

Advance programming

البرمجة المتقدمة

First lecture: **Review**

Baida'a Lala'a

Course topics

- We will study in this course:
 - Design patterns
 - Web services
 - XML
 - Remoting objects
 - Event pattern
 - Client server programming

• سوف ندرس في هذه الدورة:

• أنماط التصميم

• خدمات الويب

• XML

• الكائنات البعيدة

• نمط الحدث

• برمجة خادم العميل

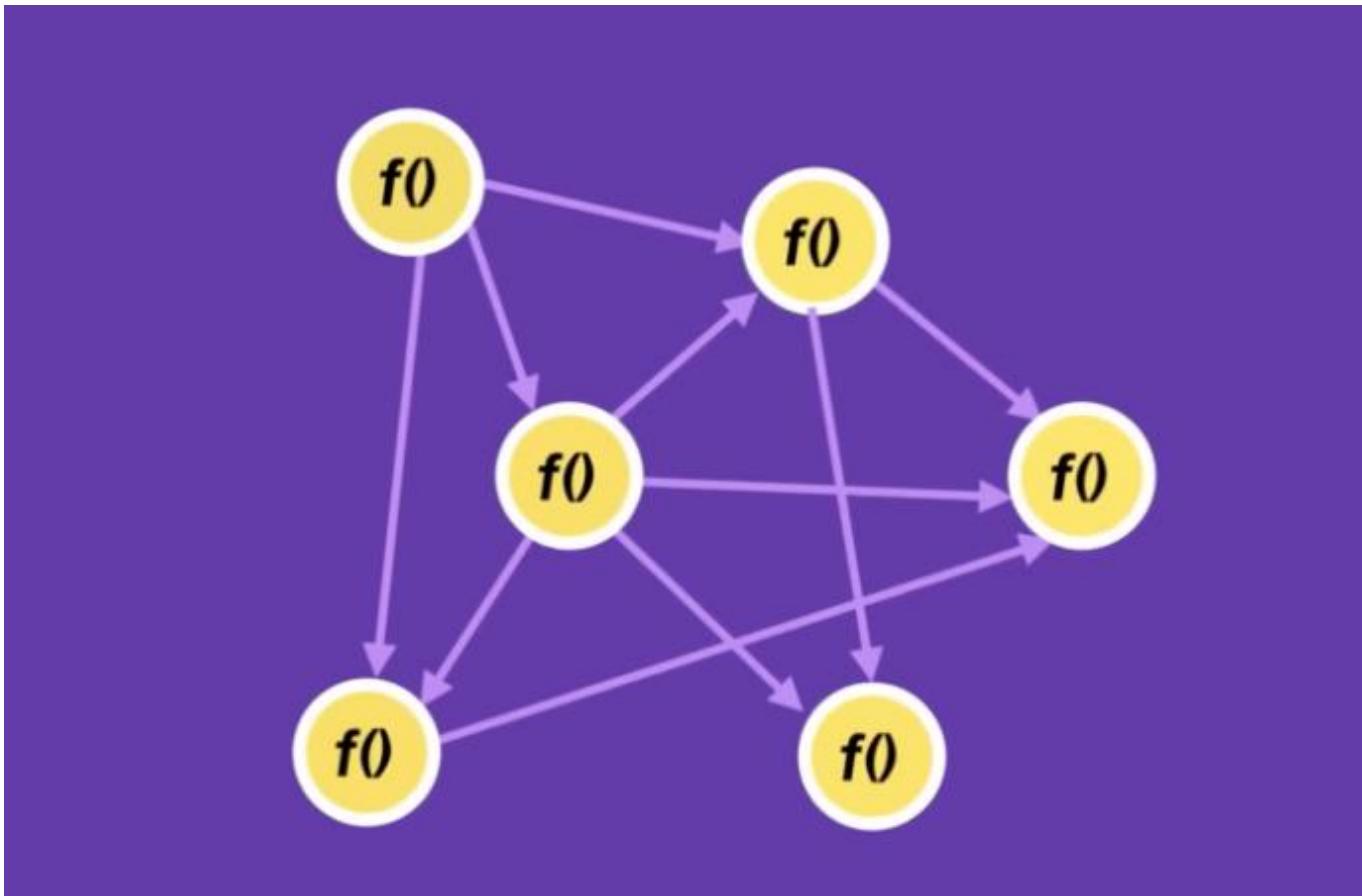
مراجع هذه المحاضرة References of This Lecture

- Starting Out with Python-Third edition--Tony Gaddis
- Introduction to Python Programming -Gowrishankar S. Veena A.
- البدء مع Python-Thirddition - TonyGaddis
- مقدمة إلى Python Programming-

نوع البرمجة Type of Programming

- **Procedural programming** is a method of writing software. It is a programming practice centered on the **procedures** or actions that take place in a program.
- **Object-oriented programming** is centered on **objects**. Objects are created from abstract data types that encapsulate data and functions together
- The object is, conceptually, a self-contained unit that consists of data attributes and methods that operate on the data attributes.
 - OOP is **a way** to **organize** and **conceptualize** a program as a set of **interacting** objects.
- إنها .البرمجة الإجرائية هي طريقة لكتابات البرمجية
- تركزت عملية البرمجة على الإجراءات أو الإجراءات التي
- خذ في البرنامج
- كائنات .البرمجة الشيئية مركزة على الأشياء •
- تم إنشاؤه من أنواع البيانات المجردة التي تغلف البيانات والوظائف
- سويا
- الكائن هو ، من الناحية المفاهيمية ، وحدة قائمة بذاتها تتكون من البيانات •
- السمات والطرق التي تعمل على سمات البيانات
- من التفاعل **programasa** وسيلة لتنظيم و تصور مجموعة OOPIs •
- شاء .

البرمجة الإجرائية Procedural programming



البرمجة الإجرائية Procedural programming

- In procedural programming, due to the fact that functions are linked together, the program might even break with just one change. This is what developers call spaghetti code.

- إجرائياً
- البرمجة ، بسبب
- حقيقة أن الوظائف
- مرتبطة ببعضها البعض
- البرنامج قد حتى
- قطع مع واحد فقط
- هذا هو ما يتغيرون
- المطورين يسمون السبايغيتى
- الشفرة.



التكلف وإخفاء البيانات

Encapsulation and Data hiding

OOP addresses the problem of code and data separation through encapsulation and data hiding.

Encapsulation refers to the combining of data and code into a single object.

Data hiding refers to an object's ability to hide its data attributes from code that is outside the object. Only the object's methods may directly access and make changes to the object's data attributes. An object typically hides its data, but allows outside code to access its methods. As shown in Figure 10-2, the object's methods provide programming statements outside the object with indirect access to the object's data attributes

معالجة مشكلة الكود وفصل البيانات من خلال الكبسلة OOP وتبين البيانات.

يشير التغليف إلى تجميع البيانات والتعليمات البرمجية في كائن واحد.

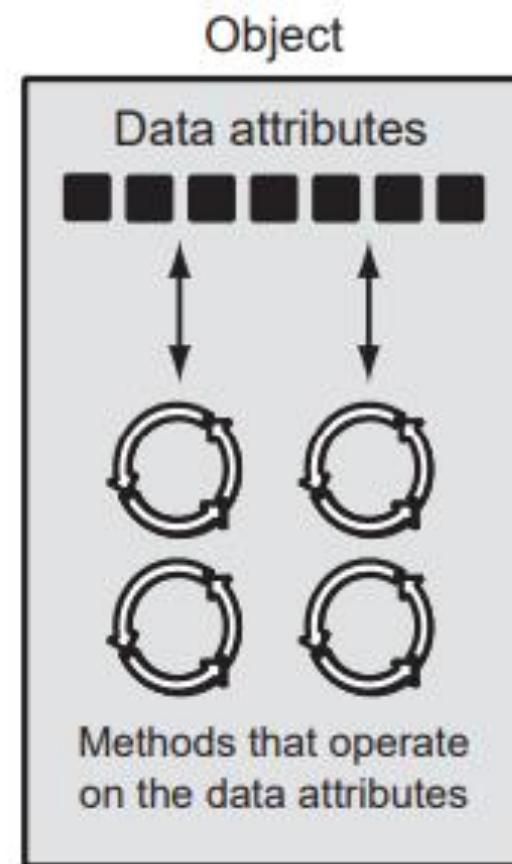
تشير البيانات إلى قدرة الكائن على إخفاء سمات بياناته من التعليمات البرمجية

فقط طرق الكائن هي التي يمكن الوصول إليها . خارج الكائن والقيام بها بشكل مباشر

تغيير سمات بيانات الكائن: الكائن عادةً يخفي البيانات ، ولكن كما هو موضح . يسمح لكود خارجي بالوصول إلى الأساليب في الشكل ٢-١٠ ، الكائن

الأساليب توفر البرمجة البيانات خارج الكائن مع الوصول غير المباشر إلى سمات بيانات الكائن

Figure 10-1 An object contains data attributes and methods

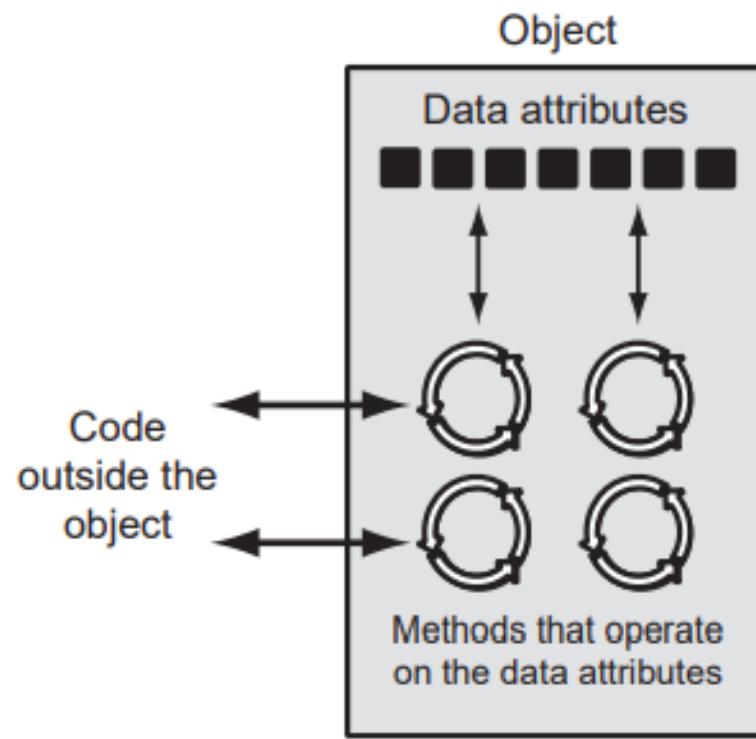


When an object's data attributes are hidden from outside code, and access to the data attributes is restricted to the object's methods, the data attributes are protected from accidental corruption. In addition, the code outside the object does not need to know about the format or internal structure of the object's data. The code only needs to interact with the object's methods. When a programmer changes the structure of an object's internal data attributes, he or she also modifies the object's methods so that they may properly operate on the data.

