OOP thinking

Dr. Fadi Tirkawi

التجريد في اللغات غرضية التوجه

The progress of abstraction

- All programming languages provide abstractions.
- Assembly language is a small abstraction of the underlying machine.
- Many so-called “imperative” languages that followed (such as FORTRAN, BASIC, and C) were abstractions of assembly language. These languages are big improvements over assembly language, but their primary abstraction still requires you to think in terms of the structure of the computer rather than the structure of the problem you are trying to solve
- The object-oriented approach goes a step further by providing tools for the programmer to represent elements in the problem space. This representation is general enough that the programmer is not constrained to any particular type of problem. We refer to the elements in the problem space and their representations in the solution space as “objects.”

الميزات الأساسية تمثل طرق الخالصة للبرمجة غرضية التوجه
These characteristics represent a pure approach to object-oriented programming

- **كل شئ هو غرض. تخيل بأن الغرض عبارة عن متحول سحري لتخزين البيانات و يستطيع القيام ببعض الأفعال لنا.**
- **البرنامج هو عبارة عن مجموعة من الأغراض تخبر بعضها من خلال ارسال رسائل**
- **كل غرض له الذاكرة الخاصة به.**
- **كل غرض له نوع. و كل غرض هو مثال عن صنف ما. و الكلمتين الآتيتين هما كلمتان مترادفتان** "class" is synonymous with "type."
- **كل الأغراض من نوع محدد تستقبل نفس النوع من الرسائل.**

الغرض يملك موصل

An object has an interface

- إن مصطلح النوع او الصنف هو مصطلح منذ بدء البشرية. مثال هناك صنف أسماك أو صنف طيور.
- الفكرة أن كل الأغرض أو الأشياء رغم كونها فريدة unique فإنها تكون جزء من صنف معين. و هذا الصنف له شيئين أساسيين هما :
  - الصفات fields or attributes
  - و سلوك behaviors
- ما نفعل في البرمجة غرضية التوجه هو خلق نوع أو صنف جديد نعطيه الصفات و السلوك الذي نريده برمجيا

مثال عن غرض و إستخدام الموصل

```
Light lt = new Light();
lt.on();
```

- Here, the name of the type/class is **Light**, the name of this particular **Light** object is **lt**, and the requests that you can make of a **Light** object are to turn it on, turn it off, make it brighter, or make it dimmer. You create a **Light** object by defining a “reference” (**lt**) for that object and calling **new** to request a new object of that type. To send a message to the object, you state the name of the object and connect it to the message request with a period (dot). From the standpoint of the user of a predefined class, that’s pretty much all there is to programming with objects.

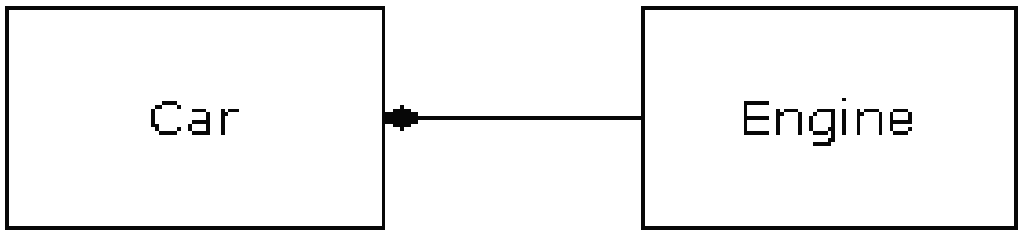
An object provides services

- While you’re trying to develop or understand a program design, one of the best ways to think about objects is as “service providers.” Your program itself will provide services to the user, and it will accomplish this by using the services offered by other objects.

The hidden implementation

- Client programmers
- Public
- Private
- Protected
- without

Reusing the implementation



الفصول و الأبحاث التي يجب تغطيتها على أقل تقدير

- **Object creation, use & lifetimes**
- **Concurrency**
- **Inheritance: reusing the interface**
- **Interchangeable objects with polymorphism**
- **Collections and iterators**
- **Ensuring proper cleanup**
- **Exception handling: dealing with errors**

Java and the Internet

- **Client-side programming**
- **The Web as a giant server**
- **Plug-ins**
- **Security**
- **Internet vs. intranet**
- **Server-side programming**

```
// HelloDate.java
import java.util.*;
 public class Hello {
        public static void main(String[] args) {
        System.out.println("Hello World!");
        }
 }
```

مبادئ و أساسيات في لغة الـ Java

د. فـــــادي تركـــــاوي

جمـــل التعريـــف
Declaration Statement

- هذه الجمل تقوم بتعريف المتغيرات Variables التي سوف يتم إستخدامها داخل البرنامج و ذلك بتحديد نوع البيانات التي تخزن فيها. و يجب أن تأتي جمل التعريف في بداية الطريقة Methods لأن لغة java لا تسمح بإستخدام أي من المتغيرات إلا بعد تعريفها و يتبع تسمية المتغيرات القواعد الآتية:
- يتكون الأسم نم حروف الهجاء و الأرقام
- أن لا يبدأ الأسم برقم
- لا يحتوي على مسافات فارغة
- لايكون من الكلمات المفتاحية المحجوزة
- لا يحتوي على علامات خاصة غير _ , $

Primitive Data Types
أنواع المتحولات الأولية

النوع (type)	الحجم بالبت (Size in bits)	المدى (range) أو القيمة (value)	ملاحظات
boolean	١	True or false	قيمة منطقية
char	16	to FFFF٠٠٠٠	متغير يحمل حرفاً واحداً فقط
byte	8	-128 to +127	
short	16	-32,768 to +32767	قيمة صحيحة في المدى الموضح قرين كل نوع
int	32	-2,147,483,648 to +2,147,483,647	
long	64	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	
float	32	-3.40292347 E+38 to + 3.40292347 E+38	قيمة تحتوي على نقطة عشرية في المدى الموضح قرين كل نوع
double	64	-1.79769313488231570 E+308 to +1.79769313488231570 E+308	

Your First Program in Java

```
1.  // Fig. 2-1: Welcome1.java
2.  // A first program in Java.
3.
4.     public class Welcome1 {
5.  // main method begins execution of Java application
6.
7.   public static void main( String args [ ] )
8.   {
9.  System.out.println( "Welcome to Java Programming!" );
10.
11. } // end method main
12.
13. } // end class Welcome1
```

شرح البرنامج

السطر الأول **// Fig. 2-1: Welcome1.java**

جملة من جمل التعليق.

جملة التعليق Comment Statement

تبدأ بـ// ثم يأتي بعدها أي نص مثل سطر١ ، سطر٢ ، وجمل التعليق يتم إهمالها أثناء ترجمة البرنامج وتنفيذه فهي جملة غير تنفيذية.

وتستخدم جمل التعليق لشرح البرنامج وتوثيقة داخلياً وكذلك للتعريف بوظيفة كل جزء وهي تسهل قراءة البرنامج وتعطي فكرة عن وظيفة كل جزء فيه عند كتابتها. وجمل التعليق قد تأتي في سطر واحد فقط أو جزء من سطر وفي هذه الحالة يجب أن تسبق بـ // أما إذا زادت جملة التعليق عن سطر فإنه في هذه الحالة يتم استخدام delimiter / * بحيث تبدأ بها الجمل وتنتهي بـ * / delimiter.

شرح البرنامج

وكل الجمل بين /*..... */ يتم إهمالها بواسطة المترجم Compiler، وجمل التعليق تفيد المبرمج في أنها تتيح له الفرصة لإضافة أي شرح لأي جزء من أجزاء البرنامج، ويمكن كتابة جمل التعليق بين العلامتين و / * و **/ وفي هذه الحالة يمكن استخدام خاصية من خصائص البرمجة بلغة الجافا وهي Javadoc لكي تقوم بقراءة البرنامج وتجميع كل التعليقات الموجودة فيه لعمل توثيق كامل للبرنامج ولكننا لن نتعرض لهذه الخاصية لأنها خارج نطاق هذه الحقيبة.

السطر الثاني **// A first program in Java**

جملة تعليق ثانيه تبين الغرض من البرنامج .

السطر الثالث سطر فارغ – المبرمج يستخدم الأسطر الفارغة والفراغات البينية لكي يُسهل قراءة البرنامج، والأسطر الفارغة والمسافات الفارغة تُهمل بواسطة المترجم ويمكن استخدامها وقتما يشاء المبرمج.

