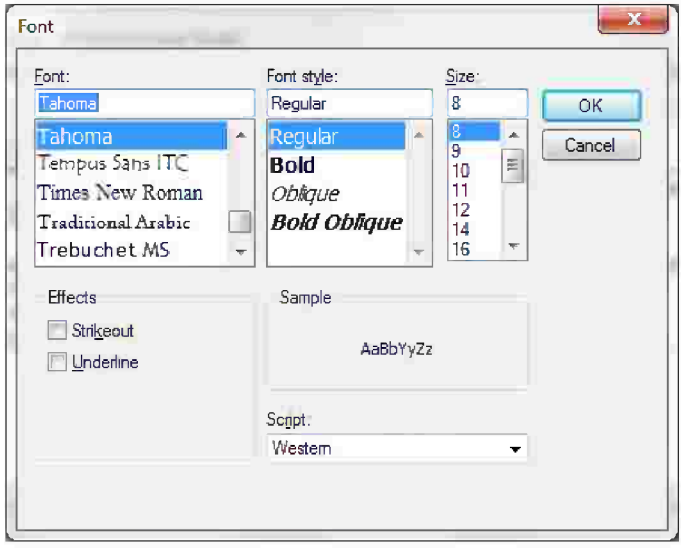
شكل ١٧-٨ إزالة القائمة "عن"والتي تشغل حيزا من القائمة الرئيسية بلا فائدة.

استخدام النقاط في حالة القوائم التي تفتح مربعات حوار

من الأشكال القياسية للقوائم استخدام ثلاث نقاط تلي القائمة إذا كانت ستفتح مربع حوار. تذكر دائماً استخدامها، لتعطي مظهرا محترفا لتطبيقاتك.

تقديم البدائل للمستخدم

عند رغبتك في اتخاذ المستخدم لقرار ذى خيارات محدودة، يستحسن أن تمده بوسيلة ليختار منها بدلا من إدخال الخيار كتابةً. سيسهل ذلك على المستخدم، كما سيقلل من احتمالية المدخلات الخاطئة كما أنه سيسهل للغاية من عملية التحقق من المدخلات. كمثال على ذلك مربع الحوار الخاص باختيار الخط وحجمه (انظر شكل ١٧-٩).



شكل ١٧-٩ المربع الحوارى **Font**نموذج لتقديم الخيارات للمستخدم

مربع السرد مقابل مربع السرد والتحرير

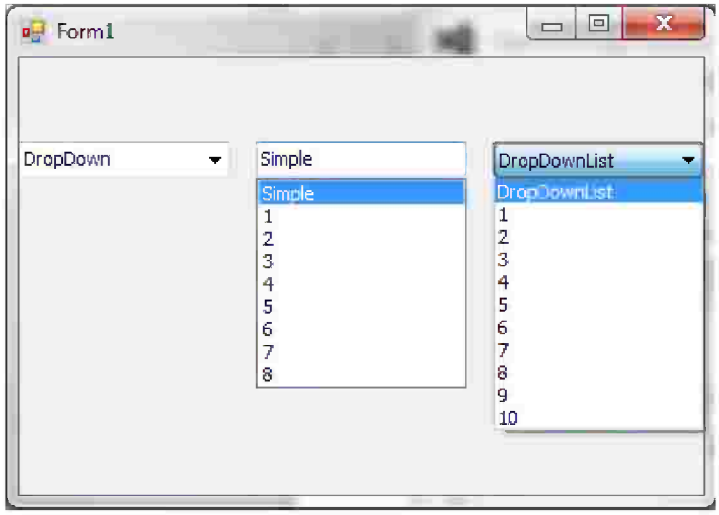
من الأدوات المناسبة لعرض الخيارات مربع السرد **ListBox**، حيث يمكنك اختيار أحد العناصر (أو أكثر من عنصر لبعض التطبيقات).ولكن يعاب على هذه الأداة عدم قدرة المستخدم على إدخال قيمة جديدة غير القيم الموجودة بالفعل. أيضا يستهلك مربع السرد مساحة النافذة بطريقة غير جيدة.

من الأشياء القياسية في مربعات السرد أن المستخدم يجب أن يستطيع الاختيار بتعليم عنصر ثم الضغط على زر أو بالنقر المزدوج مباشرة على العنصر

عند رغبتك في توفير مساحة النافذة أو إتاحة الفرصة للمستخدم لإدخال عناصر جديدة غير المسرودة أمامه فإن الأداة المثلى هي استخدام "مربع السرد والتحرير" **ComboBox**،ولهذا المربع عدة أنماط تحدد بالخاصية **DropDownStyle** تظهر في شكل ١٧-١٠وهي:

- **DropDown**: قائمة منسدلة من الخيارات، ومربع نص لإدخال قيم أخرى.
- **Simple**: مربع سرد عادي فوقه مربع نص (وهو نمط نادر الاستخدام).