The way in which outside code interacts with the methods, however, does not change.

عندما يتم إخفاء سمات بيانات كائن ما من الرمز الخارجي ، ويتم الوصول إليها سمات البيانات مقيدة بأساليب الكائن ، سمات البيانات هي بالإضافة إلى ذلك ، يقوم محمي من الفساد العرضي الكائن لست بحاجة إلى معرفة التنسيق أو البنية الداخلية الـ.لبيانات الكائن عندما يكون يجب أن تتفاعل مع أساليب الكائن مبرمج تغيير سمات البيانات الداخلية للكائن ، هي أو هي أيضًا تعديل أساليب الكائن بحيث يمكن تشغيلها بشكل صحيح على البيانات. ومع ذلك ، فإن الطريقة التي تتفاعل فيها الشفرة مع هذه الأساليب لا يتغيرون.

Figure 10-2 Code outside the object interacts with the object's methods



إعادة استخدام الكائن Object Reusability

- In addition to solving the problems of code and data separation, the use of OOP has also been encouraged by the trend of object reusability. An object is not a stand-alone program, but is used by programs that need its services.

بالإضافة إلى حل مشاكل الكود وفصل البيانات ، فإن تشجيع من اتجاه OOP استخدام الكائن إعادة الاستخدام: الكائن ليس برنامجاً قائماً بذاته ، ولكن يتم استخدامه بواسطة البرامج التي تحتاج إلى خدماتها.

Example

Imagine that your alarm clock is actually a software object. If it were, it would have the following data attributes:

- current_second (a value in the range of 0–59)
 - current_minute (a value in the range of 0–59)
 - current_hour (a value in the range of 1–12)
 - alarm_time (a valid hour and minute)
 - alarm_is_set (True or False)
- إذا كان كذلك ، فإنه تخيل أن المنبه الخاص بك هو في الواقع كائن برمجي .
- سوف تمتلك سمات البيانات التالية :
 - Current_second (قيمة في نطاق ٠-٥٩)
 - Current_minute (قيمة في نطاق من ٠ إلى ٥٩)
 - الساعة الحالية (قيمة في نطاق من ١ إلى ١٢)
 - alarm_time (ساعة ودقيقة صالحة)
 - alarm_is_set (صواب خطأ)

Every second the increment_current_second method executes. This changes the value of the current_second data attribute.

If the current_second data attribute is set to 59 when this method executes, the method is programmed to reset current_second to 0, and then cause the increment_current_minute method to execute. This method adds 1 to the current_minute data attribute, unless it is set to 59. In that case, it resets current_minute to 0 and causes the increment_current_hour method to execute. The increment_current_minute method compares the new time to the alarm_time.

If the two times match and the alarm is turned on, the sound_alarm method is executed.

- هذا التغيير كل ثانية يتم تنفيذ طريقة التزاييد الحالى_ الثانية
- قيمة سمة البيانات الثانية الحالية
- إذا كانت سمة البيانات الحالية_ الثانية تم تعينها إلى ٥٩ عندما تكون طريقة_ الثانية ، فإن ملف
- ، ثم يتسبب في ذلك Current_second to0 الطريقة المبرمجة لإعادة تعين
- increment_current_minutemethod ١ إلى للتنفيذ. تضيف هذه الطريقة ١ إلى increment_current_minuteattribute
- ، ما لم يتم تعينها إلى ٥٩. في هذه الحالة ، يتم إعادة تعينها
- ويسبب طريقة الزيادة الحالية في الساعة للتنفيذ ٠
- طريقة التزاييد الحالية
- إذا تم تشغيل الجهاز مرتين ، فإن طريقة إنذار الصوت تكون
- أعدم

As you can see, the data attributes are merely values that define the state that the alarm clock is currently in. You, the user of the alarm clock object, cannot directly manipulate these data attributes because they are private. To change a data attribute's value, you must use one of the object's methods. The following are some of the alarm clock object's methods:

- `set_time`
- `set_alarm_time`
- `set_alarm_on`
- `set_alarm_off`

كما ترون ، فإن السمات الخاصة بالبيانات ما زالت ذات قيمة كبيرة تحدد حالة الإنذار الساعة في الوقت الحالي. أنت ، مستخدم كائن ساعة الذراع ، لا يمكنك معالجته بشكل مباشر لتغيير سمات البيانات لأنها خاصة قيمة سمة البيانات ، أنت يجب أن تستخدم إحدى طرق بعض العناصر التالية من كائن . الكائن ساعة الذراع •أساليب •ضبط الوقت

Each method manipulates one or more of the data attributes. For example, the `set_time` method allows you to set the alarm clock's time. You activate the method by pressing a button on top of the clock. By using another button, you can activate the `set_alarm_time` method.

In addition, another button allows you to execute the `set_alarm_on` and `set_alarm_off` methods. Notice that all of these methods can be activated by you, who are outside the alarm clock. Methods that can be accessed by entities outside the object are known as **public methods**.

- كل طريقة تعالج واحدة أو أكثر من سمات البيانات ، على سبيل المثال ، ملف
- بضبط ساعة الذبذبات `set_timemethod` يسمح لك
- عن طريق استخدام زر آخر ، أنت .طريقة الضغط على زر أعلى الساعة
- يمكن تنشيط `theset_alarm_timemethod`.
- بالإضافة إلى ذلك ، يسمح لك زر آخر بتنفيذ مجموعة `alarm_onand`
- لاحظ أن جميع هذه الأساليب تم تفعيلها بواسطتك ، `set_alarm_off`. طرق
- الطرق التي يمكن أن تلجم إليها الكيانات .من هو الخارج على مدار الساعة
- يعرف الكائن الخارجي بالطرق العامة

The alarm clock also has private methods, which are part of the object's private, internal workings. External entities (such as you, the user of the alarm clock) do not have direct access to the alarm clock's private methods. The object is designed to execute these methods automatically and hide the details from you. The following are the alarm clock object's

private methods:

- increment_current_second
- increment_current_minute
- increment_current_hour
- sound_alarm

أيضاً على طرق خاصة `Thealarm` تحتوي ساعة ، والتي تكون جزءاً من كائن خاص ، الكيانات الخارجية (مثل مستخدم . الأعمال الداخلية الساعة) لا تفعل ذلك الوصول المباشر إلى الأساليب الخاصة لساعة تنشيط تم تصميم الكائن لـ . الساعة الآتى . ينفذ هذه الطريقة تلقائياً وتفاصيل مخفية منك `arethealarm clockobject's` طرق خاصة `increment_current_second` • `increment_current_minute` • `sound_alarm` • `زيادة_ساعة_حالية`

نقطة تفتيش Checkpoint

- 10.1 What is an object?
- 10.2 What is encapsulation?
- 10.3 Why is an object's internal data usually hidden from outside code?
- 10.4 What are public methods? What are private methods?
- ما هو الكائن؟ 10.1
- ما هو التغليف؟ 10.2
- لماذا البيانات الداخلية للكائن مخفية عادة من الترميز الخارجي؟ 10.3
- ما هي الطرق العامة وما هي الطرق الخاصة؟ 10.4

Concept: A class is code that specifies the data attributes and methods for a particular type of object.

المفهوم: Aclass هو رمز يحدد البيانات سمات وأساليب لنوع معين من الكائنات

- Now, let's discuss how objects are created in software.
- Before an object can be created, it must be **designed by a programmer**. The programmer determines the **data attributes and methods** that are necessary and then creates a class.
- Think of a class as a "**blueprint**" that objects may be created from. It serves a similar purpose as the blueprint for a house. The blueprint itself is not a house, but is a detailed description of a house. When we use the blueprint to build an actual house, we could say we are building an instance of the house described by the blueprint. If we so desire, we can build several identical houses from the same blueprint. **Each house is a separate instance of the house described by the blueprint**. This idea is illustrated in Figure 10-3.

الآن ، دعنا نناقش الأشياء التي تم إنشاؤها في البرامج .
يمكن إنشاء كائن قبل أن يتم تصميمه بواسطة •
ال مبرمج يحدد المبرمج سمات البيانات وأساليب التي تكون
ضرورية ثم كريتسا كلاس .
الذي يمكن إنشاء `classasa` "مخيط" فكر في •
انها تخدمه مماثلة . الكائنات منه
المخيط نفسه ليس منزلاً ، ولكنه . الغرض من أجل منزل
مفصل عندما نستخدم البصمة الزرقاء لبناء منزل . وصف المنزل
 حقيقي ، يمكننا ذلك
مثالاً عن المسكن الموصوف في `wearebuildingan` قل
إذا رغبت في ذلك ، المخيط
كل . يمكننا بناء عدة منازل متطابقة من نفس المخيط
منزل منفصل
هذه . مثيل المنزل الموصوف بواسطة الطبعة الزرقاء
الفكرة واضحة في ، الشكل ٣-١٠ .