شرح البرنامج

السطر الرابع

public class Welcome1 {

وهو يبدأ بتعريف الكائن class وإعطائه اسم (identifier). كل برنامج بلغة جافا يحتوي على الأقل على تعريف لكائن واحد يقوم المبرمج بتعريفه. وهذه الكائنات هي الكائنات المعرفة عن طريق المستخدم User defined classes.

والكلمة class تقوم بتعريف الكائن ويتبعها اسم هذا الكائن وهو Welcome1 (في هذا البرنامج). والكلمة class من الكلمات المحجوزة في اللغة التي لها استخدامات خاصة ولذلك لا تصلح لأن تستخدم كاسم معرف (identifier).

وعند كتابة أسماء الكائنات يفضل أن يبدأ الحرف الأول في اسم class بحرف كبير مثل Welcome1. وكذلك إذا كان يتكون من أكثر من اسم فإن كل اسم يبدأ بحرف كبير مثل SampleClassName واسم الكائن يعرف بالاسم المعرف identifier .

شرح البرنامج

السطر السابع

public static void main(String args[ ])

يمثل جزءاً من كل تطبيق جافا (Java Application) حيث يبدأ تنفيذ البرنامج من الـ main، والأقواس بعد الـ main توضح أن الـ main هو أحد المقاطع الرئيسة (block) في بناء التطبيق ويسمى method(الطريقة). وكل كائن (class) يجب أن يحتوي على الأقل على طريقة (method) واحدة وقد يحتوي على أكثر من طريقة ويجب أن تكون واحدة من هذه الطرق على الأقل تسمى main ويجب أن تعرف كما في السطر السابع. وفي حالة عدم وجود الmain فإنه لن يتم تنفيذ أي جزء من أجزاء البرنامج. والطرق (methods) تقوم بمعالجة البيانات وأداء بعض العمليات وبالتالي ينتج عنها بعض البيانات أو الخرج عند اكتمال تنفيذها.

والكلمة المحجوزة void تبين أن (الطريقة) method سوف تقوم بأداء عملية ما مثل (طباعة سطر - حساب مضروب عدد ما – حساب المتوسط الحسابي ..... الخ)

السطر التاسع

System.out.println("Welcome to Java Programming!");

يخبر الكمبيوتر بطباعة الجملة Welcome to Java Programming! والموجودة بين علامات التنصيص " ". والجملة بين علامات التنصيص تسمى String والمسافات الفارغة في String تهمل بواسطة المترجم .

الجملة System.out تعرف بأنها جملة الخرج القياسية Standard Output Object، .وهذه الجملة تقوم بإظهار الجمل النصية وكذلك أي معلومات أو بيانات في نافذة الأوامر حيث يتم تنفيذ برامج الجافا. والـ Method المسماة System.out.println في هذا البرنامج تظهر النص في سطر واحد في نافذة الأوامر Command Window وعندما تنتهي الطباعة فإن المؤشر يوضع في بداية السطر التالي، وهذا يماثل ضغط مفتاح Enter في لوحة المفاتيح عند الكتابة.

وفي نهاية السطر وضعت الفاصلة المنقوطة ; وهذا يعني أن جملة جافا (Java Statement) قد انتهت. وكل جملة من جمل الجافا يجب أن تنتهي بفاصلة منقوطة. والفاصلة المنقوطة تحدد نهاية الجملةStatement Terminal.

الترجمة و التنفيذ

١ - غير الفهرس إلى الفهرس الذي تم حفظ البرنامج فيه ثم اكتب الأمر التالي

Javac Welcome1.Java

إذا كان البرنامج يحتوي على أخطاء بنائية (Syntax Errors) فإن هذه الأخطاء سوف تظهر في نافذة الأوامر موضحاً فيها رقم السطر ومكان الخطأ وتفسير محتمل للخطأ. وفي هذه الحالة يجب تصحيح هذه الأخطاء في البرنامج ثم إعادة هذه الخطوة السابقة ثانية ويتم تكرارها حتى يصبح البرنامج بدون أخطاء ويعطي العبارة التالية No Errors . وفي هذه الحالة فإن المفسر يقوم بإنشاء وحفظ ملف جديد يسمى Welcome1.class يحتوي على الـ Byte code. هذا الملف ينتج من ترجمة جمل لغة الجافا بواسطة المترجم إلى byte code وهي صورة أخرى للبرنامج وهي الصورة التنفيذية للبرنامج. و لتنفيذ البرنامج من خلال نافذة الأوامر نكتب الأمر java Welcome1 في نافذة الأوامر. وإذا لم يتوفر الملف ذو الامتداد class. فإن المفسر لا يستطيع تنفيذ البرنامج ويعطي رسالة خطأ.

وتنفيذ البرنامج يبدأ من main method ثم ينتقل إلى الجمل التنفيذية فيه والتي تقوم بإظهار الجملة بين علامتي التنصيص. وعند تنفيذ البرنامج فإن المفسر يقوم بتنفيذ Byte Code الناتج من عملية الترجمة والموجود في الملف ذي الامتداد class.

مؤثرات الطباعة

الحرف الخاص	الوصف
\n	سطر جديد. يضع المؤشر في بداية السطر التالي
\t	مسافة أفقية. تحريك المؤشر مسافة معينة إلى النقطة التالية في السطر
\r	carriage return. يضع المؤشر في بداية السطر الحالي ولا يتقدم إلى السطر التالي ، وأي حرف يطبع يتم طباعته على حرف سابق تم كتابته في نفس السطر
\\	شرطة خلفية. إظهار " \ " في الخرج
\"	علامة تنصيص مزدوجة. إظهار علامة التنصيص المزدوجة

مثال

```
1.    // Fig. 2-4: Welcome3.java
2.    // Printing multiple lines with a single statement.
3.
4. public class Welcome3 {
5.
6.    // main method begins execution of Java application
7.    public static void main( String args[ ] )
8.
9.    {
10.   System.out.println( "Welcome\nto\nJava\nProgramming!" );
11.   } // end method main
12.   } // end class Welcome3
```

النتيجة

```
Welcome
to
Java
Programming!
```

العمليات في لغة Java

- العمليات الرياضية
- العمليات المنطقية
- عمليات المقارنة
- عمليات التخصيص أو العمليات الإسنادية

العمليات الإسنادية Assignments

تستخدم هذه العملية لتخصيص قيمة ما في متغير وذلك بعد تعريفه ، ونستخدم العملية = للتعبير عن التخصيص في لغة الجافا :

أمثلة:

```
x = 1 ;
```

تم تخصيص 1 إلى المتغير x

```
radius = 1.0 ;
```

تم تخصيص القيمة 1.5 إلى المتغير x

```
a = 'A' ;
```

تم تخصيص الحرف 'A' إلى المتغير a

خطأ شائع : تخصيص قيمة من نوع بيانات مختلف عن نوع المتغير ، فمثلا إذا كان x=1.0 يمكن أن تعطي خطأ وذلك إذا كان x معرف على أنه قيمة صحيحة int ، لذلك يجب أن يكوناً معرف على أنه double أي يقبل الكسور .

أيضا لاحظ أن اسم المتغير يجب أن يكون على اليسار بمعنى أن الجملة 1 = x تعتبر خطأ .

