ملخص شامل ومبسط للغة البرمجة C#

لغة C# (سي شارب) هي أداة قوية ورائعة لبناء مختلف أنواع البرامج. هذا الملخص سيأخذك في جولة لفهم أساسياتها وميزاتها الهامة بطريقة سهلة ومباشرة.

1. ما هي C# ولماذا هي مهمة؟

- **ببساطة:** C# لغة برمجة حديثة طورتها شركة مايكروسوفت. تخيلها كصندوق أدوات متطور يمكنك استخدامه لبناء:
  - برامج تعمل على الكمبيوتر (سطح المكتب).
  - مواقع وتطبيقات ويب تفاعلية.
  - ألعاب (خاصة باستخدام محرك Unity الشهير).
  - تطبيقات للموبايل (أندرويد و iOS).
  - خدمات تعمل في الخلفية أو على السحابة (Cloud).
  - وأكثر من ذلك بكثير!
- **تاريخ موجز:** ظهرت C# لأول مرة عام 2002 وتطورت كثيرًا عبر السنين، حيث أضافت مايكروسوفت ميزات جديدة مع كل إصدار لجعلها أقوى وأسهل في الاستخدام. الإصدارات الحديثة (مثل C# 12 مع .NET 8) تركز على تحسين الأداء وتبسيط كتابة الكود.
- **علاقتها بـ .NET:** C# تعمل بشكل أساسي مع منصة اسمها ".NET" (دوت نت). فكر في .NET كبيئة التشغيل والمكتبة الضخمة التي توفر لـ C# الأدوات والمكونات الجاهزة لبناء التطبيقات. .NET الحديثة تعمل على أنظمة تشغيل مختلفة (Windows, macOS, Linux).

2. أساسيات الكتابة بلغة C# (بناء الجملة)

كتابة الكود في C# تشبه كتابة تعليمات للكمبيوتر، وهناك قواعد بسيطة:

- **الأوامر (التعليمات):** كل أمر أو تعليمة تنتهي بفاصلة منقوطة ; مثل: int age = 25;.
- **الكتل البرمجية:** مجموعة أوامر توضع بين قوسين معقوفين { }. تستخدم لتجميع الكود المتعلق ببعضه (مثل داخل دالة أو شرط).
- **حالة الأحرف:** C# تفرق بين الحروف الكبيرة والصغيرة. myVariable يختلف عن MyVariable.
- **التعليقات:** ملاحظات لك أو للمبرمجين الآخرين لشرح الكود. لا يقرأها الكمبيوتر.
  - // تعليق لسطر واحد
  - /* تعليق يمتد
    لعدة أسطر */
- **مساحات الأسماء (Namespaces):** طريقة لتنظيم الكود وتقسيمه إلى مجموعات منطقية لتجنب تضارب الأسماء. نستخدم using في بداية الملف للوصول إلى الأصناف الموجودة في مساحة أسماء معينة بسهولة (مثل using System;).

3. تخزين البيانات: المتغيرات وأنواعها

البرامج تحتاج لتخزين معلومات مؤقتة أثناء عملها. نستخدم لذلك:

- **المتغيرات (Variables):** مثل صناديق صغيرة في ذاكرة الكمبيوتر لها اسم ونوع محدد لتخزين قيمة يمكن تغييرها.
  - **أمثلة لأنواع البيانات الأساسية:**

- int: للأعداد الصحيحة (مثل 10, -5, 0).
- double أو float: للأعداد العشرية (مثل 3.14, -0.5). decimal للأرقام المالية التي تحتاج دقة عالية جدًا.
- bool: لتخزين قيمة منطقية، إما true (صحيح) أو false (خطأ).
- char: لتخزين حرف واحد (مثل 'A', 'ب', '$'). توضع بين علامتي اقتباس فرديتين.
- string: لتخزين نص أو سلسلة من الحروف (مثل "أحمد", "مرحباً بالعالم!"). توضع بين علامتي اقتباس مزدوجتين.
  - **تعريف المتغير:** نوع_البيانات اسم_المتغير = قيمة_أولية;

```
int userAge = 30;       // متغير لتخزين العمر
string userName = "سارة"; // متغير لتخزين الاسم
bool isActive = true;    // متغير لتخزين حالة النشاط
```