A Class

- Another way of thinking about the difference between a class and an object is to think of the difference **between a cookie cutter and a cookie**. While a cookie cutter itself is not a cookie, **it describes a cookie. The cookie cutter can be used to make several cookies**, as shown in Figure 10-4. Think of a class as a cookie cutter and the objects created from the class as cookies.
- So, a class is a description of an object's characteristics. When the program is running, it can use the class to create, in memory, as many objects of a specific type as needed. Each object that is created from a class is called an **instance of the class**

هناك طريقة أخرى للتفكير في الفرق بين فئة و موضوع

- بينما .للتفكير في الفرق بين ملف تعريف الارتباط وملف تعريف الارتباط
- يمكن .نفسها ليست ملف تعريف ارتباط ، إنها تصف ملف تعريف الارتباط `cookiecutter`
- فكر في كلاسسا. يمكن استخدامها لعمل العديد من ملفات تعريف الارتباط ، كما هو موضح في الشكل ٤-١٠
- ملف تعريف الارتباط والشيء الذي تم إنشاؤه من كلاسيكيات ملفات تعريف الارتباط
- إذن ، وصف كلاسيزا الخصائص الكائن. عند البرنامج •
- قيد التشغيل ، يمكنه استخدام الكلاسيكيات لإنشاء العديد من الكائنات في الذاكرة
- كل كائن تم إنشاؤه من فئة تسمى ملف .النوعية المطلوبة
- مثيل من الفصل

A Class

Figure 10-3 A blueprint and houses built from the blueprint

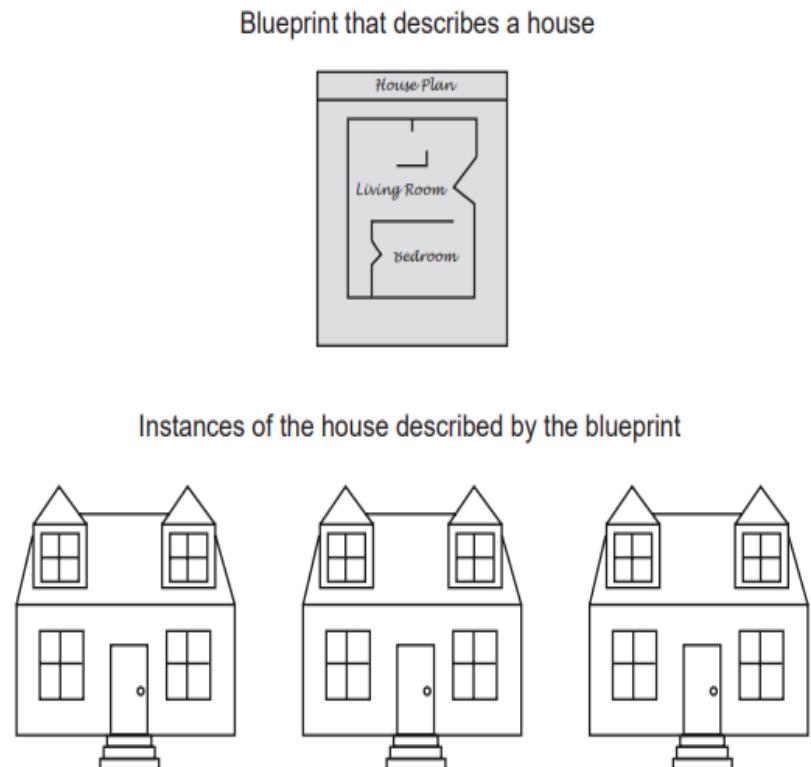
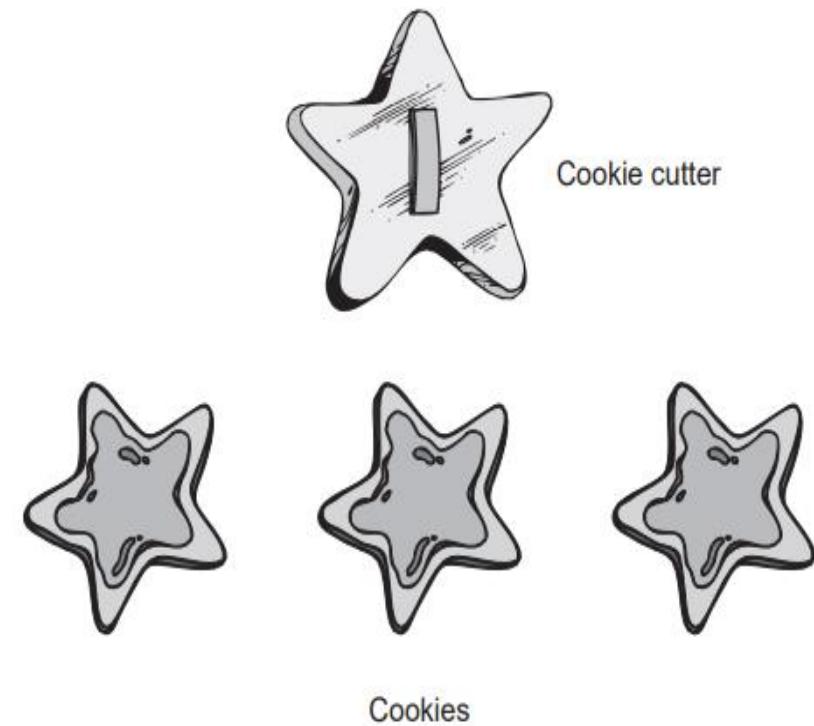


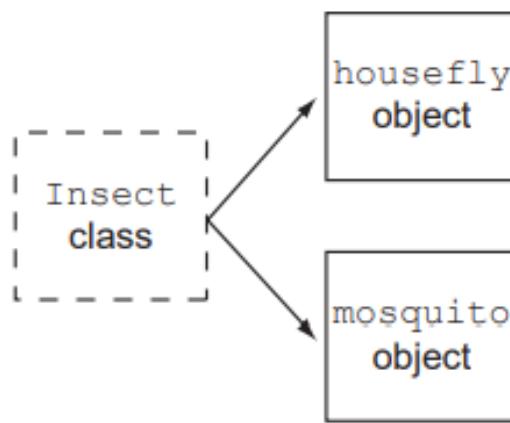
Figure 10-4 The cookie cutter metaphor



A Class

Figure 10-5 The housefly and mosquito objects are instances of the Insect class

The Insect class describes the data attributes and methods that a particular type of object may have.



The housefly object is an instance of the Insect class. It has the data attributes and methods described by the Insect class.

The mosquito object is an instance of the Insect class. It has the data attributes and methods described by the Insect class.

A Class Definitions تعاريفات AClass

- To create a class, you write a class definition.
- A class definition is a set of statements that define a class's methods and data attributes.
- Let's look at a simple example. Suppose we are writing a program to simulate the tossing of a coin. In the program we need to repeatedly toss the coin and each time determine whether it landed heads up or tails up. Taking an object-oriented approach, we will write a class named Coin that can perform the behaviors of the coin. Program 10-1 shows the class definition, which we will explain shortly. Note that this is not a complete program

، أنت تكتب فئة تعريف `To create a class` فئة

- تعريف الفئة هو مجموعة من العبارات التي تحدد طرق الفصل .
• وسمات البيانات
• دعونا ننظر إلى برنامج بسيط

يجب أن يكون محاكاة عملة معدنية في البرنامج مرارًا وتكرارًا
لكل عملة وقت محدد ما إذا كانت هبطت رأسًا أم لا
نهج وجوه المنحى ، وسوف نكتب الطبقة `Takingan` .
المسمة العملة التي يمكن أن تؤدي سلوكيات العملة. البرنامج 10-1
عرض تعريف الفئة ، والذي سوف يشرح بعد قليل
هذا ليس برنامج كامل



NOTE: The `__init__` method is usually the first method inside a class definition.

Program 10-2 (coin_demo1.py)

```
1 import random
2
3 # The Coin class simulates a coin that can
4 # be flipped.
5
6 class Coin:
7
8     # The __init__ method initializes the
9     # sideup data attribute with 'Heads'.
10
11    def __init__(self):
12        self.sideup = 'Heads'
13
14    # The toss method generates a random number
15    # in the range of 0 through 1. If the number
16    # is 0, then sideup is set to 'Heads'.
17    # Otherwise, sideup is set to 'Tails'.
18
19    def toss(self):
20        if random.randint(0, 1) == 0:
```

Program 10-2 (continued)

```
21             self.sideup = 'Heads'
22         else:
23             self.sideup = 'Tails'
24
25     # The get_sideup method returns the value
26     # referenced by sideup.
27
28     def get_sideup(self):
29         return self.sideup
30
31 # The main function.
32 def main():
33     # Create an object from the Coin class.
34     my_coin = Coin()
35
36     # Display the side of the coin that is facing up.
37     print('This side is up:', my_coin.get_sideup())
38
39     # Toss the coin.
40     print('I am tossing the coin . . .')
41     my_coin.toss()
42
43     # Display the side of the coin that is facing up.
44     print('This side is up:', my_coin.get_sideup())
45
46 # Call the main function.
47 main()
```

Program Output

```
This side is up: Heads
I am tossing the coin . . .
This side is up: Tails
```

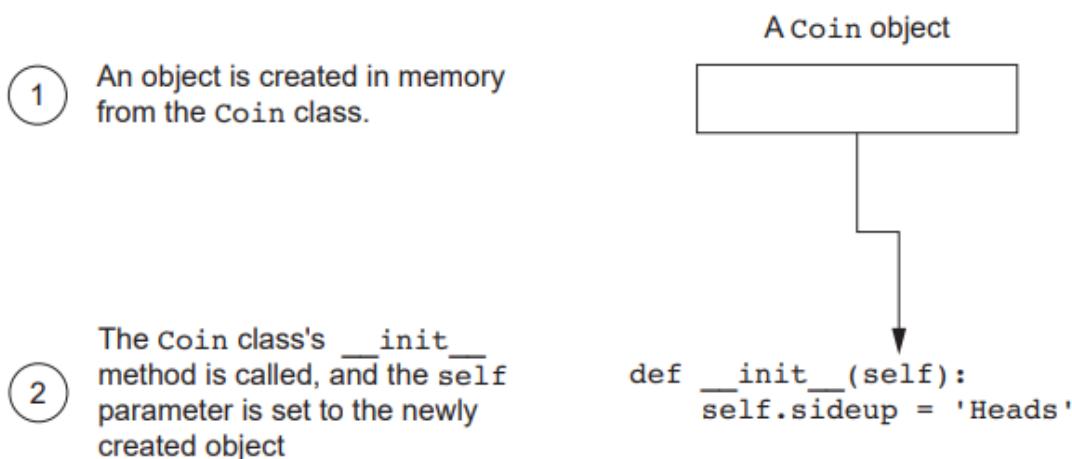
Program Output

```
This side is up: Heads
I am tossing the coin . . .
This side is up: Heads
```

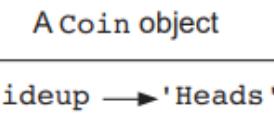
Program Output

Figure 10-6 Actions caused by the `coin()` expression

my_coin = Coin()

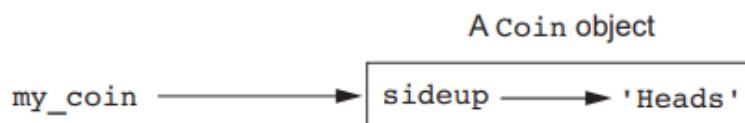


After these steps take place,
a `Coin` object will exist with its
`sideup` attribute set to '`Heads`'.