الإختصارات في العمليات الإسنادية

العامل	مثال	ما يقابله دون اختصار
+ =	c + = 7	c = c + 7
- =	d - = 4	d = d - 4
* =	e * = 5	e = e * 5
/ =	f = f / 3	f = f / 3
% =	g % = 9	g = g % 9

أختصارات الزيادة و النقصان

العامل	مثال	توضيح
+ +	+ + a	يتم زيادة المتغير a بمقدار 1 ثم تستخدم القيمة الجديدة للمتغير a في التعبير المتواجد فيه
+ +	+ + a	تستخدم القيمة الأولى للمتغير a في التعبير المتواجد فيه ثم بعد ذلك يتم زيادة المتغير a بمقدار 1
--	--b	يتم إنقاص المتغير b بمقدار 1 ثم تستخدم القيمة الجديدة للمتغير b في التعبير المتواجد فيه
--	b--	تستخدم القيمة الأولى للمتغير a في التعبير المتواجد فيه ثم يتم إنقاص المتغير b بمقدار ١

مثال

```
// Preincrementing and postincrementing


public class Increment {
public static void main( String args[] )
{
int c;

c = 5;
System.out.println( c );   // print 5
System.out.println( c++ ); // print 5 then postincrement
System.out.println( c );   // print 6

System.out.println();      // skip a line

c = 5;
System.out.println( c );   // print 5
System.out.println( ++c ); // preincrement then print 6
System.out.println( c );   // print 6
}
}
```

العمليات الرياضية

العملية	التعبير بالجافا
جمع	f+7
طرح	f-7
ضرب	b*m
قسمة	x/y
موديلاس	r%s

الأفضلية

- الأقواس
- الضرب و القسمة
- الجمع و الطرح
- الضرب المنطقي
- الجمع المنطقي

y = a * x * x + b * x + c
6 1 2 4 3 5

المعاملات المنطقية

العامل	العملية
&&	And المنطقية
||	Or المنطقية
!	Not المنطقية

مثال

```
// Logical Operator
public class Logical
{
public static void main ( String [] args )
{
//declare & initialize test variable
boolean a  = true , b = false ;
boolean c1 =(a && a);      // test if both are true
boolean c2 =(a && b);
boolean c3 =(b && b);

boolean c4 =(a || a) ;       //test if either is true
boolean c5 =(a || b);
boolean c6 =(b || b);

boolean c7 = !a;             // invert initial values
boolean c8 = !b:

// display the results
System.out.println (“and:\n1:”+c1+”2:”+c2+”3:”+c3);
System.out.println (“or:\n4:”+c4+”5:”+c5+”6:”+c6);
System.out.println (“not:\n7:”+c7+”8:”+c8);
}
}
```

النتيجة و شرح البرنامج

```
and:
1:true2:false3:false
or:
4:true5:true6:false
not:
7:false8:true
```

السطر رقم 8 تم تعريف متغيرين من النوع boolean هما a ، b وإعطائهما قيم ابتدائية هي true، false على الترتيب.

السطور 9,10,11

تم تعريف المتغيرات c1,c2,c3 وذلك لمناقشة الحالات المختلفة للعملية &&

السطور 13,14,15

تم تعريف المتغيرات c4,c5,c6 وذلك لمناقشة الحالات المختلفة للعملية ||

السطور 17,18

تم تعريف المتغيرين c7,c8 وذلك لمناقشة الحالات المختلفة للعملية !

السطور 21,22,23

هي جمل لطباعة المتغيرات c1,c2,c3,c4,c5,c6,c7,c8.

عمليات المقارنة

شكل عمليات التساوي في الجبر	شكل عمليات التساوي في الجافا	مثال على الشرط في الجافا	معنى الشرط
=	==	x==y	x تساوي y
≠	!=	x!=y	x لا تساوي y

شكل العمليات العلاقية في الجبر	شكل العمليات العلاقية في الجافا	مثال على الشرط في الجافا	معنى الشرط
>	>	x>y	x أكبر من y
<	<	x<y	x أقل من y
≥	>=	x>=y	x أكبر من أو تساوي y
≤	<=	x<=y	x أقل من أو تساوي y

خطأ شائع:

عند كتابة العمليات =! ، == ، = < ، => ، وبينهما مسافة مثل == ، <= ،=> ، =! يعطي البرنامج syntax error.

أيضا عند عكس رموز العملية الواحدة مثل =< ، => ، != يعطي أيضا syntax error

Control Commands

DR. FADI TIRKAWI

- في معظم لغات البرمجة الإجرائية يحتل مصطلح وقت الحياة أو lifetime جزء كبير من وقت و جهد البرمجة.
- كم يستمر المتحول متوفر للتعامل معه.

The Scope المجال

- في حال عرفنا متحول داخل مجال فإنه يستمر حتى نهاية هذا المجال
- لتسهيل التعامل مع المجال عند كتابة الكود نقوم بالإزاحة

```
{
int x = 12;
 // Only x available
        {
        int q = 96;
        // Both x & q available
        }
// Only x available
// q “out of scope”
}
```

خطأ شائع

```
{
int x = 12;
        {
        int x = 96; // Illegal
        }
}
```

- الخطأ هو .....

garbage collector

```
{
Car s = new Car();
} // End of scope
```

- Java has a *garbage collector*, which looks at all the objects that were created with **new** and figures out which ones are not being referenced anymore. Then it releases the memory for those objects, so the memory can be used for new objects. This means that you never need to worry about reclaiming memory yourself. You simply create objects, and when you no longer need them, they will go away by themselves.

Creating new data types: class

- class ATypeName { /* Class body goes here */ }
- you can create an object of this type using **new**:
- ATypeName a = new ATypeName();

Fields and methods

- you can put two types of elements in your class:
  - *fields* (sometimes called data members)
  - *methods* (sometimes called *member functions*).

```
class DataOnly {
    int i;
   float f;
   boolean b;
}

DataOnly d = new DataOnly();
```

You can assign values to the fields

For example:

```
d.i = 47;
d.f = 1.1f; // 'f' after number indicates float constant
d.b = false;
```

The **DataOnly** class cannot do much of anything except hold data, because it has no methods.

Methods, arguments, and return values

- returnType methodName(/* Argument list */) { /* Method body */ }

IF

عادة يتم تنفيذ الجمل في البرامج بطريقة تتابعيه جملة بعد أخرى ولكن عند استخدام أداة من أدوات التحكم مثل if لا تنفذ الجملة إلا إذا تحقق الشرط السابق.

- في الفصل السابق درسنا شكلاً من أشكال اتخاذ القرار عن طريق استخدام جملة if في أبسط صورها وقلنا

```
If (number1 == number2 )
result = result+ " number1 == number2 " ;
```

جملة if السابقة تسمى if selection structure حيث إنها تقوم بانتخاب الجملة التي يتم تنفيذها

فمثلا : افرض أن درجة اجتياز أي اختبار هي 60 من 100 تكتب جملة if على النحو

```
If (studentGrade > = 60 )
System.out.println (" passed" );
```

وبذلك نرى أن كلمة (" passed ") لن يتم طباعتها على الشاشة إلا إذا كانت درجة الطالب في الاختبار أكبر من أو تساوي 60

If / else

يمكننا استخدام جملة if \ else إذا أردنا

- عند تحقق الشرط تنفذ جملة ما ثم ينفذ باقي البرنامج
- عند عدم تحقق الشرط تنفذ جملة أخرى ثم بعدها ينفذ باقي البرنامج

Switch

في حالات الاختيار من متعدد يمكن أن تحل جملة switch محل if/else المتعددة

تكتب جملة switch على الشكل التالي:

```
switch (switch-expression)
{ case value1: statement(s)1;break;
  case value2: statement(s)2;break;
  …
  case valueN: statement(s)N;break;
        default :        statement(s)- for – default;
}
```

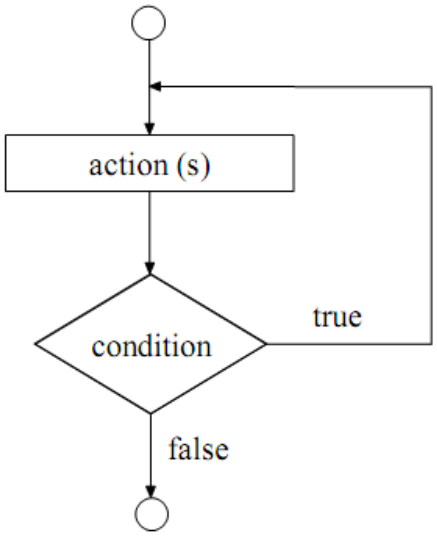
while

تسمح الجمل التكرارية للمبرمج أن يعرف جملة ما أو عدة جمل أن يحدث لها تكرار طالما أن الشرط صحيح

Do / While

تستخدم حلقة do–while كسابقتها while لعمل تكرار لجملة أو عدة جمل ويكون التركيب البنائي لها على الشكل :

```
Do
{
//      الجمل المراد تكرارها
} while ( شرط استمرار الحلقة ) ;
```



for

تستخدم أيضا هذه الجملة لعمل تكرار لجملة أو عدة جمل ويكون التركيب البنائى لها على الشكل :

```
for ( إعطاء قيمة ابتدائية للعداد  ;  شرط استمرار الحلقة  ;  جملة زيادة أو نقصان العداد )
    {
        //الجملة المراد تكرارها
    }
```