- **الثوابت (Constants):** مثل المتغيرات لكن قيمتها ثابتة لا يمكن تغييرها بعد تحديدها أول مرة. نستخدم const لتعريفها.

```
const double Gravity = 9.8;
```

- **الفرق بين أنواع القيمة (Value Types) وأنواع المرجع (Reference Types):**
  - **أنواع القيمة (مثل int, double, bool, struct):** المتغير يخزن القيمة الفعلية مباشرة. عند نسخها، يتم إنشاء نسخة جديدة مستقلة من القيمة.
  - **أنواع المرجع (مثل string, array, class, delegate):** المتغير يخزن "عنوان" أو "مرجع" لمكان القيمة في الذاكرة. عند نسخها، يتم نسخ العنوان فقط، ويشير المتغيران لنفس البيانات الأصلية في الذاكرة. (تغيير البيانات عبر مرجع يؤثر على الآخر).

4. إجراء العمليات: عوامل التشغيل (Operators)

أدوات لإجراء عمليات حسابية، مقارنات، أو منطقية على المتغيرات والقيم:

- **حسابية:** +, -, *, / (قسمة), % (باقي القسمة).
- **مقارنة:** == (هل يساوي؟), != (هل لا يساوي؟), >, <, >=, <=. نتيجتها تكون true أو false.
- **منطقية:** && (و - AND), || (أو - OR), ! (ليس - NOT). تُستخدم لربط الشروط.
- **إسناد:** = (لوضع قيمة في متغير), +=, -= (للإضافة أو الطرح من القيمة الحالية للمتغير).

5. تنظيم الكود: البرمجة الشيئية (OOP)

طريقة شائعة لتنظيم الكود تجعله أسهل في الفهم والصيانة وإعادة الاستخدام. تعتمد على فكرة "الأشياء" أو "الكائنات".

- **الصنف (Class):** هو المخطط أو القالب الذي نصمم به نوعًا جديدًا من الكائنات. يصف الصنف ما هي البيانات (الخصائص) وما هي الأفعال (الدوال) التي يمكن لهذا النوع من الكائنات القيام بها.
  - مثال: صنف سيارة قد يحتوي على خصائص مثل لون, موديل, ودوال مثل تشغيل_المحرك().
- **الكائن (Object):** هو نسخة حقيقية يتم إنشاؤها من الصنف.
  - مثال: سيارتي_التويوتا هي كائن من صنف سيارة. لها لونها وموديلها الخاص، ويمكنها تشغيل محركها.

```
(الصنف (القالب //
```

```
public class Book
{
   public string Title; // خاصية: عنوان الكتاب
   public string Author; // خاصية: المؤلف

// دالة (فعل): عرض معلومات الكتاب
   public void DisplayInfo()
   {
      Console.WriteLine($"الكتاب: {Title}, المؤلف: {Author}");
   }
}

// إنشاء كائن (نسخة حقيقية) من الصنف Book
Book myFavoriteBook = new Book();
myFavoriteBook.Title = "بداية ونهاية";
myFavoriteBook.Author = "نجيب محفوظ";
myFavoriteBook.DisplayInfo(); // استدعاء الدالة لعرض المعلومات
```

- **المبادئ الأساسية لـ OOP:**
  - **التغليف (Encapsulation):** إخفاء التفاصيل الداخلية للكائن وحماية بياناته. يتم ذلك عادة بجعل البيانات private (خاصة) وتوفير طرق (دوال أو خصائص public) للتعامل معها.
  - **الوراثة (Inheritance):** صنف يرث خصائص ودوال من صنف آخر (الأب). هذا يساعد على إعادة استخدام الكود. (مثال: صنف قطة يرث من صنف حيوان).
  - **تعدد الأوجه (Polymorphism):** إمكانية أن تأخذ الكائنات أشكالًا متعددة أو تستجيب لنفس الرسالة بطرق مختلفة. (مثال: دالة Draw() ترسم دائرة إذا كان الكائن دائرة، وترسم مربعًا إذا كان الكائن مربعًا).
  - **التجريد (Abstraction):** التركيز على "ماذا" يفعله الكائن بدلاً من "كيف" يفعله، وإخفاء التعقيدات غير الضرورية. يتم تحقيقه باستخدام الواجهات والأصناف المجردة.