- **DropDownList**: قائمة منسدلة فقط ولا يوجد مربع إدخال لإدخال قيم أخرى (أي أن هذا النمط مكافئ لمربع السردولكن بحجم مصغر).

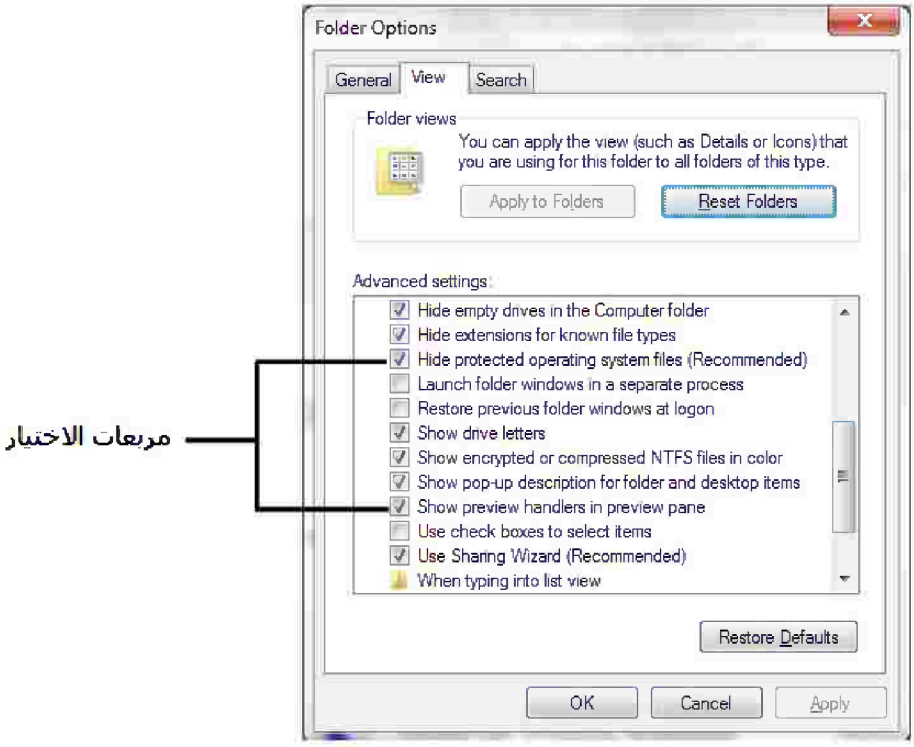


شكل ١٧-١٠الأنماط المختلفة من مربع السرد والتحرير.

مربعات السردوالمربعات المنسدلة كلاهما أدوات مفيدة إلا أن الإسراف في استخدامهما قد يكون ذو تأثير ضار، فكلما زادت العناصر فيهما زاد وقت تحميل النموذج،واعترى البرنامج بطء في العرضوالتنفيذ. لذا لابد بملء المربعات بالعناصر المتوقعة للاختيار فقط.

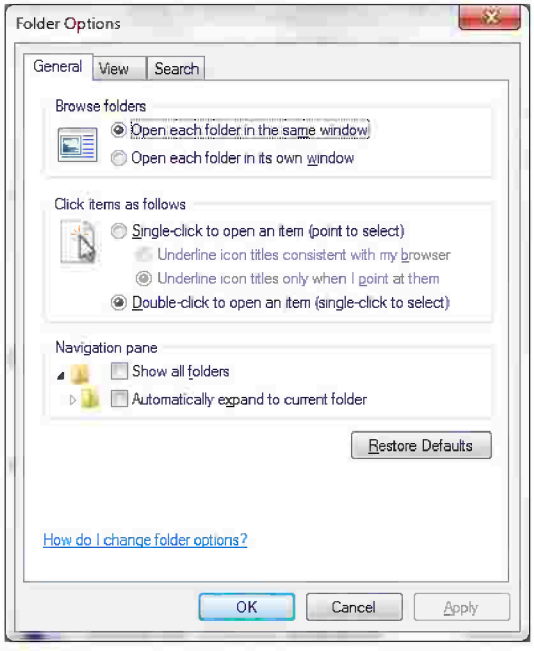
أزرار الاختيار (Radio Buttons)مقابل مربعات الاختيار (Check Boxes)

لتقديم عدد محدود من الخيارات نستخدم مربعات الاختيار **Check Boxes** وأزرار الاختيار **RadioButtons**، تستخدم مربعات الاختيار عند وجود خيارين فقط (نعم ، لا) مثل المستخدمة في المربع الحوارى **Folder Options** الذي يتم إظهاره من مستكشف **Windows Explorer** كما يبدو في شكل ١٧-١١.