Break

تستعمل هذه الجمل عندما يراد تغيير المسار الطبيعي للبرنامج فمثلا عندما تستخدم جملة break داخل بناء جملة switch , do/while , for , while تسبب الخروج منها فورا ويستمر تنفيذ باقي جمل البرنامج التي تلي بناء الجملة والاستخدام الشائع لجملة break هو للهروب مبكرا من تنفيذ حلقة أو لإهمال تنفيذ باقي جملة switch .

static keyword

- عندما نصنع صنف ما class فإننا فقط نوضح كيف يبدو و كيف يتصرف الغرض من هذا النوع من هذا الصنف.
- و فعلياً لا يمكن فعل أي شئ قبل خلق غرض من هذا الصنف باستخدام new. و في هذه اللحظة يتم حجز الذاكرة للغرض. و عندها تصبح الميزات attributes and methods و الطرق متاحة
- و لكن هناك حالات تكون هذه الطريقة غير كافية
- إحداها هي أن نملك تخزين لقطعة من المعطيات بغض النظر كم غرض تم خلقه. أو حتى في حالة لم يتم خلق أي غرض. أو في حالة أردنا طريقة method لا غير مرتبطة بأي غرض.

```
class StaticTest {
  static int i = 47;
}
```

- Now even if you make two **StaticTest** objects, there will still be only one piece of storage for **StaticTest.i.** Both objects will share the same **i**. Consider:

```
StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();
```

- At this point, both **st1.i** and **st2.i** have the same value of 47 since they refer to the same piece of memory.
- There are two ways to refer to a **static** variable. As the preceeding example indicates, you can name it via an object, by saying, for example, **st2.i**. You can also refer to it directly through its class name, something you cannot do with a non-static member.

```
StaticTest.i++;
```

Static function

```
class StaticFun {
  static void incr() { StaticTest.i++; }
}
```

```
StaticFun sf = new StaticFun();
sf.incr();
```

```
StaticFun.incr();
```

التحميل الزائد & الباني
Constructors & Methods Overloading

DR. FADI TIRKAWI

الهدف من الباني

- يمكنك أن تتخيل بأنك تصنع تابع تسميه initialize لكل صنف تقوم بكتابته. لذلك تم وضع شئ اسمه الباني من أجل ضمان تهيئة العناصر التي تخص صنف ما. و هو ما يدعى بـ Constructor
- إن لغة C# أو لغة الجافا تستدعى الباني بشكل أتوماتيكي عندما يتم خلق غرض. و بهذا يتم ضمان بأن الغرض تم تهيئته
- لتسهيل اختيار اسم مناسب للباني فإنه يتم استخدام نفس اسم الصنف او الـ class الشريحة التالية فيها مثال يوضح ذلك.

Constructor

C#

```
public class Taxi
{
    public bool isInitialized;
    public Taxi()
    {
        isInitialized = true;
    }
}

class TestTaxi
{
    static void Main()
    {
        Taxi t = new Taxi();
        Console.WriteLine(t.isInitialized);
    }
}
```

```
// Demonstration of a simple constructor.

class Rock {
  Rock() { // This is the constructor
    Console.writeLine("Creating Rock");
  }
}

public class SimpleConstructor {
static void Main(  ) {
    for(int i = 0; i < 10; i++)
      new Rock();
}
} ///:~
```

خرج البرنامج السابق

"Creating Rock",
"Creating Rock",
"Creating Rock",
"Creating Rock",
"Creating Rock",
"Creating Rock",
"Creating Rock",
"Creating Rock",
"Creating Rock",
"Creating Rock"

مثال ٢ عن الباني

يبين هذا المثال بأن الباني قد يحتوي على دخل أو على بارامیتر دخول

```
class Rock2 {
  Rock2(int i) {
 Console.writeLine("Creating Rock number " + i);
  }
}

public class SimpleConstructor2 {
static void Main(  ) {
    for(int i = 0; i < 10; i++)
      new Rock2(i);
}
} ///:~
```

"Creating Rock number 0",
"Creating Rock number 1",
"Creating Rock number 2",
"Creating Rock number 3",
"Creating Rock number 4",
"Creating Rock number 5",
"Creating Rock number 6",
"Creating Rock number 7",
"Creating Rock number 8",
"Creating Rock number 9"

التحميل الزائد لتوابع الباني

و التحميل الزائد لتابع عادي

- إحدى الميزات الهامة للغات البرمجة و هي التحميل الزائد mothods overloading و تعني هذه الميزة بأنك قد تحمل نفس كلمة ما العديد من المعاني وهذا مفيد عندما تأتي بصيغ مختلفة مثل:
- Wash a car
- Wash a shirt
- Wash a dog
- فمن السئ بأن تقول برمجياً
- Washcar()
- Washshirt()
- Washdog()
- لكن الأفضل برمجياً هو إستخدام
- Wash (car)
- Wash (shirt)
- Wash (dog)
- *تعريف تحميل الطرق أو التوابع هو اساسي ليسمح لطريقة بنفس الأسم بأن تستخدم مع متحولات ادخال بطرق و انواع مختلفة.*

مثال

```
class Tree {
  int height;
  Tree() {
Console.writeLine("Planting a seedling");
    height = 0;
  }
  Tree(int i) {
 Console.writeLine("Creating new Tree that is “     + i + " feet tall");
    height = i;
  }
  void info() {
 Console.writeLine("Tree is " + height + " feet tall");
  }
  void info(String s) {
Console.writeLine(s + ": Tree is “   + height + " feet tall");
  }
}
static void Main( ) {
    for(int i = 0; i < 5; i++) {
      Tree t = new Tree(i);
      t.info();
      t.info("overloaded method");
    }
Tree  z= new Tree(); // Overloaded constructor:
}
```

خرج المثال السابق

"Creating new Tree that is 0 feet tall",
"Tree is 0 feet tall",
"overloaded method: Tree is 0 feet tall",
"Creating new Tree that is 1 feet tall",
"Tree is 1 feet tall",
"overloaded method: Tree is 1 feet tall",
"Creating new Tree that is 2 feet tall",
"Tree is 2 feet tall",
"overloaded method: Tree is 2 feet tall",
"Creating new Tree that is 3 feet tall",
"Tree is 3 feet tall",
"overloaded method: Tree is 3 feet tall",
"Creating new Tree that is 4 feet tall",
"Tree is 4 feet tall",
"overloaded method: Tree is 4 feet tall",
"Planting a seedling"

Default constructors

```
class Bird { int i; }
static void Main( )
 {
Bird nc = new Bird(); // Default! }
```

- إن السطر الذي يحوي new Bird() يخلق غرض جديد و يستدعى الباني الإفتراضي على الرغم أن لم يتم تعريفه بشكل صريح.
- The line new Bird();
  creates a new object and calls the default constructor, even though one was not explicitly defined.

- However, if you define any constructors (with or without arguments), the compiler will *not* synthesize one for you:

```
class Hat {
    int y;
    double z;
    Hat(int i) {y = i;}
    Hat(double d) {z = d;}
    Hat(String x) {
     }
}
```

- في حال كتبت التعليمات الأتية ماذا سيحصل
- new Hat(1); يستدعى الباني الأول
- new Hat(1.6); يستدعى الباني الثاني

```
new Hat();
```

- المترجم سوف يظهر خطأ في الترجمة لأنه لم يجد باني يطابق هذا الباني الذي قمت بإستدعائه
- the compiler will complain that it cannot find a constructor that matches.