6. التحكم في مسار البرنامج (Flow Control)

كيف نجعل البرنامج يتخذ قرارات أو يكرر أوامر معينة:

- **الشروط (if, else if, else):** لتنفيذ كود معين فقط إذا كان شرط ما صحيحًا.

```
int temperature = 25;
if (temperature > 30) {
Console.WriteLine("الجو حار!");
} else if (temperature < 15) {
Console.WriteLine("الجو بارد!");
} else {
Console.WriteLine("الجو معتدل.");
}
```

- **الاختيار من متعدد (switch):** بديل لـ if/else if المتعددة عندما نقارن متغيرًا واحدًا بقيم محددة.

```
char grade = 'B';
switch (grade) {
    case 'A': Console.WriteLine("ممتاز"); break;
    case 'B': Console.WriteLine("جيد جداً"); break;
    case 'C': Console.WriteLine("جيد"); break;
    default: Console.WriteLine("مقبول"); break;
}
```

- **التكرار (Loops):** لتكرار تنفيذ كود معين عدة مرات.
  - **for**: عندما نعرف عدد مرات التكرار مسبقًا.

```
// طباعة الأرقام من 1 إلى 5
for (int i = 1; i <= 5; i++) {
    Console.WriteLine(i);
}
```

  - **while**: لتكرار طالما أن شرطًا معينًا صحيح (يتم فحص الشرط قبل كل مرة).

```
int counter = 0;
while (counter < 3) {
    Console.WriteLine("تكرار...");
    counter++;
}
```

  - **do-while**: مثل while لكنه ينفذ الكود مرة واحدة على الأقل ثم يفحص الشرط.
  - **foreach**: للمرور على جميع العناصر في مجموعة (مثل قائمة أو مصفوفة) بسهولة.

```
string[] colors = { "أحمر", "أخضر", "أزرق" };
foreach (string color in colors) {
    Console.WriteLine(color);
}
```

7. تنظيم المهام: الدوال (Methods)

الدوال هي كتل من الكود لها اسم، تقوم بمهمة محددة. يمكننا استدعاؤها (تشغيلها) متى احتجنا إليها. هذا يجعل الكود منظمًا وقابلاً لإعادة الاستخدام.

- **تعريف الدالة:**

```
// دالة بسيطة لا تُرجع قيمة (void) وتأخذ معامل واحد
public void PrintMessage(string message)
{
    Console.WriteLine(message);
}
```

```
{

// دالة تُرجع قيمة (من نوع int) وتأخذ معاملين
public int Sum(int num1, int num2)
{
int result = num1 + num2;
return result ; // إرجاع النتيجة
{
```

- **استدعاء الدالة:**
```
PrintMessage("مرحباً!"); // استدعاء الدالة الأولى

int total = Sum(5, 7); // استدعاء الدالة الثانية وتخزين النتيجة في total
Console.WriteLine($"المجموع هو: {total}"); // سيطبع 12
```

- **المعاملات (Parameters):** هي المتغيرات التي تستقبلها الدالة لتستخدمها في عملها. يمكن تمريرها بعدة طرق (أشهرها بالقيمة by value، أو بالمرجع ref و out للسماح للدالة بتغيير المتغير الأصلي).
- **التحميل الزائد (Overloading):** يمكن أن يكون لديك دوال بنفس الاسم لكن تختلف في عدد أو نوع معاملاتها.