After this, the `=` operator assigns the `Coin` object that was just created to the `my_coin` variable. Figure 10-7 shows that after the statement in line 12 executes, the `my_coin` variable will reference a `Coin` object, and that object's `sideup` attribute will be assigned the string '`Heads`'.

Figure 10-7 The `my_coin` variable references a `Coin` object



Hiding Attributes إخفاء السمات

- An object's data attributes should be private, so that only the object's methods can directly access them. This protects the object's data attributes from accidental corruption. However, in the Coin class that was shown in the previous example, the sideup attribute is not private. It can be directly accessed by statements that are not in a Coin class method. Program 10-3 shows an example.

يجب أن تنفصل سمة بيانات كائن ما ، بحيث يكون الكائن فقط الأساليب التي تستجيب لها بشكل مباشر. تحمي بيانات الكائن الصفات من الفساد العرضي تم عرضه في المثال السابق ، هذا السمة الجانبية ليست كذلك خاص. يمكن الوصول إليها مباشرة من خلال البيانات التي لا توجد في يوضح البرنامج ١٠-٣ مثال: Coinclass طريقة

Program 10-3 (coin_demo2.py)

Lines 1 through 30 are omitted. These lines are the same as lines 1 through 30 in Program 10-2.

```
31 # The main function.  
32 def main():  
33     # Create an object from the Coin class.  
34     my_coin = Coin()  
35  
36     # Display the side of the coin that is facing up.  
37     print('This side is up:', my_coin.get_sideup())  
38  
39     # Toss the coin.  
40     print('I am tossing the coin . . .')  
41     my_coin.toss()  
42  
43     # But now I'm going to cheat! I'm going to  
44     # directly change the value of the object's  
45     # sideup attribute to 'Heads'.  
46     my_coin.sideup = 'Heads'  
47  
48     # Display the side of the coin that is facing up.
```

```
49     print('This side is up:', my_coin.get_sideup())  
50  
51 # Call the main function.  
52 main()
```

Program Output

```
This side is up: Heads  
I am tossing the coin . . .  
This side is up: Heads
```

Program Output

```
This side is up: Heads  
I am tossing the coin . . .  
This side is up: Heads
```

Program Output

```
This side is up: Heads  
I am tossing the coin . . .  
This side is up: Heads
```

- If we truly want to simulate a coin that is being tossed, then we don't want code outside the class to be able to change the result of the toss method. To prevent this from happening, we need to make the sideup attribute **private**.
- In Python you can hide an attribute **by starting its name with two underscore characters**. If we change the name of the sideup attribute to `__sideup`, then code outside the Coin class will not be able to access it. Program 10-4 shows a new version of the Coin class, with this change made.

إذا كنت ترغب في محاكاة عملات معدنية محاكية للعملة ، فلن تفعل ذلك نريد الكودات في أي نوع يمكن تغييره من النتيجة لمنع حدوث ذلك ، لا بد من القيام بذلك السمة الجانبية خاصة من خلال اسم `Pythonyou canhidean` في السمة • البداية مع اثنين إذا تم تغيير اسم هذه السمة الجانبية .أحرف سفلية ، ثم لن يتم عرض `_sideup` إلى `codeoutsidetheCoin` class يعرض البرنامج ١٠ - ٤ إصداراً جديداً من .الوصول إليه فئة العملة ، مع هذا متغير.

Program 10-4 (coin_demo3.py)

```
1 import random
2
3 # The Coin class simulates a coin that can
4 # be flipped.
5
6 class Coin:
7
8     # The __init__ method initializes the
```

Program 10-4 (*continued*)

```
9         # __sideup data attribute with 'Heads'.
10
11     def __init__(self):
12         self.__sideup = 'Heads'
13
14     # The toss method generates a random number
15     # in the range of 0 through 1. If the number
16     # is 0, then sideup is set to 'Heads'.
17     # Otherwise, sideup is set to 'Tails'.
18
19     def toss(self):
20         if random.randint(0, 1) == 0:
21             self.__sideup = 'Heads'
22         else:
23             self.__sideup = 'Tails'
24
25     # The get_sideup method returns the value
26     # referenced by sideup.
27
28     def get_sideup(self):
29         return self.__sideup
30
```

```
31 # The main function.  
32 def main():  
33     # Create an object from the Coin class.  
34     my_coin = Coin()  
35  
36     # Display the side of the coin that is facing up.  
37     print('This side is up:', my_coin.get_sideup())  
38  
39     # Toss the coin.  
40     print('I am going to toss the coin ten times:')  
41     for count in range(10):  
42         my_coin.toss()  
43         print(my_coin.get_sideup())  
44  
45 # Call the main function.  
46 main()
```

Program Output

```
This side is up: Heads  
I am going to toss the coin ten times:  
Tails  
Heads
```

تخزين الفصول في وحدات Storing Classes in Modules

- The programs you have seen so far in this chapter have the Coin class definition in the same file as the programming statements that use the Coin class. This approach works fine with **small programs** that use only one or two classes. As programs use more classes, however, the need to organize those classes becomes greater.
- Programmers commonly organize their class definitions by storing them in **modules**. Then the modules can be **imported** into any programs that need to use the classes they contain. For example, suppose we decide to store the Coin class in a module named coin. Program 10-5 shows the contents of the coin.py file. Then, when we need to use the Coin class in a program, we can import the coin module. This is demonstrated in Program 10-6.

البرامج التي تتخذها منذ بداية هذا الفصل تعريف فئة العملة في نفس الملف عبارة عن بيان البرمجة الذي يستخدم فئة العملة. هذا النهج ي العمل مع البرامج الصغيرة التي تستخدم فقط فئة أو فصليين. ومع ذلك ، فإن المزيد من الطبقات تصبح أكثر تنظيماً. ينظم المبرمجون عادة لفئاتهم تعريفاتهم في وحدات. يمكن استيراد هذه الوحدات إلى أي برمج تحتاج إلى فئات توسيعية لها تحتوي على ثم عندما يعرض البرنامج ٥-١٠ محتويات ملف العملة نحن ، يمكننا needtousetheCoinclassina هذا هو استيراد وحدة العملة ٦-١٠ موضّح في البرنامج

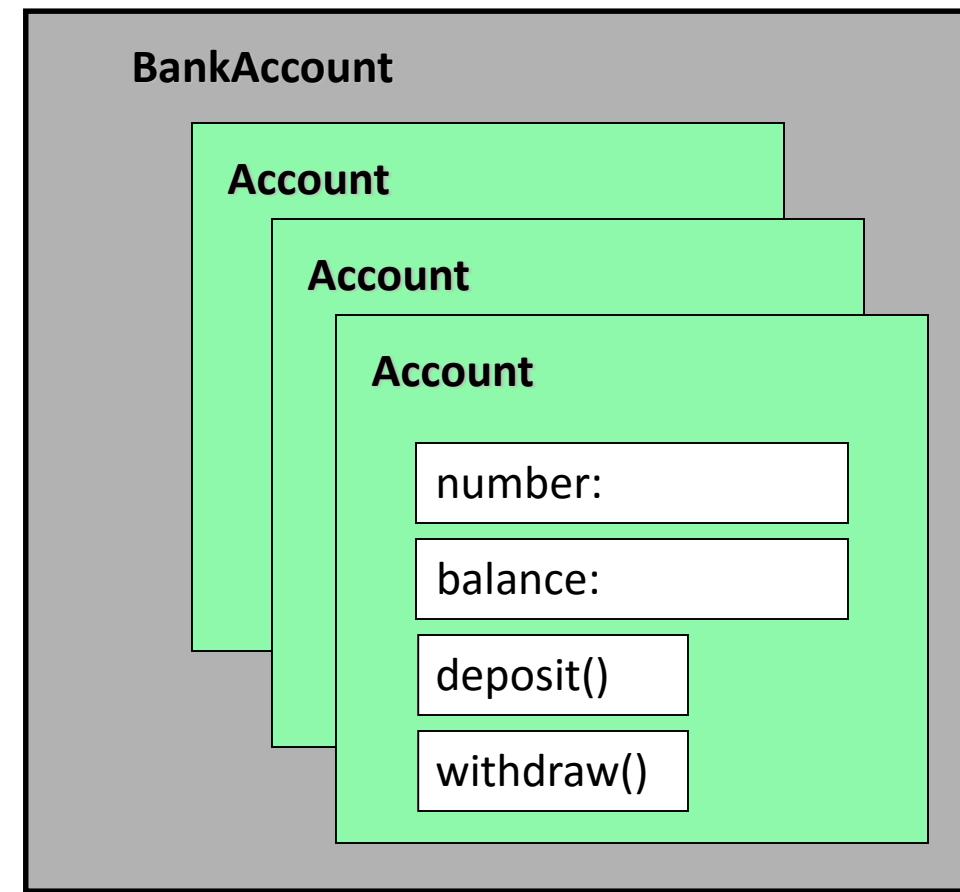
Program 10-6 (coin_demo4.py)

```
1 # This program imports the coin module and
2 # creates an instance of the Coin class.
3
4 import coin
5
6 def main():
7     # Create an object from the Coin class.
8     my_coin = coin.Coin()
9
10    # Display the side of the coin that is facing up.
11    print('This side is up:', my_coin.get_sideup())
12
13    # Toss the coin.
14    print('I am going to toss the coin ten times:')
15    for count in range(10):
16        my_coin.toss()
17        print(my_coin.get_sideup())
18
19    # Call the main function.
20 main()
```