- It’s as if when you don’t put in any constructors, the compiler says “You are bound to need *some* constructor, so let me make one for you.” But if you write a constructor, the compiler says “You’ve written a constructor so you know what you’re doing; if you didn’t put in a default it’s because you meant to leave it out.”
  - في حال لم تكتب أي باني إفتراضي فإن المترجم يقول «أنك تحتاج باني و سنقوم بصنعه لك».
  - لـــــــــــــــكـــــــــــــــن إذا كتبت باني محدد فإن المترجم سيقول «لقد كتبت باني لذلك أنت تعلم ماذا تعمل ،و أنت لم تضعه لأنك بشكل متقصد لا تريد باني إفتراضي»

Member initialization

تهيئة الأعضاء

```
void f() {
  int i;
  i++; // Error -- i not initialized
}
```

Fields in a class

- If a primitive is a field in a class, however, things are a bit different. Since any method can initialize or use that data, it might not be practical to force the user to initialize it to its appropriate value before the data is used.
  - من أجل المتحولات الأولية للحقول في صف ما فإن الأمور تختلف قليلا. ليس من الإجبار بأن يقوم المستخدم بإعطاء قيم تهيئة للمتحولات أو الحقول ضمن الصف و في حال لم تتم إعطاء قيم تهيئة فإن المترجم يقوم بتهيئة العناصر الأولية ضمن الصف.

مثال عن تهيئة المتحولات الأولية بشكل افتراضي

```
public class InitialValues {
bool t;
 char c;
 byte b;
 short s;
 int i;
 long l;
 float f;
 double d;
 void print(String s) {Console.writeLine(s); }
 void printInitialValues() {
  print("Data type      Initial value");
  print("boolean       " + t);
  print("char          [" + c + "]");
  print("byte          " + b);
  print("short         " + s);
  print("int           " + i);
  print("long          " + l);
  print("float         " + f);
  print("double        " + d);
 }}
static void Main( ) {
  InitialValues iv = new InitialValues();
  iv.printInitialValues();

} ///:~
```

نتيجة تنفيذ البرنامج

```
"Data type      Initial value",
"boolean       false",
"char          [" + (char)0 + "]",
"byte          0",
"short         0",
"int           0",
"long          0",
"float         0.0",
"double        0.0"
```

Inheritance
الوراثة

Dr. Fadi Tirkawi

Class Reusing
استخدام الكلاسات الأخرى بطريقتي الدمج و الوراثة

- نعود إلى نقطة هامة في البرمجة غرضية التوجه و هي إعادة إستخدام الكود Code Reuse.و هي من الميزات الهامة الأساسية التي تميز البرمجة الغرضية التوجه عن البرمجة الأجرائية مثل لغة C.
تحت هذه النقطة يمكن ان نذكر شيئين أساسيين هما :
  - الجمع Composition
  - و الثاني هو الوراثة .Inheritance (Is a) الدمج أو الجمع Compositoin و هو يعني أنك تخلق و تستخدم غرض لنوع او صنف موجود ضمن صنفك الجديد. كجزء منه أو مملوك له (.has a / part of)
- نبقى عند نقطة Code Reuseو عند النقطة الأساسية الثانية و هي الوراثة Inheritance.الهدف هنا ببساطة هو الأستفادة من صنف موجود بما فيه من خصائص و توابع. التابع الذي يرث تابع اخر فيجب ان يكون من نفس النوع. بمعنى أخر نرث مربع من شكل هندسي لان المربع له مساحة قد يرثها من الشكل الهندسي.

- (الوراثة، التغليف ، تعدد الأشكال) تعد من المواصفات الأساسية الثلاثة (أو ركائز) للبرمجة الكائنية.
- تمكّنك الوراثة من إنشاء فئات جديدة تعيد استخدام وتوسّع وتعدّل السلوك الذي تم تعريفه في الفئات الأخرى. الفئة المورثة للأعضاء التي تسمى الفئة الأساسية و الفئة التي ترث هذه الأعضاء الفئة المشتقة.

- مفهومياً، الفئة المشتقة عبارة عن تخصيص من الفئة الأساسية. على سبيل المثال، إذا كان لديك فئة أساسية Animal، قد تكون هناك فئة مشتقة Mammalوفئة مشتقة أخرى .Reptile
- الـ Mammalهو Animal، و الـ Reptileهو Animal، ولكن كلاً من الفئات المشتقة تمثل تخصيص مختلف من الفئة الأساسية.

- عندما تقوم بتعريف فئة بأن تكون مشتقة من فئة أخرى، تكسب الفئة المشتقة ضمنياً كل أعضاء الفئة الأساسية باستثناء الدالة الإنشائية والدالة المدمّرة.
- وبالتالي يمكن للفئة المشتقة إعادة استخدام التعليمات البرمجية الموجودة في الفئة الأساسية دون الحاجة إلى إعادة كتابتها مره أخرى. يمكنك إضافة أعضاء أكثر في الفئة المشتقة. بهذه الطريقة، تكون الفئة المشتقة قد وسّعت وظيفة الفئة الأساسية.

Base and Derived Classes

```
using System;
namespace InheritanceApplication
{
   class Shape
   {
      public void setWidth(int w)
      {
         width = w;
      }
      public void setHeight(int h)
      {
         height = h;
      }
      protected int width;
      protected int height;
   }

   // Derived class
   class Rectangle: Shape
   {
      public int getArea()
      {
         return (width * height);
      }
   }
```

```
   class RectangleTester
   {
      static void Main(string[] args)
      {
         Rectangle Rect = new Rectangle();

         Rect.setWidth(5);
         Rect.setHeight(7);

         // Print the area of the object.
         Console.WriteLine("Total area: {0}",  Rect.getArea());
         Console.ReadKey();
      }
   }
}
```

```
Total area: 35
```

Initializing Base Class

- The derived class inherits the base class member variables and member methods. Therefore the super class object should be created before the subclass is created. You can give instructions for superclass initialization in the member initialization list.

```
using System;
namespace RectangleApplication
{
   class Rectangle
   {
      //member variables
      protected double length;
      protected double width;
      public Rectangle(double l, double w)
      {
         length = l;
         width = w;
      }

      public double GetArea()
      {
         return length * width;
      }

      public void Display()
      {
         Console.WriteLine("Length: {0}", length);
         Console.WriteLine("Width: {0}", width);
         Console.WriteLine("Area: {0}", GetArea());
      }
   }//end class Rectangle
```

```
   class Tabletop : Rectangle
   {
      private double cost;
      public Tabletop(double l, double w) : base(l, w)
      { }
      public double GetCost()
      {
         double cost;
         cost = GetArea() * 70;
         return cost;
      }
      public void Display()
      {
         base.Display();
         Console.WriteLine("Cost: {0}", GetCost());
      }
   }
   class ExecuteRectangle
   {
      static void Main(string[] args)
      {
         Tabletop t = new Tabletop(4.5, 7.5);
         t.Display();
         Console.ReadLine();
      }
   }
}
```

- When the above code is compiled and executed, it produces the following result:

```
Length: 4.5
Width: 7.5
Area: 33.75
Cost: 2362.5
```

Arrays

Arrays

- C# arrays are zero indexed; that is, the array indexes start at zero. Arrays in C# work similarly to how arrays work in most other popular languages There are

- When declaring an array, the square brackets ([]) must come after the type, not the identifier. Placing the brackets after the identifier is not legal syntax in C#.

- int[] table; // not int table[];

- Another detail is that the size of the array is not part of its type as it is in the C language.

```
int[] numbers; // declare numbers as an int array of any size
numbers = new int[10]; // numbers is a 10-element array
numbers = new int[20]; // now it's a 20-element array
```

- Single-dimensional arrays:
  - int[] numbers;
- Multidimensional arrays:
  - string[,] names;
- In C#, arrays are objects (discussed later in this tutorial) and must be instantiated. The following examples show how to create arrays:
- Single-dimensional arrays:
  - int[] numbers = new int[5];
- Multidimensional arrays:
  - string[,] names = new string[5,4]; 2d
  - int[,,] buttons = new int[4,5,3]; 3d
- Array-of-arrays
  - byte[][] scores = new byte[5][];
  - for (int x = 0; x < scores.Length; x++)
  - { scores[x] = new byte[4]; }

declares and instantiates arrays

```
// arrays.cs
using System;
class DeclareArraysSample
{
    public static void Main()
    {
        // Single-dimensional array
        int[] numbers = new int[5];

        // Multidimensional array
        string[,] names = new string[5,4];

        // Array-of-arrays (jagged array)
        byte[][] scores = new byte[5][];

        // Create the jagged array
        for (int i = 0; i < scores.Length; i++)
        {
            scores[i] = new byte[i+3];
        }

        // Print length of each row
        for (int i = 0; i < scores.Length; i++)
        {
            Console.WriteLine("Length of row {0} is {1}", i, scores[i].Length);
        }
    }
}
```