8. تخزين مجموعات البيانات: المصفوفات والمجموعات

- **المصفوفات (Arrays):** طريقة لتخزين عدد ثابت من العناصر من نفس النوع (مثل قائمة درجات الطلاب). نصل لكل عنصر باستخدام رقم فهرس (يبدأ من 0).
```
int[] scores = new int[5]; // مصفوفة لتخزين 5 درجات (أعداد صحيحة)
scores[0] = 90; // وضع قيمة في العنصر الأول
scores[1] = 85;
Console.WriteLine(scores[0]); // قراءة قيمة العنصر الأول (سيطبع 90)

string[] names = { "علي", "فاطمة", "حسن" }; // تعريف وتهيئة مباشرة
```

- **المجموعات (Collections):** هياكل بيانات أكثر مرونة من المصفوفات (موجودة في System.Collections.Generic). حجمها يمكن أن يتغير، وتوفر وظائف أكثر.
  - **List<T>:** قائمة ديناميكية. يمكن إضافة وحذف عناصر بسهولة. T هو نوع العناصر (مثل List<string> لقائمة نصوص).
```
List<string> shoppingList = new List<string>();
shoppingList.Add("خبز");
shoppingList.Add("حليب");
shoppingList.Remove("خبز");
Console.WriteLine(shoppingList.Count); // عدد العناصر الحالية
```

- **Dictionary<TKey, TValue>**: مثل القاموس. يخزن أزواجًا من (مفتاح، قيمة). نستخدم المفتاح الفريد للوصول السريع للقيمة المقابلة. (مثل Dictionary<string, string> لتخزين اسم الدولة وعاصمتها).

```
Dictionary<string, string> capitals = new Dictionary<string, string>();
capitals.Add("مصر", "القاهرة");
capitals["السعودية"] = "الرياض";
Console.WriteLine(capitals["مصر"]); // يطبع "القاهرة"
```

9. التعامل مع الأخطاء (Exception Handling)

أحيانًا تحدث أخطاء غير متوقعة أثناء تشغيل البرنامج (استثناءات). C# توفر طريقة للتعامل معها بأمان ومنع توقف البرنامج فجأة.

- **try-catch-finally**:
  - **try { ... }**: نضع الكود الذي قد يسبب خطأ هنا.
  - **catch (نوع الخطأ ex) { ... }**: إذا حدث خطأ من النوع المحدد داخل try، يتم تنفيذ الكود هنا. يمكن وضع أكثر من catch لأنواع أخطاء مختلفة. ex هو متغير يحتوي على معلومات عن الخطأ.
  - **finally { ... }**: الكود هنا يتم تنفيذه دائمًا، سواء حدث خطأ أم لا. مفيد لتنظيف الموارد (مثل إغلاق ملف).

```
try
{
   Console.Write("أدخل رقمًا: ");
   int number = int.Parse(Console.ReadLine()); // قد يسبب خطأ لو أدخل المستخدم نصًا
   Console.WriteLine($"الرقم هو: {number}");
}
catch (FormatException) // التقاط خطأ تنسيق الإدخال تحديدًا
{
   Console.WriteLine("خطأ: الرجاء إدخال رقم صحيح.");
}
catch (Exception ex) // التقاط أي خطأ آخر غير متوقع
{
   Console.WriteLine($"حدث خطأ غير متوقع: {ex.Message}");
}
finally
{
   Console.WriteLine("تم الانتهاء من محاولة القراءة.");
}
```

10. الخصائص والفهارس (Properties & Indexers)