Program Output

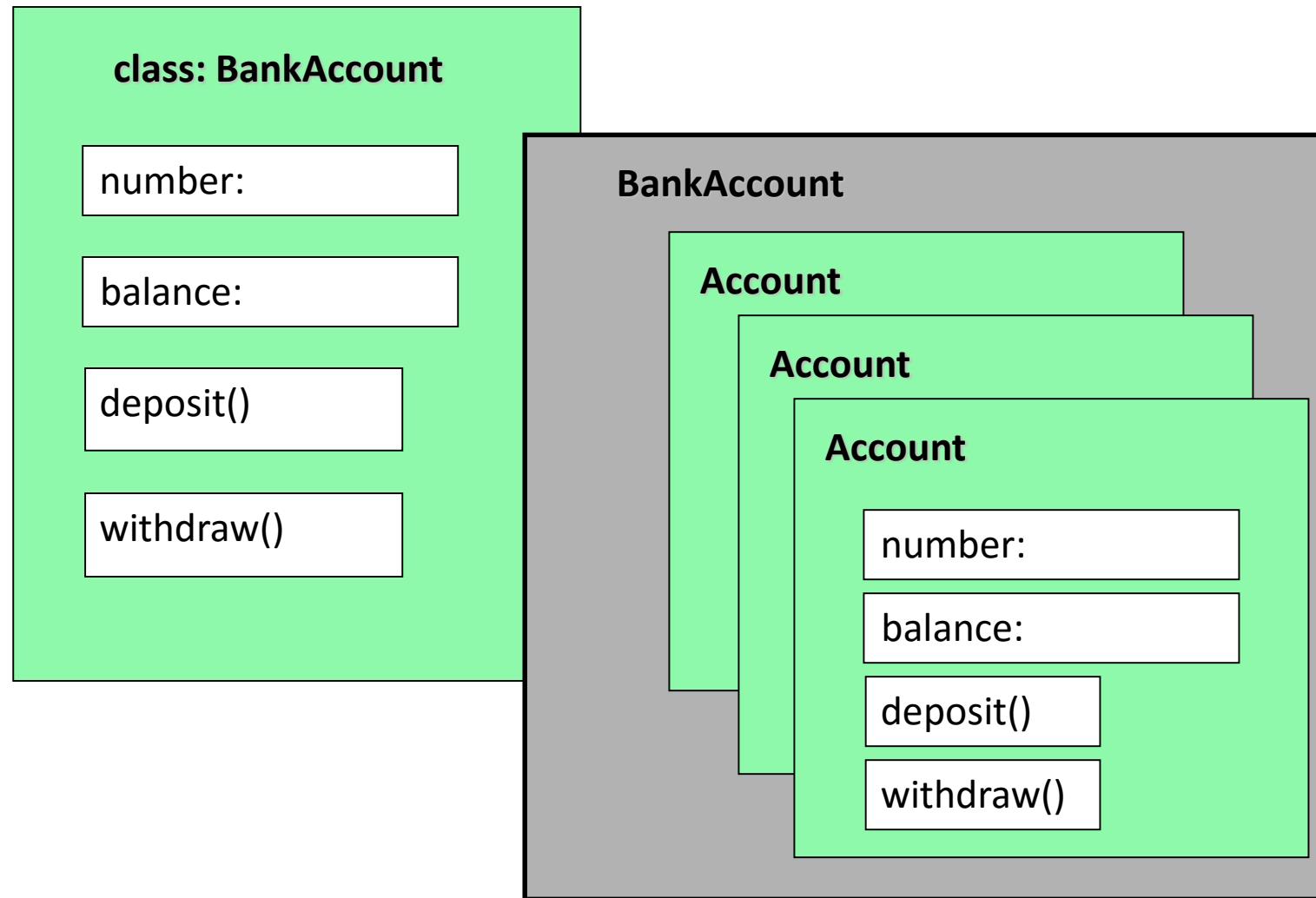
```
This side is up: Heads
I am going to toss the coin ten times:
Tails
Tails
Heads
Tails
Heads
Heads
```

- In **object-oriented languages**, they are defined together.
 - An object is a collection of attributes and the behaviors that operate on them.
- Variables in an object are called ***attributes***.
- Procedures associated with an object are called ***methods***.
- في اللغات الشيئية، تم تعريفهما الكائن هو مجموعة سمات بمعاً وسلوكيات التي تعمل معهم. المتغيرات في كائن تسمى السمات الإجراءات المرتبطة كائن يسمى الأساليب



Bank Example

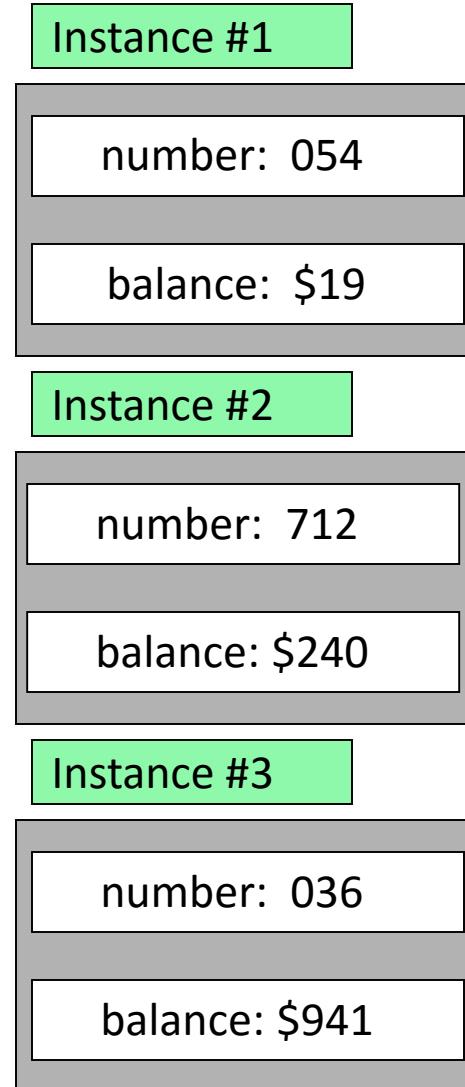
- The “BankAccount” class describes the attributes and behaviors of bank accounts.
- The “BankAccount” class defines two state variables (account number and balance) and two methods (deposit and withdraw).
- فئة "BankAccount"
 - تصف الصفات وسلوكيات البنك
 - فئة حسابات
 - تعريف "BankAccount" دوالتين المتغيرات (حساب العدد twomethods (الإيداع وانسحب).



Bank Example (Cont.)

- When the program runs there will be many instances of the account class.
- Each instance will have its own account number and balance (*object state*)
- Methods can only be invoked .

عند وجود البرنامج
سوف تتأثر بالعديد من الظروف فئة
كل حالة سيكون لها مكان رقم .الحساب
(حالة الكائن)الحساب والرصيد
الطريقة يمكن الاستدعاء إليها فقط.



The BankAccount Class

Program 10-7 (bankaccount.py)

```
1 # The BankAccount class simulates a bank account.  
2  
3 class BankAccount:  
4  
5     # The __init__ method accepts an argument for  
6     # the account's balance. It is assigned to  
7     # the __balance attribute.  
8  
9     def __init__(self, bal):  
10        self.__balance = bal  
11  
12    # The deposit method makes a deposit into the  
13    # account.  
14  
15    def deposit(self, amount):  
16        self.__balance += amount
```

```
17  
18     # The withdraw method withdraws an amount  
19     # from the account.  
20  
21     def withdraw(self, amount):  
22         if self.__balance >= amount:  
23             self.__balance -= amount  
24         else:  
25             print('Error: Insufficient funds')  
26  
27     # The get_balance method returns the  
28     # account balance.  
29  
30     def get_balance(self):  
31         return self.__balance
```

Notice that the `__init__` method has two parameter variables: `self` and `bal`. The `bal` parameter will accept the account's starting balance as an argument. In line 10 the `bal` parameter amount is assigned to the object's `__balance` attribute

تحتوي على متغيرين `__init__` لاحظ أن طريقة `self`: معلمين و `bal`: المعلمة `bal`.
الصورة `startingbalance`an حساب `bal` ، يتم تخصيص مبلغ معلمة 10 في السطر جدال إلى الكائن "s" `__balance`attribute

Program 10-8 (account_test.py)

```
1 # This program demonstrates the BankAccount class.  
2  
3 import bankaccount  
4  
5 def main():  
6     # Get the starting balance.  
7     start_bal = float(input('Enter your starting balance: '))  
8  
9     # Create a BankAccount object.  
10    savings = bankaccount.BankAccount(start_bal)  
11  
12    # Deposit the user's paycheck.  
13    pay = float(input('How much were you paid this week? '))  
14    print('I will deposit that into your account.')  
15    savings.deposit(pay)  
16  
17    # Display the balance.  
18    print('Your account balance is $', \  
19        format(savings.get_balance(), ',.2f'), \  
20        sep='')  
21  
22    # Get the amount to withdraw.  
23    cash = float(input('How much would you like to withdraw? '))  
24    print('I will withdraw that from your account.')  
25    savings.withdraw(cash)  
26  
27    # Display the balance.  
28    print('Your account balance is $', \  
29        format(savings.get_balance(), ',.2f'), \  
30        sep='')
```

```
31  
32     # Call the main function.  
33     main()
```

Program Output (with input shown in bold)

```
Enter your starting balance: 1000.00   
How much were you paid this week? 500.00   
I will deposit that into your account.  
Your account balance is $1,500.00  
How much would you like to withdraw? 1200.00   
I will withdraw that from your account.  
Your account balance is $300.00
```

Program Output (with input shown in bold)

```
Enter your starting balance: 1000.00   
How much were you paid this week? 500.00   
I will deposit that into your account.  
Your account balance is $1,500.00  
How much would you like to withdraw? 2000.00   
I will withdraw that from your account.  
Error: Insufficient funds  
Your account balance is $1,500.00
```

Program 10-10 (account_test2.py)

```
1 # This program demonstrates the BankAccount class
2 # with the __str__ method added to it.
3
4 import bankaccount2
5
6 def main():
7     # Get the starting balance.
8     start_bal = float(input('Enter your starting balance: '))
9
10    # Create a BankAccount object.
11    savings = bankaccount2.BankAccount(start_bal)
12
13    # Deposit the user's paycheck.
14    pay = float(input('How much were you paid this week? '))
15    print('I will deposit that into your account.')
16    savings.deposit(pay)
17
18    # Display the balance.
19    print(savings)
20
21    # Get the amount to withdraw.
22    cash = float(input('How much would you like to withdraw? '))
23    print('I will withdraw that from your account.')
24    savings.withdraw(cash)
25
26    # Display the balance.
27    print(savings)
28
29 # Call the main function.
30 main()
```

32
33 # The __str__ method returns a string
34 # indicating the object's state.
35
36 def __str__(self):
37 return 'The balance is \$' + format(self.__balance, ',.2f')

We added __str__ in Account class

Program Output (with input shown in bold)

```
Enter your starting balance: 1000.00 
How much were you paid this week? 500.00 
I will deposit that into your account.
```

(program output co

Checkpoint

نقطة تفتيش

10.5 You hear someone make the following comment: "A blueprint is a design for a house. A carpenter can use the blueprint to build the house. If the carpenter wishes, he or she can build several identical houses from the same blueprint." Think of this as a metaphor for classes and objects. Does the blueprint represent a class, or does it represent an object?