- **Output**
- Length of row 0 is 3
- Length of row 1 is 4
- Length of row 2 is 5
- Length of row 3 is 6
- Length of row 4 is 7

Initialize Single-Dimensional Array

  - int[] numbers = new int[5] {1, 2, 3, 4, 5};
  - string[] names = new string[3] {"Matt", "Joanne", "Robert"};
- You can omit the size of the array, like this:
  - int[] numbers = new int[] {1, 2, 3, 4, 5};
  - string[] names = new string[] {"Matt", "Joanne", "Robert"};
- You can also omit the **new** operator if an initializer is provided, like this:
  - int[] numbers = {1, 2, 3, 4, 5};
  - string[] names = {"Matt", "Joanne", "Robert"};

Initialize Multidimensional Array

  - int[,] numbers = new int[3, 2] { {1, 2}, {3, 4}, {5, 6} };
  - string[,] siblings = new string[2, 2] { {"Mike","Amy"}, {"Mary","Albert"} };
- You can omit the size of the array, like this:
  - int[,] numbers = new int[,] { {1, 2}, {3, 4}, {5, 6} };
  - string[,] siblings = new string[,] { {"Mike","Amy"}, {"Mary","Albert"} };
- You can also omit the **new** operator if an initializer is provided, like this:
  - int[,] numbers = { {1, 2}, {3, 4}, {5, 6} };
  - string[,] siblings = { {"Mike", "Amy"}, {"Mary", "Albert"} };

Accessing Array Members

  - int[] numbers = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0}; numbers[4] = 5;
- The following code declares a multidimensional array and assigns 5 to the member located at [1, 1]:
  - int[,] numbers = { {1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10} }; numbers[1, 1] = 5;
- The following is a declaration of a single-dimension array of array that contains two elements.
  - int[][] numbers = new int[][] { new int[] {1, 2}, new int[] {3, 4, 5} };
  - The following statements assign 58 to the first element of the first array and 667 to the second element of the second array:
  - numbers[0][0] = 58;
  - numbers[1][1] = 667;

تعددية الأشكال

د. فادي تركاوي

Polymorphism
تعددية الأشكال

- كلمة polymorphism تعني بأن نملك العديد من الأشكال أو تعددية الأشكال. في البرمجة الغرضية التوجه يمكن التعبير عن هذه الصفة كالآتي:
«واجهة مشتركة واحد لكن وظائف متعددة»
- تعددية الأشكال ممكن أن تكون ديناميكة أو ستاتيكية. تعددية الأشكال الستاتيكية تحدث خلال وقت الترجمة **static polymorphism** بينما تعددية الأشكال الديناميكية **dynamic polymorphism** تحصل خلال التنفيذ.

Polymorphism and Object Oriented Programming

- بشكل أساسي نستطيع الحصول من خلال هذه الميزة على سلوك و تصرف مختلف من أجل نوع ما. و هذا يمكننا من توسعة عمل التطبيق.

- Essentially we are able to get many different types of object behaviour from a single reference type
  - This enables us to write easily extensible applications

Static Polymorphism

- The mechanism of linking a function with an object during compile time is called early binding. It is also called static binding. C# provides two techniques to implement static polymorphism. They are:
  - Function overloading
  - Operator overloading
- We discuss operator overloading in next chapter.

Static Polymorphism مثال عن

- Printing int: 5
- Printing float
- Printing strin

```
namespace PolymorphismApplication
{
   class Printdata
   {
      void print(int i)
      {
         Console.WriteLine("Printing int: {0}", i );
      }

      void print(double f)
      {
         Console.WriteLine("Printing float: {0}" , f);
      }

      void print(string s)
      {
         Console.WriteLine("Printing string: {0}", s);
      }
      static void Main(string[] args)
      {
         Printdata p = new Printdata();

         // Call print to print integer
         p.print(5);

         // Call print to print float
         p.print(500.263);

         // Call print to print string
         p.print("Hello C++");
         Console.ReadKey();
      }
   }
}
```

Dynamic Polymorphism

```
using System;
namespace PolymorphismApplication
{
   class Shape
   {
      protected int width, height;
      public Shape( int a=0, int b=0)
      {
         width = a;
         height = b;
      }
      public virtual int area()
      {
         Console.WriteLine("Parent class area :");
         return 0;
      }
   }
   class Rectangle: Shape
   {
      public Rectangle( int a=0, int b=0): base(a, b)
      {

      }
      public override int area ()
      {
         Console.WriteLine("Rectangle class area :");
         return (width * height);
      }
   }
```

```
class Triangle: Shape
{
   public Triangle(int a = 0, int b = 0): base(a, b)
   {

   }
   public override int area()
   {
      Console.WriteLine("Triangle class area :");
      return (width * height / 2);
   }
}
class Caller
{
   public void CallArea(Shape sh)
   {
      int a;
      a = sh.area();
      Console.WriteLine("Area: {0}", a);
   }
}
class Tester
{

   static void Main(string[] args)
   {
      Caller c = new Caller();
      Rectangle r = new Rectangle(10, 7);
      Triangle t = new Triangle(10, 5);
      c.CallArea(r);
      c.CallArea(t);
      Console.ReadKey();
   }
}
```

- عند تنفيذ الكود في الشريحة السابقة يكون الناتج

- Rectangle class area:
- Area: 70
- Triangle class area:
- Area: 25

References and Inheritance
المراجع و الوراثة

- مرجع الغرض نستطيع من خلاله التأشير على الغرض من صنفه الأصلي أو لأي غرض مشتق (مورث) منه بواسطة الوراثة.
- بفرض لدينا صنفين Holiday و صنف مورث منه أسمه Christmas فنستطيع كتابة الآتي:

```
Holiday day;
day = new Holiday();
…
day = new Christmas();
```

Dynamic Binding
الربط ديناميكيا

- A polymorphic reference is one which can refer to different types of objects at different times. It morphs!
- The type of the actual instance, not the declared type, determines which method is invoked. (النوع الحقيقي للغرض هو يحدد أي طريقة سنستدعي)
- Polymorphic references are therefore resolved at *run-time*, not during compilation.
  - This is called ***dynamic binding.*** ***(خلال التنفيذ سيتم تحديد المرجع للغرض و ليس خلال الترجمة)***

Dynamic Binding
الربط ديناميكيا

- Suppose the `Holiday` class has a method called `Celebrate`, and the `Christmas` class redefines it (overrides it).
- Now consider the following invocation:

```
day.Celebrate();
```

- If `day` refers to a `Holiday` object, it invokes the `Holiday` version of `Celebrate`; if it refers to a `Christmas` object, it invokes the `Christmas` version

Overriding Methods
التغلب على الطرق

- C# requires that all class definitions communicate clearly their intentions.
- The keywords *virtual*, *override* and *new* provide this communication.
- If a base class method is going to be overridden it should be declared *virtual.*
- A derived class would then indicate that it indeed does override the method with the *override* keyword.

Overloading vs. Overriding

- Overloading deals with multiple methods in the same class with the same name but different signatures
- Overloading lets you define a similar operation in different ways for different data
- Example:

```
int foo(string[] bar);
int foo(int bar1, float a);
```

- Overriding عبارة عن طريقة موجودة في الصنف الأب و , الصنف الإبن و لها نفس البارمترات
- Overriding يجعلنا التركيب بتعريف نفس العمليات و لكن بطرق مختلفة من أجل انواع مختلفة من الغرض
- Example:

```
class Base {
   public virtual int foo() {} }
class Derived {
   public override int foo()
   {}}
```

Overloading vs. Overriding

- Overloading deals with multiple methods in the same class with the same name but different signatures
- Overloading lets you define a similar operation in different ways for different data
- Example:

```
int foo(string[] bar);
int foo(int bar1, float a);
```