شكل ١٧-١١ استخدام مربعات الاختيار في المربع الحوارى **Folder Options** في مستكشف **Windows**.

تستخدم أزرار الاختيار على الجانب الآخر عندما تكون الخيارات أكثر من اثنين أو ليست من النوع (نعم، لا)،وهي تستخدم في مجموعاتوتوضع في حاوية،وعند اختيار إحداها يلغى اختيار الآخرين تلقائيا. نموذج لاستخدام أزرار الاختيار في **Windows** يظهر في شكل ١٧-١٢ الذي يبين التبويب **General** بالمربع**Folder Options** والأزرار تتيح لك التبديل بين فتح المجلدات والملفات بالنقر المزدوج أو بالنقر المفرد.



شكل ١٧-١٢استخدام أزرار الاختيار في المربع الحوارى Folder Options .

بناء الانطباع الجيد عن البرنامج لدى المستخدم

انطباع المستخدمومملاحظاته الأولية عن البرنامج تساهم إلى حد كبير في نجاحه. من أهم هذه الانطباعات على سبيل المثال سرعة البرنامج، والتي قد تكون بذلت جهدا كبيرا في تحسينها إلا أنك قد لا تستطيع الإيحاء بهذا الانطباع لدى المستخدم،وترجع هذه النقطة أيضا إلى تصميم الواجهة. هناك العديد من طرق الخداع للمستخدم لتقليل إحساسه بالوقت عند استخدامه لبرنامجك، لتخلق لديه انطباعا جيدا عن السرعة. مثال لذلك الأحداث التي تتم عند بدء Windowsوالتي تعتبر بطيئة نوعا ما عند بدء التشغيل(Booting) إلا أن الأصواتوالرسوموالحركةوتنويع الشاشات يلفت انتباه المستخدم بعيدا عن ذلك.

أول ما يجب عليك ملاحظته: أن المستخدم يحب أن يجد ردا عندما ينقر أحد الأزرار، لا تتأخر في الرد. يمكن أن تتأخر في تنفيذ وظيفة لكن من غير المقبول ألا ترد على زر أو قائمة. هناك العديد من العوامل لزيادة سرعة البرنامج نسردها فيما يلي:

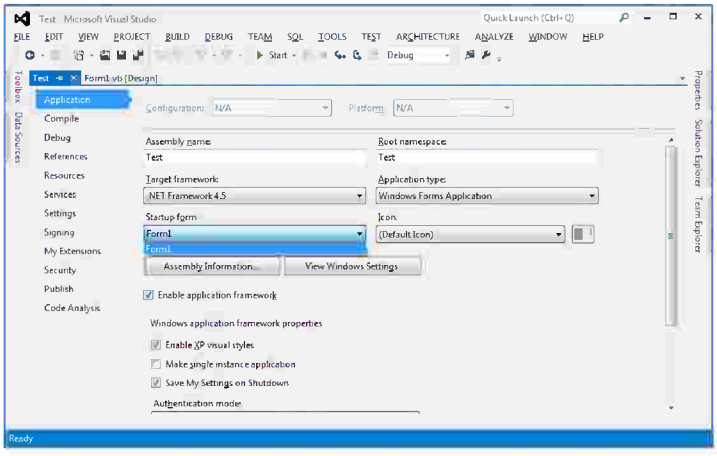
التحميل المبدئي للبرنامج

إذا قمت بتحميل كافة النماذج في البرنامج عند بدايته فإنها ستستقر في الذاكرة مما يساهم في سرعة ظهورها للمستخدم، هذا الأسلوب يبطئ قليلا من وقت التحميل المبدئي لكنه يسرع بالبرنامج كثيرا بعد ذلك. لكن لاعتبارات الذاكرة لا يمكن استخدام هذه الطريقة إذا زاد عدد النوافذ عن ٥ مثلا حيث سيؤدي ذلك إلى نتيجة عكسية إذ يحتاجWindowsفي أغلب الأحيان إلى التبديل بين الذاكرة الحقيقيةوملف التبديل على القرص الصلبSwappingوهي عملية طويلة للغاية.