10.6 In this chapter, we use the metaphor of a cookie cutter and cookies that are made from the cookie cutter to describe classes and objects. In this metaphor, are objects the cookie cutter, or the cookies?

10.7 What is the purpose of the `__init__` method? When does it execute?

10.8 What is the purpose of the `self` parameter in a method?

10.9 In a Python class, how do you hide an attribute from code outside the class?

10.10 What is the purpose of the `__str__` method?

10.11 How do you call the `__str__` method?

10.5 تسمع أحداً يقول التعليق التالي "Blueprint": هو تصميم لمنزل.

إذا رغب النجار يمكن أن يستخدم النجار طبعاً طبعاً لبناء المنزل، فيمكنه البناء

عدة منازل متطابقة من نفس الطبعة الزرقاء". فكر في استعارة هذه الفصول الدراسية

فئة أم أنها تمثل كائناً ما؟ Does the blueprint هل تمثل في هذا الفصل ، نستخدم استعارة طباخ وطهي مصنوع من cookycutter. In this metaphor توصف الفئات والكائنات هي الأشياء التي تم العثور عليها ، أو ملف بسكويت؟

10.7 ما هو الغرض من الطريقة `__init__` ؟

10.8 ما هو الغرض من المعلمة الذاتية في طريقة؟

10.9 In a Python class، كيف يتم تحديد السمة من الصنوف البرمجية؟

10.10 ما هو الغرض من طريقة `__str__` ؟

10.11 كيف يمكنك استدعاء طريقة `__str__` ؟

Concept: Each instance of a class has its own set of data attributes.

المفهوم : كل مثيل لفئة لديه مجموعة البيانات الخاصة به
صفات.

- When a method uses the **self parameter** to create an attribute, the attribute belongs to the specific object that self references. We call these attributes instance attributes because they belong to a specific instance of the class.

عندما تستخدم الطريقة المعلمة الذاتية لإنشاء السمة ، فإن •
يعود السبب في ذلك إلى الكائن المحدد الذي يشير إلى الذات •
هذه النسب في المقام الأول هي النسب لأنهم ينتمون إلى فئة معينة •
مثيل من الفئة •

Figure 10-8 The `coin1`, `coin2`, and `coin3` variables reference three `Coin` objects

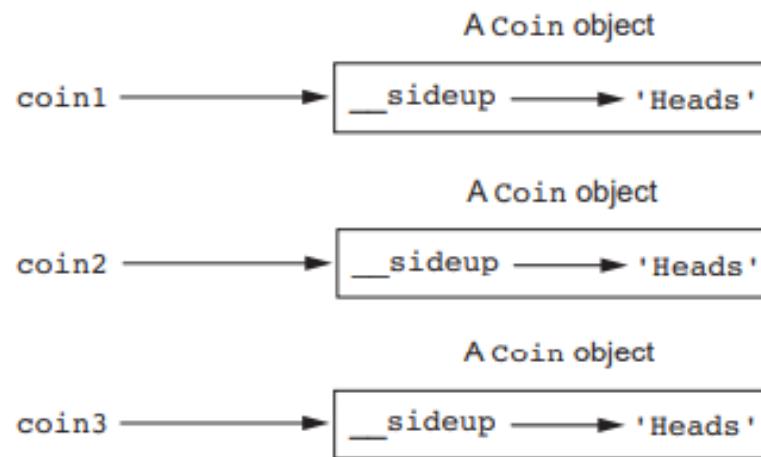
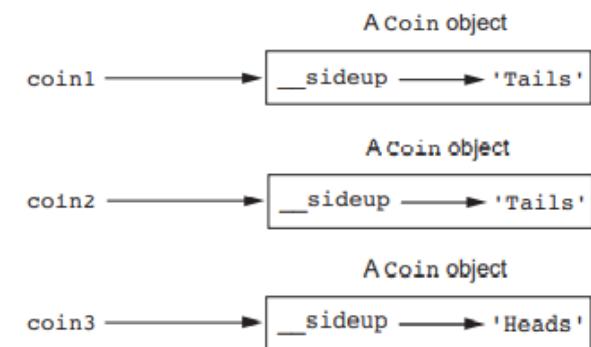


Figure 10-9 The objects after the `toss` method



Then, the statements in lines 22 through 24 call each object's `toss` method:

```
coin1.toss()  
coin2.toss()  
coin3.toss()
```

Figure 10-9 shows how these statements changed each object's `_sideup` attribute in the program's sample run.

Accessor and Mutator Methods

طرق Accessor و Mutator

- A method that returns a value from a class's attribute but does not change it is known as an **accessor method**. Accessor methods provide a safe way for code outside the class to **retrieve the values of attributes**, without exposing the attributes in a way that they could be changed by the code outside the method
- A method that **stores a value in a data attribute** or changes the value of a data attribute in some other way is known as a **mutator method**. Mutator methods can control the way that a class's data attributes are modified. When code outside the class needs to change the value of an object's data attribute, it typically calls a mutator and passes the new value as an argument. If necessary, the mutator can validate the value before it assigns it to the data attribute.

الطريقة التي تعيد السمة الخاصة بفئة الورم القيم ولكنها لا تتغير طرق الموصل. طريقة الموصل المعروفة لاسترجاع قيم السمات ، دون فضح الصفات بطريقة يمكن أن يتغيروا بها بأسلوب الشفرة طريقة تخزن قيمة أو تغيير قيمة البيانات •

طرق المотор يمكن .السمة بطريقة أخرى طريقة المطر المعرفة التحكم في طريقة تعديل سمات بيانات الفصل سمة ، classneeds to change the value of an object's data عادةً ما تسمى إذا لزم الأمر ، يمكن للوتر .المتحور والمرور بحجة جديدة للقيمة صحة القيمة قبل تعينها لسمة البيانات



NOTE: Mutator methods are sometimes called “setters,” and accessor methods are sometimes called “getters.”

Program 10-12 (cellphone.py)

```
1 # The CellPhone class holds data about a cell phone.  
2  
3 class CellPhone:  
4  
5     # The __init__ method initializes the attributes.  
6  
7     def __init__(self, manufact, model, price):  
8         self.__manufact = manufact  
9         self.__model = model  
10        self.__retail_price = price  
11  
12    # The set_manufact method accepts an argument for  
13    # the phone's manufacturer.  
14  
15    def set_manufact(self, manufact):  
16        self.__manufact = manufact
```

Program 10-12 (continued)

```
17  
18    # The set_model method accepts an argument for  
19    # the phone's model number.  
20  
21    def set_model(self, model):  
22        self.__model = model  
23  
24    # The set_retail_price method accepts an argument  
25    # for the phone's retail price.  
26  
27    def set_retail_price(self, price):  
28        self.__retail_price = price  
29  
30    # The get_manufact method returns the  
31    # phone's manufacturer.  
32  
33    def get_manufact(self):  
34        return self.__manufact  
35  
36    # The get_model method returns the  
37    # phone's model number.  
38  
39    def get_model(self):  
40        return self.__model  
41  
42    # The get_retail_price method returns the  
43    # phone's retail price.  
44  
45    def get_retail_price(self):  
46        return self.__retail_price
```

Program 10-13 (cell_phone_test.py)

```
1 # This program tests the CellPhone class.  
2  
3 import cellphone  
4  
5 def main():  
6     # Get the phone data.  
7     man = input('Enter the manufacturer: ')  
8     mod = input('Enter the model number: ')  
9     retail = float(input('Enter the retail price: '))  
10    # Create an instance of the CellPhone class.  
11    phone = cellphone.CellPhone(man, mod, retail)  
12  
13    # Display the data that was entered.  
14    print('Here is the data that you entered: ')  
15    print('Manufacturer:', phone.get_manufact())  
16    print('Model Number:', phone.get_model())  
17    print('Retail Price: $', format(phone.get_retail_price(), ',.2f'), sep='')  
18  
19    # Call the main function.  
20 main()
```

Program Output (with input shown in bold)

```
Enter the manufacturer: Acme Electronics [Enter]  
Enter the model number: M1000 [Enter]  
Enter the retail price: 199.99 [Enter]  
Here is the data that you entered:  
Manufacturer: Acme Electronics  
Model Number: M1000  
Retail Price: $199.99
```

Program 10-14

(cell_phone_list.py)

```
1 # This program creates five CellPhone objects and
2 # stores them in a list.
3
4 import cellphone
5
6 def main():
7     # Get a list of CellPhone objects.
8     phones = make_list()
9
10    # Display the data in the list.
11    print('Here is the data you entered:')
12    display_list(phones)
13
14 # The make_list function gets data from the user
15 # for five phones. The function returns a list
16 # of CellPhone objects containing the data.
17
18 def make_list():
19     # Create an empty list.
20     phone_list = []
21
22     # Add five CellPhone objects to the list.
23     print('Enter data for five phones.')
24     for count in range(1, 6):
25         # Get the phone data.
26         print('Phone number ' + str(count) + ':')
27         man = input('Enter the manufacturer: ')
28         mod = input('Enter the model number: ')
29         retail = float(input('Enter the retail price: '))
30         print()
31
32         # Create a new CellPhone object in memory and
33         # assign it to the phone variable.
34         phone = cellphone.CellPhone(man, mod, retail)
35
36         # Add the object to the list.
```

Storing Objects in a List

```
37         phone_list.append(phone)
38
39     # Return the list.
40     return phone_list
41
42 # The display_list function accepts a list containing
43 # CellPhone objects as an argument and displays the
44 # data stored in each object.
45
46 def display_list(phone_list):
47     for item in phone_list:
48         print(item.get_manufact())
49         print(item.get_model())
50         print(item.get_retail_price())
51         print()
52
53 # Call the main function.
54 main()
```

Program Output (with input shown in bold)

Enter data for five phones.