- Overriding deals with two methods, one in a parent class and one in a child class, that have the same signature
- Overriding lets you define a similar operation in different ways for different object types
- Example:

```
class Base {
   public virtual int foo() {} }
class Derived {
   public override int foo() {}}
```

```
class Shape {
 void draw() {}
 void erase() {}
 }

class Circle extends Shape {
void draw() {
System.out.println("Circle.draw()");
}
void erase() {
System.out.println("Circle.erase()");
 }
 }

class  Square extends Shape {
void draw() {
System.out.println("Square.draw()");
 }
void erase() {
System.out.println("Square.erase()");
 }
 }
```

```
// A "factory" that randomly creates shapes:
Class RandomShapeGenerator {
private Random rand = new Random();
 public Shape next() {
switch(rand.nextInt(2)) {
default:
case 0: return new Circle();
 case 1: return new Square();
} }
 }
public class Shapes {
private static RandomShapeGenerator gen = new
RandomShapeGenerator();
 public static void main(String[] args) {
Shape[] s = new Shape[9];
 // Fill up the array with shapes:
for(int i = 0; i < s.length; i++)
 s[i] = gen.next();
// Make polymorphic method calls:
for(int i = 0; i < s.length; i++)
 s[i].draw();
} ///:~
```

Windows Programming Using C#

Forms Programming I

Contents

- System.Drawing Namespace
- System.Windows.Forms Namespace
  - Creating forms applications by hand
  - Creating forms applications using Visual Studio designer

Forms Programming

- النماذج مستخدمة من اجل خلق تطبيق نوافذ مستقل و تطبيقات واجهات رسومية
- إن API تم تطويره في تطبيقات .NET.
- و تدعم حاليا تطبيقات الرسوميات من خلال المكاتب:
  - System.Drawing
    - Basic GDI+ functionality
  - System.Windows.Forms
    - Higher-level controls

System.Drawing

- هذه المكتبة أعلاه تزود العديد من البنى التي تستخدم من اجل برمجة الواجهات الرسومية.
- و هي تزود دعم من اجل عمليات الرسم الأولية.
- ويمكنك من خلال هذه المكتبة رسم أي شكل تريده و ليس فقط ما تقدمه المكاتب الاساسية المتوفرة في المكاتب مثل الأزار و الجدول و النماذج

System.Drawing.Point

- هذه المكتبة تقوم بتشكيل و رسم نقاط ثنائية البعد.
- Structure which represents a 2-D point
- Constructor (الباني)
  - `Point(int x, int y)`
- Properties (الخصائص)
  - `X` – get/set of X coordinate البعد الافقي
  - `Y` – get/set of Y coordinate البعد العمودي

System.Drawing.Size

- تمثل هذه المكتبة توضيع الحجم من خلال الطول و العرض لشكل ما
- Constructor (الباني)
  - `Size(int width, int height)`
- Properties (الخصائص)
  - `Width` – get/set width (توضيع و الحصول على العرض)
  - `Height` – get/set height (توضيع و الحصول على الارتفاع)

System.Drawing.Rectangle

- عبارة عن مستطيل نعطيه الزاوية العليا اليسارية. و الطول و العرض.
- Constructor (البواني)
  - `Rectangle(Point tlc, Size sz)`
  - `Rectangle(int tlx, int tly, int wd, int ht)`
- Properties (الخصائص)
  - `X` – get/set top left X coordinate
  - `Y` – get/set top left Y coordinate
  - `Height` – get/set height
  - `Width` – get/set width
  - `Bottom` – get Y coordinate of rectangle bottom
  - `Top` – get Y coordinate of rectangle top
  - `Left` – get X coordinate of right of rectangle
  - `Right` – get X coordinate of left of rectangle

System.Drawing.Color

- عبارة عن كلاس يمثل نظام الألوان alpha-RGB
- Methods (الطرق المتوفرة فيه هي)
  - `Color FromARGB(int r, int g, int b)`
  - `Color FromARGB(int alpha, int r, int g, int b)`
- Properties (الخصائص)
  - `A` – get alpha value
  - `R` – get red value
  - `G` – get green value
  - `B` – get blue value
  - `Black, White, Red, Green, Yellow, Cyan, Coral, Blue`, etc. – get values of pre-defined colors

System.Drawing.Font

- عبارة عن كلاس او صنف يمثل الخط و الحجم و نوع الخط
- Constructor (الباني)
  - `Font(string family, int points, FontStyle style)`
- Properties (الخصائص)
  - `FontFamily` – get the FontFamily value
  - `Style` – get the FontStyle Value
  - `Size` – get the font size
  - `Bold` – get true if bold
  - `Italic` – get true if italic

System.Windows.Forms

- هذا المكتبة تحتوي على متحكمات (CONTRLOS) تستخدم لواجهات النوافذ.
- المتحكم هو عبارة عن غرض عالي المتسوى يتكون من
  - نافذة التي نرسم ضمنها الاشياء
  - اشياء يتم اضافتها لهذه النافذة
  - اشياء يتم قدحها من خلال احداث تتم اضافتها و يكن ان تحدث ضمن هذه النافذة

Form Class

- الصنف يتضمن المستوى الاعلى للنوافذ
- هذا الصنف يحتوي على المتحكمات مثل الازرار
- النموذج الذي نبرمجه يكون يرث النموذج الاساسي Form
- على الرغم من يحتوي على العديد من الطرق فإن اغلب الاحيان نحن نتخاطب معه من خلال خصائصه

Form Properties

Property	Description
Location	Point of to left corner
Size	Size of form in pixels
Text	Text displayed or caption
AutoScaleDimensions	DPI resolution of display it was built for. Will be scaled to look correct on other displays.
BackColor	Background color
ForeColor	Foreground or drawing color
ClientSize	Size of drawing area without borders or scrollbars
Controls	A collection of controls owned by the form
WindowState	Whether maximized, minimized or normal
DefaultSize	Size when initially created
MinimumSize	Minimum size window can be resized to
MaximumSize	Maximum size window can be resized to

Form Events أحداث النموذج

- النماذج تزود بعدد كبير من الاحداث
- يمكن ان تضيف واحد او اكثر من طرق قدح هذه الاحداث
- يمكن ربط هذه الاحداث مع العديد الاشياء التي مكن عملها ضمن النموذج مثل الضغط على زرين مختلفين يولد نفس الحدث

Form Events

Event	Description
Load	Just before form is loaded the first time
Closing	Just before the form is closed
Closed	When the form is actually closed
Shown	Occurs when a form is first displayed
ResizeBegin	Resize operation has begun
ResizeEnd	Resize operation has ended

Form Methods

Method	Description
Activate	Activates the window and gives it focus
Close	Closes the form
Show	Makes the form visible
BringToFront	Moves to top of stacking order
Hide	Makes the form invisible
Focus	Gives the form focus

Creating Windows Applications
خلق تطبيق نوافذ

- سوف نستعرض كيف يمكن بناء تطبيق واجهة رسومية باستخدام محرر نصوص

Creating Windows Applications
خلق تطبيقات نوافذ

- في خلق واجهة رسومية سوف نستخدم
- تطبيق وهو عبارة عن صنف يحتوي طرق ساكنة للتحكم بالتطبيق
  - عنوان هو عبارة عن نص ثابت
    - زر يستجيب لضغط فأرة

Creating Windows Applications
خلق تطبيقات نوافذ

- الخطوة الاولى هي خلق صنف اسمه GreetingForm
- يرث منم Form
- و نحدد الاشياء التي نريدها ضمنه و هو عبارة عن زر و عنوان

```
public class GreetingForm : Form {
        Label   greetingLabel;
        Button  cancelButton;
          …
}
```


Creating Windows Applications
خلق تطبيقات نوافذ

- نعرف العنوان و خصائصه

```
greetingLabel = new Label();
greetingLabel.Location = new Point(16, 24);
greetingLabel.Text = "Hello, World";
greetingLabel.Size = new Size(216, 24);
greetingLabel.ForeColor = Color.Black;
```

Creating Windows Applications
خلق تطبيقات نوافذ

- خلق الزر و خصائصه

```
cancelButton = new Button();
cancelButton.Location = new Point(150, 200);
cancelButton.Size = new Size(112, 32);
cancelButton.Text = "&Cancel";
cancelButton.Click += new
  EventHandler(cancelButton_Click);
```


Creating Windows Applications
خلق تطبيقات نوافذ

- نضع الخاصئص التي تخص النموذج الاساسي main Form

```
this.AutoScaleDimensions = new
  SizeF(95.0f, 95.0f);
this.ClientSize = new Size(300, 300);
this.Text = "Hello, World";
```

Creating Windows Applications
خلق تطبيقات نوافذ

- اضافة متحكمات للنموذج الرز و العنوان
- Add the controls to the form

```
this.Controls.Add(cancelButton);
this.Controls.Add(greetingLabel);
```