- **الخصائص (Properties):** طريقة أفضل وأكثر تحكمًا للوصول إلى بيانات الكائن بدلاً من جعلها public مباشرة. تبدو كأنها متغيرات عادية عند استخدامها، لكنها في الحقيقة دوال get (للقراءة) و set (للكتابة) يمكننا وضع منطق إضافي بداخلها (مثل التحقق من صحة القيمة قبل تخزينها).

```
public class Person
```

```
{
private int _age    ; // حقل خاص لتخزين العمر

public int Age // خاصية عامة للتحكم بالعمر
}
get { return _age; } // عند قراءة قيمة Age
set
{
if (value >= 0 && value <= 120) // التحقق من القيمة قبل التخزين
}
age = value; // value_ هي القيمة التي نحاول إسنادها
else { {
Console.WriteLine("العمر غير صالح!");
{
} // عند إسناد قيمة لـ Age
{

// خاصية تلقائية (الأكثر استخدامًا إذا لم نحتج منطق خاص)
public string Name { get; set; }
{

// الاستخدام
Person p = new Person();
p.Name = "نور";
p.Age = 25; // استدعاء دالة set للخاصية Age
Console.WriteLine($"{p.Name} عمره {p.Age}"); // استدعاء دالة get للخاصية Age
p.Age = 150; // سيطبع "العمر غير صالح!" ولن يتم تغيير قيمة _age
```

- **الفهارس (Indexers):** تسمح لنا بالوصول لعناصر الكائن كأنها مصفوفة باستخدام الأقواس []. مفيدة للأصناف التي تمثل مجموعات.

11. تحديد العقود: الواجهات والأصناف المجردة

طرق لتحقيق التجريد وفرض هيكل معين على الأصناف الأخرى.

- **الواجهات (Interfaces):**
  - هي عقد أو اتفاق يحدد مجموعة من الدوال أو الخصائص التي يجب على أي صنف يريد تنفيذ هذه الواجهة أن يوفرها.
  - الواجهة لا تحتوي على كود تنفيذي، فقط على التواقيع (أسماء وأنواع المعاملات والقيم المرجعة).
  - الصنف يمكنه تنفيذ أكثر من واجهة واحدة.

- تُستخدم لضمان أن أصناف مختلفة لديها نفس القدرات الأساسية. تُعرف بـ interface.

```
الواجهة تحدد "ماذا" يجب أن يفعل المسجل //
public interface ILogger
{
   void Log(string message); // يجب على أي منفذ توفير هذه الدالة
{

// صنف ينفذ الواجهة (يوفر التنفيذ)
public class ConsoleLogger : ILogger
{
   public void Log(string message) { Console.WriteLine($"CONSOLE: {message}");
}
}

// صنف آخر ينفذ نفس الواجهة بطريقة مختلفة
public class FileLogger : ILogger
{
   public void Log(string message) { /* كود الكتابة في ملف هنا */ }
}
```

- **الأصناف المجردة (Abstract Classes):**
  - هي أصناف لا يمكن إنشاء كائنات منها مباشرة، تعمل كقاعدة للأصناف الأخرى لترث منها.
  - يمكن أن تحتوي على دوال عادية (مع كود تنفيذي) ودوال مجردة abstract (بدون كود تنفيذي، يجب على الصنف الوارث توفيرها).
  - الصنف يمكنه الوراثة من صنف مجرد واحد فقط.
  - تُستخدم لتوفير هيكل وسلوك مشترك لمجموعة أصناف متشابهة. تُعرف بـ abstract class.

متى نستخدم أيهما؟ ببساطة: استخدم **الواجهة** لتحديد قدرات أو عقود يمكن لأصناف مختلفة تنفيذها. استخدم **الصنف المجرد** لتوفير أساس مشترك (هيكل أو كود) لمجموعة أصناف مرتبطة ببعضها.

12. ميزات إضافية قوية (نظرة سريعة)

C# تحتوي على العديد من الميزات المتقدمة الأخرى، منها:

- **LINQ:** لكتابة استعلامات (Queries) للبحث والتعامل مع البيانات في المجموعات وقواعد البيانات بطريقة سهلة ومقروءة جدًا.
- **Async/Await:** لكتابة كود يتعامل مع العمليات الطويلة (مثل تحميل ملف من الإنترنت) دون أن يتجمد البرنامج، مما يحسن تجربة المستخدم.
- **Delegates و Events:** آليات متقدمة لتمرير الدوال كمتغيرات وللسماح للكائنات بإرسال إشعارات (أحداث) لكائنات أخرى عند وقوع شيء معين (مثل النقر على زر).

تم إنتاج هذا الملخص بواسطة المبرمج SayyadN.