Phone number 1:

Enter the manufacturer: **Acme Electronics**

Enter the model number: **M1000**

Enter the retail price: **199.99**

Phone number 2:

Enter the manufacturer: **Atlantic Communications**

Enter the model number: **S2**

Enter the retail price: **149.99**

Phone number 3:

Enter the manufacturer: **Wavelength Electronics**

Enter the model number: **N477**

Passing Objects as Arguments

تمرير الكائنات باسم
الحج

Program 10-15 (coin_argument.py)

```
1 # This program passes a Coin object as
2 # an argument to a function.
3 import coin
4
5 # main function
6 def main():
7     # Create a Coin object.
8     my_coin = coin.Coin()
9
10    # This will display 'Heads'.
11    print(my_coin.get_sideup())
12
13    # Pass the object to the flip function.
14    flip(my_coin)
15
16    # This might display 'Heads', or it might
17    # display 'Tails'.
18    print(my_coin.get_sideup())
19
20    # The flip function flips a coin.
21    def flip(coin_obj):
22        coin_obj.toss()
23
24    # Call the main function.
25    main()
```

Program Output

```
Heads
Tails
```

Program Output

```
Heads
Heads
```

Program Output

```
Heads
Tails
```

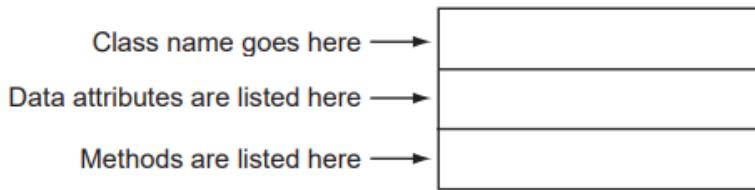
Techniques for Designing Classes

تقنيات تصميم الفصول

- **The Unified Modeling Language**
- When designing a class, it is often helpful to draw a **UML** diagram. UML stands for Unified Modeling Language. It provides a set of standard diagrams for graphically depicting object oriented systems. Figure 10-10 shows the general layout of a UML diagram for a class. Notice that the diagram is a box that is divided into three sections. The top section is where you write the **name of the class**. The middle section holds a list of the class's **data attributes**. The bottom section holds a list of the class's **methods**.

UML، من المفيد رسم مخطط Whendesigninga فئة UML تقف على UnifiedModelingLanguage. يوفر مجموعة من مخططات معيارية لتصوير الأنظمة الموجهة للكائنات بيانياً. UML يوضح الشكل ١٠-١٠ التخطيط العام لمخطط لفصل دراسي. لاحظ أن المخطط عبارة عن صندوق ينقسم إلى أقسام الجزء العلوي هو المكان الذي تكتب فيه الوسط. اسم الطبقة Thebottom قسم بسمات بيانات الفصل قائمة بأساليب الفصل.

Figure 10-10 General layout of a UML diagram for a class



Following this layout, Figure 10-11 and 10-12 show UML diagrams for the `Coin` class and the `CellPhone` class that you saw previously in this chapter. Notice that we did not show the `self` parameter in any of the methods, since it is understood that the `self` parameter is required.

Figure 10-11 UML diagram for the `Coin` class

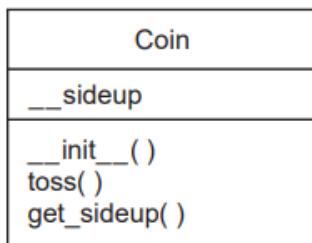
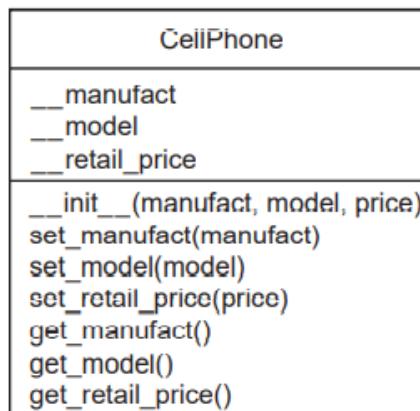


Figure 10-12 UML diagram for the `CellPhone` class



Finding the Classes in a Problem

إيجاد الفصول في مشكلة

- When developing an object-oriented program, one of your first tasks is to identify the classes that you will need to create. Typically, your goal is to identify the different types of real-world objects that are present in the problem and then create classes for those types of objects within your application.
- Over the years, software professionals have developed numerous techniques for finding the classes in a given problem. One simple and popular technique involves the following steps.
 - 1. Get a written description of the problem domain.
 - 2. Identify all the nouns (including pronouns and noun phrases) in the description. Each of these is a potential class.
 - 3. Refine the list to include only the classes that are relevant to the problem
 - Some of the nouns might represent objects, not classes.
 - Some of the nouns might represent simple values that can be assigned to a variable and do not require a class.

أثناء تطوير برنامج موجه نحو كائن ، واحدة من مهامك الأولى لتحديد الفئة التي سوف تحتاج إلى إنشاء . عادة ، هدفك هو تحديد أنواع مختلفة من الكائنات في العالم الحقيقي الموجودة في المشكلة وخلق فصولاً لهذه الأنواع من الكائنات في داخلنا تطبيق.

• على مر السنين ، تم تطوير تقنيات عدبية صوتية محترفة برمجية لإيجادها معين المشكلة . تقنية بسيطة وشعبية تتضمن الخطوات التالية.

1. احصل على وصف مكتوب لمجال المشكلة.
2. تحديد جميع الأسماء (بما في ذلك الأسماء والعبارات) في الوصف. كل من هذه عيسى فئة محتملة.
3. إعادة تحديد القائمة لتشمل فقط الفئة التي ليست ذات صلة بالمشكلة . بعض الأسماء تمثل الأشياء وليس الفئات.
- بعض الأسماء تمثل الأشياء وليس الفئات.
- بعض nouns might be values that تمثل ، nouns might be can be سط

Identifying a Class's Responsibilities

تحديد مسؤوليات الفصل

Once the classes have been identified, the next task is to identify each class's responsibilities.

A class's responsibilities are

- the things that the class is responsible for knowing
- the actions that the class is responsible for doing

بمجرد تحديد الفئة ، فإن المهمة التالية هي التعرف على كل منها . مسؤوليات الفصل هي

- الأشياء التي تعتبر الطبقة مسؤولة عن المعرفة
- الإجراءات التي تعتبر الطبقة مسؤولة عن فعلها

فئة العميل (مثال) (EX.)

In the context of our problem domain, what must the Customer class know? The description directly mentions the following items, which are all data attributes of a customer:

- the customer's name
- the customer's address
- the customer's telephone number

Now let's identify the class's methods. In the context of our problem domain, what must

the Customer class do? The only obvious actions are:

- initialize an object of the Customer class
- set and return the customer's name
- set and return the customer's address
- set and return the customer's telephone number

في سياق مجال مشكلتنا ، ما الذي يجب أن يعرفه فئة العميل؟

الإشارة مباشرة إلى العناصر التالية ، والتي تمثل جميع سمات بيانات العميل:

اسم العميل

عنوان العميل

رقم هاتف العميل

الآن دعنا نحدد طرق الفصل من `In the context`. من

مجال مشكلتنا ، ما يجب

فئة العملاء هي التصرفات الواضحة فقط:

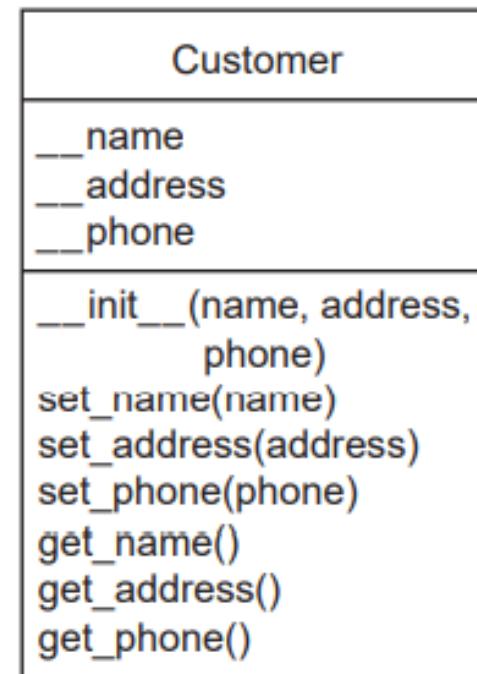
كائن من فئة العميل

تعيين وإرجاع اسم العميل

تعيين وإعادة عنوان العميل

تعيين وإرجاع رقم هاتف العميل

Figure 10-13 UML diagram for the Customer class



Program 10-20 (customer.py)

```
1 # Customer class
2 class Customer:
3     def __init__(self, name, address, phone):
4         self.__name = name
5         self.__address = address
6         self.__phone = phone
7
8     def set_name(self, name):
9         self.__name = name
10
11    def set_address(self, address):
12        self.__address = address
13
14    def set_phone(self, phone):
15        self.__phone = phone
16
17    def get_name(self):
18        return self.__name
19
20    def get_address(self):
21        return self.__address
22
23    def get_phone(self):
24        return self.__phone
```

Checkpoint

10.15 The typical UML diagram for a class has three sections. What appears in these

three sections?

10.16 What is a problem domain?

10.17 When designing an object-oriented application, who should write a description of the problem domain?

10.18 How do you identify the potential classes in a problem domain description?

10.19 What are a class's responsibilities?

10.20 What two questions should you ask to determine a class's responsibilities?

10.21 Will all of a class's actions always be directly mentioned in the problem domain

description?