- و تزويد الزر بالحدث المتعلق به
- And provide the event handler

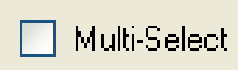
```
protected void cancelButton_Click(object sender, EventArgs e)
   {
         Application.Exit();
   }
```

- * see HelloForm

التصميم من خلال visual studio

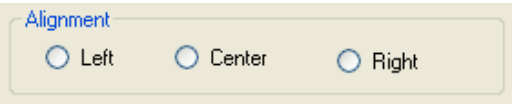
- و هو عبارة عن واجهة يتم من خلالها سحب و وضع الشكل الذي نريده مثل زر ، عنوان ، ...
- يتم توليد الكود هنا اتوماتيكيا
- يمكن ان نذهب فورا للحدث و نكتب له الكود
- يسرع ذلك كتابة الكود

CheckBoxes



- Labeled boxes which can be checked or unchecked
  - `Checked` – get/set Boolean to determine if box is checked
  - `CheckedChanged` – delegate called when the box is checked or unchecked
- * see ListBoxDemo

GroupBox

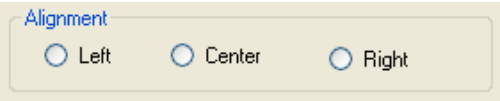


- Displays a border around a group of controls
- Can have optional label controlled by Text property
- Controls can be added by
  - Placing them within the group box in the designer
  - Adding to the Controls list programmatically
- * see TextBoxDemo

Panels

- A panel is like a group box but does not have a text label
- It contains a group of controls just like group box
  - `BorderStyle` – get/set border style as
    - `BorderStyle.Fixed3D`
    - `BorderStyle.FixedSingle`
    - `BorderStyle.None`

Radio Buttons



- Radio buttons are similar to checkboxes, but
  - Appear slightly different
  - Allow buttons to be grouped so that only one can be checked at a time
- A group is formed when the radio buttons are in the same container – usually a group box or panel

Radio Buttons

- `Checked` – get/set Boolean indicating if the button is checked
- `CheckedChanged` – delegate invoked when the button is checked or unchecked

- * see TextBoxDemo

TextBox

- This is a single line or multi-line text editor
  - `Multiline` – get/set Boolean to make multiline
  - `AcceptsReturn` – in a multiline box, if true then pressing Return will create a new line. If false then the button referenced by the `AcceptButton` property of the form, will be clicked.
  - `PasswordChar` – if this is set to a char, then the box becomes a password box

TextBox

- `ReadOnly` – if true, the control is grayed out and will not accept user input
- `ScrollBars` – determines which scrollbars will be used: ScrollBars.None, Vertical, Horizontal, Both
- `TextAlign` – get/set HorizontalAlignment.Left, Center, or Right
- `TextChanged` – event raised when the text is changed

File Dialog

- The file dialog allows you to navigate through directories and load or save files
- This is an abstract class and you use
  - `OpenFileDialog`
  - `SaveFileDialog`
- You should create the dialog once and reuse it so that it will remember the last directory the user had navigated to

File Dialog

- `InitialDirectory` – string representing the directory to start in
- `Filter` – a string indicating the different types of files to be displayed
  - A set of pairs of display name and pattern separated by vertical bars
    - `Windows Bitmap|*.bmp|JPEG|*.jpg|GIF|*.gif`
- `FilterIndex` – the filter to use as an origin 1 index

File Dialog

- FileName – the name of the file selected
- ShowDialog – a method to show the dialog and block until cancel or OK is clicked

```
if (openDialog.ShowDialog() == DialogResult.OK) {
    Image img = Image.FromFile(openDialog.FileName);
    pictureBox1.Image = img;
}
```

- * see ImageViewer

Image Class

- An abstract class that can store an image
- Several concrete classes are used for image types such as BMP, GIF, or JPG
  - `FromFile(string fname)` – loads any supported image format from a file
  - `FromStream(stream)` – loads an image from a stream
  - `Height` – image height
  - `Width` – image width
- *see ImageViewer

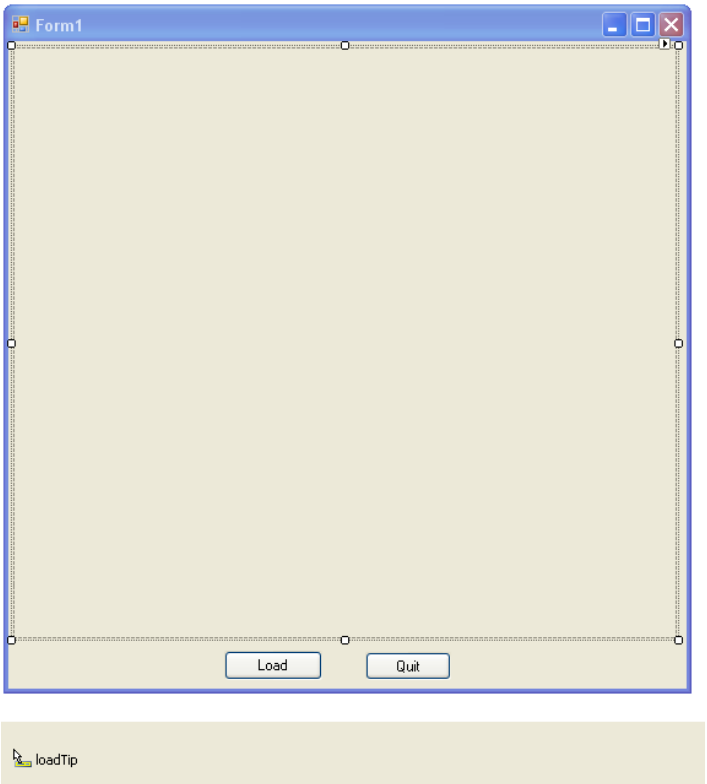
PictureBox Class

- This displays an image
  - `Image` – assigned an Image object to display
  - `SizeMode` – determines what to do if the image does not fit into the window
    - `Normal`
    - `StretchImage`
    - `AutoSize`
    - `CenterImage`
    - `Zoom`
- * see ImageViewer

ToolTips

- These are the small pop-up boxes which explain the purpose of a control
- To use
  - Create a new tooltip in the designer
  - Drop the tooltip onto the form
  - The tooltip will appear on a tray below the form
- * see ImageViewer

ToolTips



ToolTips

- After the tooltip appears in the tray, a new tooltip property appears for every component
- This can be assigned different text for each component
- That text will be displayed when the mouse hovers over that component

GUI Programming

- Windows Desktop Apps
  - Based on .NET framework
  - Windows Forms programming with C#
- Three key pieces:
  - Forms
  - Controls
  - Events

Building Forms from code

- Create class that inherits from `System.Windows.Forms.Form`
- Set form properties in constructor
- Pass form object to Application.Run to make it the start up form
- Use Show() or ShowDialog() to open secondary forms

Application Object

- System.Windows.Forms.Application
  - Singleton
- Application.Run starts message loop
- Application.Exit ends message loop
  - Closing the start-up form invokes Application.Exit
- Use properties to get information about running program

Control

- Interface for a form is built using controls
- Framework includes many useful controls
  - Button, TextBox, Labels, etc.
- Large third-party control market
- You can create your own custom controls
- Some controls are supplied by other framework
  - i.e. Ribbon control is in WPF but in .NET Framework

Controls

- Positioning controls on Form
  - Set Top & Left properties or
  - Set the Location Property (x,y)
- Sizing controls
  - Set the Width and/or Height properties or
  - Set the Size property

Add controls to a Form

- Create a private field for the control
- Create instance of control and set property values in form constructor
- Set control properties
- Add control instance to form's Controls collection

Events and Event Handlers

- Events are raised as user interacts with program
  - A button is clicked
  - A form is resized
- You can write a function that handles these events, called event handlers
- An event may have multiple event handlers
- An event handler may handle multiple

Dynamic Event Handler in C#

- Use += operator to attach an event handler
- Use -= operator to detach an event handler

Example:

```
 public MyForm()
 {
         ShowMessageButton.Click += ClickHandler;
….
 }

  private void ClickHandler(object send, EventArgs e)
  {
                      ….
  }
```