في حالة النوافذ الكثيرة يستحسن تقسيمها إلى مجموعاتوتحميل الجزء القريب من الاستخدام فقط، كأن نحمل نوافذ إدخال البيانات في نظام لقاعدة البيانات عند طلب المستخدم إدخال بيان واحد مثلا، ونحمل نوافذ البحث عند طلب المستخدم أحد الاستفسارات.

استخدام الإجراء SubMain

بدلاً من بدء البرنامج من خلال نموذج معين، يمكنك أيضاً بدء البرنامج من خلال إجراء Submainوذلك باختيار من مربع السرد StartUpForm داخل نافذة صفحات خصائص المشروع كما يبدو في شكل ١٧-١٣.استخدم هذه الطريقة إذا كنت تحتاج عملية استهلال كبيرة Initialization للمتغيراتوالكائنات، مثل فتح ملفات الإعدادات، التحقق من مسار البرنامج ، الاتصال بقاعدة بيانات خارجية.وإذا استغرق ذلك وقتا طويلا فغالبا ما سنحتاج لشاشة تجذب الانتباه.



شكل ١٧-١٣استخدام نافذة خصائص المشروع لتحديد البدء بـSubMain

تجنب الأخطاء الشائعة في كتابة الكود

أيا كان وضعك في العمل،وأيا كان حجم مؤسستك إن كنت تعمل في مؤسسة، أو حجم عملك الخاص إن وجد في حقل البرمجة ، فلا مفر من حاجتك إلى قراءة شخص آخر للكود الذي تكتبه، إما في الوقت الحاضر أو في المستقبل. هذا الأمر يحتم عليك أخذ كتابة الكود بعين الاعتبار، أي شكله المقروء فضلا عن وظيفته.

كتابة كود سهل مقروء

هذا الأمر مهم، حتولو كنت الوحيد الذي سيقرأ الكود، محاذاة العباراتوضبطها فوق بعضها لا يساهم فقط في تحسين شكل الكود،وإنما أيضا في تسهيل فهم وظيفته.

هذا الأمر يصبح في المرتبة الأولى من الأهمية إن كنت ستتبادل هذا الكود مع غيرك، كما لو كنت تعمل في شركة كبيرة، يعطي أيضا هذا الأمر انطباعا عن دقة وطريقة تفكيرك.

استخدام ثوابت VisualBasic مسبقة التعريف

في Visual Basic 2012صار استخدام الثوابت المسماة أسهل من أي وقت مضى، إنك تكتب الثابت مباشرةولا تحتاج لإضافة ملفات تعريف أو ما شابه كما كان يحدث سابقا. لذا لم يعد هناك أي عذر لك في عدم استخدام هذه الثوابت. حاول دائما تدريب نفسك على

هذه الثوابت بدلا من قيمها الرقمية. تخيل نفسك تقرأ السطر التالي من الكود بعد فترة من كتابته:

```
MessageBox.Show("Are you happy", "Hi", 4)
```

بالطبع لن تتذكر دلالة الرقم 4 حتى لو كانت ذاكرتك حديدية.ولكن انظر إلى السطر التالي:

```
MessageBox.Show("Are you happy", "Hi",
MessageBoxButtons.YesNo)
```

لا شك أن فهم المعنى حاليا صار أيسر كثيرا، علما بأن استخدام هذه الثوابت لن يقلل مطلقا من كفاءة البرنامج عند التشغيل فهي مكافئة لاستخدام الأرقام تماما.

استخدام الملاحظات

لن أطيل في هذه النقطة، تذكر دائما أن تكتب وظائف كل ما قد يصعب فهمه من الكود في ملاحظة (تذكر أن الملاحظة تبدأ بعلامة ' أو كلمة **REM**)، هذا الأمر قد تتكاسل عنه في البداية، ولكن كلما زاد حجم برامجك أصبحت الملاحظات شئ من صميم البرنامج،وإلا صار من الصعب جدا تعديله.

استخدام أسماء معبرة للمتغيراتوالكائنات

يتيح لك **Visual Basic** تسمية المتغيرات بأسماء يصل طولها إلى ٢٥٥ حرف.وأسماء الكائنات بأطوال تصل إلى ٤٠ حرف.وذلك كافٍ للتعبير عن وظيفة المتغير أو الكائن. لذا لا تتبع العادات السيئة مثل تسمية المتغيرات **myvar1** و**myForm** و **c2** وأمثالها ، أو ترك الأسماء الافتراضية **TextBox1** و**TextBox2**، إن متابعة كود مكتوب بهذه الطريقة يشبه السير في متاهة،ومن الأيسر كتابة البرنامج ثانية عن أن تعدل في كود كهذا. الأمر أيضا يزداد أهمية إذا كان هناك من سيقرأ الكود بعدك.