10.15 مخطط UML النموذجي لأقسام الفصل . ما يظهر
الثلاثية؟

10.16 ما هو مجال المشكلة؟

10.17 عند تصميم تطبيق موجه للكائنات ، من الذي يجب
أن يكتب وصفاً للمشكلة
نطاق؟

10.18 كيف تحدد الفئات المحتملة في وصف مجال
المشكلة؟

10.19 ما هي مسؤوليات فئة المنطقة؟

10.20 ما المسؤولين الذين يجب أن تطرحهم لتحديد
مسؤوليات الفصل؟

10.21 هل ستشار دائماً إلى جميع تصرفات الفصل
مباشرةً في مجال المشكلة
وصف؟

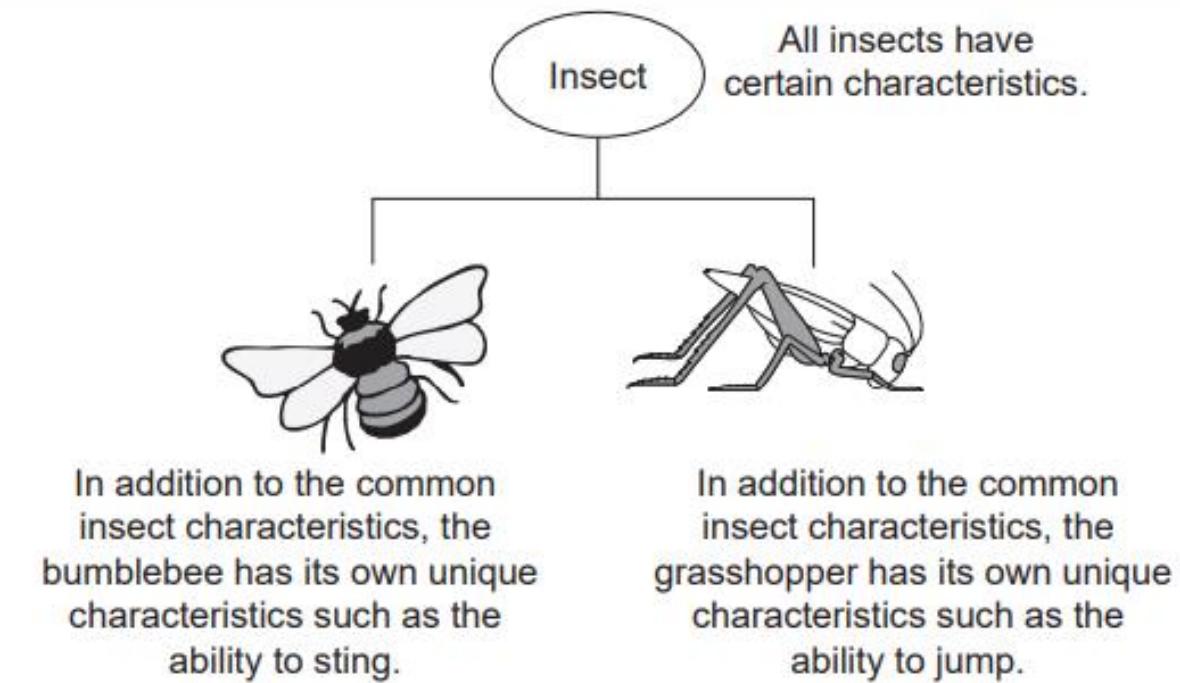
Concept: Inheritance allows a new class to extend an existing class. The new class inherits the members of the class it extends

المفهوم: الوراثة تسمح لفئة جديدة بتوسيع الموجودة
تراث الطبقة الجديدة أعضاء صنفها.
يمتد

- Generalization and Specialization In the real world, you can find many objects that are specialized versions of other more general objects. For example, the term “insect” describes a general type of creature with various characteristics. Because grasshoppers and bumblebees are insects, they have all the general characteristics of an insect. In addition, they have special characteristics of their own

التعيم والتخصيص في العالم الواقعي ،
يمكنك العثور على العديد
الأشياء التي تخص إصدارات متخصصة
لأشياء أخرى عامة
على سبيل المثال ، مصطلح "حشرة"
يصف النوع العام للمخلوق
لأننا نظرد ونحل . مع خصائص مختلفة
نحل
هي حشرات لها خصائص عامة للحشرة
بالإضافة إلى ذلك ، لديهم خصائص خاصة
بهم

Figure 11-1 Bumblebees and grasshoppers are specialized versions of an insect

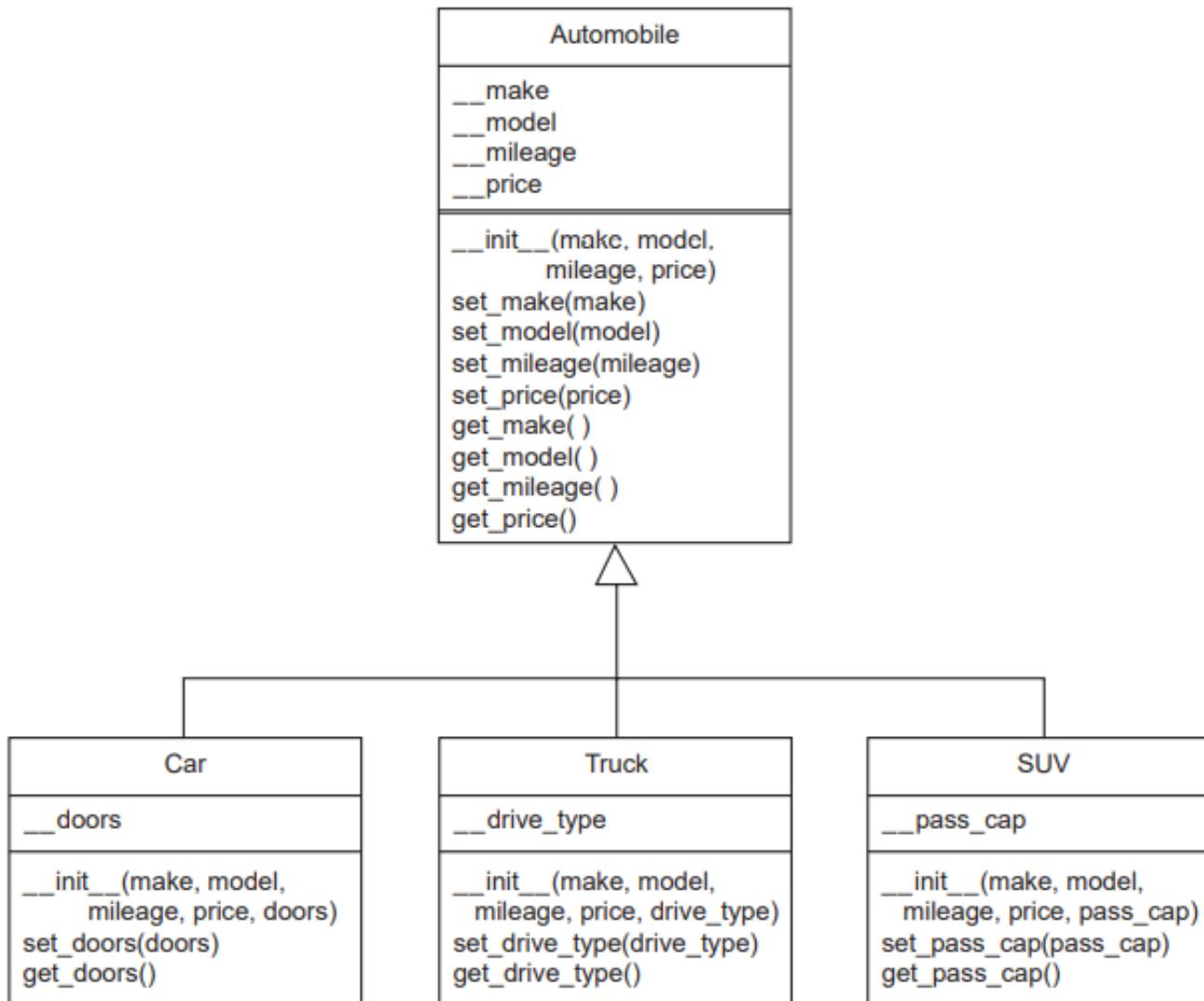


الميراث و علاقه "Is a" Relationship

- When an "is a" relationship exists between objects, it means that the specialized object has all of the characteristics of the general object, plus additional characteristics that make it special. In object-oriented programming, inheritance is used to create an "is a" relationship among classes. This allows you to extend the capabilities of a class by creating another class that is a specialized version of it.

" بين الكائنات ، Is a عندما توجد علاقة " فهذا يعني أن الكائن المتخصص لديه كل خصائص الكائن العام ، بالإضافة إلى الخصائص الإضافية التي تجعلها مميزة البرمجة ، الميراث تستخدم لإنشاء علاقة " Is a " وهذا يسمح أيضاً بتوسيع بين الطبقات قدرات الفصل الدراسي خلق فئة أخرى هي نسخة متخصصة منه

Figure 11-2 UML diagram showing inheritance



Example

A Car class with data attributes for the make, year model, mileage, price, and the number of doors.

A Truck class with data attributes for the make, year model, mileage, price, and the drive type.

An SUV class with data attributes for the make, year model, mileage, price, and the passenger capacity

فئة السيارة مع سمات البيانات
الخاصة بها ، موديل السنة ،
الأميال ،
السعر وعدد الأبواب.

فئة الشاحنات ذات سمات البيانات
الخاصة بالموضوع ، نموذج السنة
، الأميال ،
السعر والنوع.

فئة سيارات الدفع الرباعي مع
سمات البيانات الخاصة بالموضوع
، نموذج السنة ، الأميال ،
السعر وقدرة الركاب

Program 11-1 (Lines 1 through 44 of vehicles.py)

```
1 # The Automobile class holds general data
2 # about an automobile in inventory.
3
4 class Automobile:
5     # The __init__method accepts arguments for the
6     # make, model, mileage, and price. It initializes
7     # the data attributes with these values.
8
9     def __init__(self, make, model, mileage, price):
10         self.__make = make
```

Program 11-1 *(continued)*

```
11         self.__model = model
12         self.__mileage = mileage
13         self.__price = price
14
15     # The following methods are mutators for the
16     # class's data attributes.
17
18     def set_make(self, make):
19         self.__make = make
20
21     def set_model(self, model):
22         self.__model = model
23
24     def set_mileage(self, mileage):
25         self.__mileage = mileage
26
27     def set_price(self, price):
28         self.__price = price
29
30     # The following methods are the accessors
31     # for the class's data attributes.
32
33     def get_make(self):
34         return self.__make
35
36     def get_model(self):
37         return self.__model
38
39     def get_mileage(self):
40         return self.__mileage
41
42     def get_price(self):
43         return self.__price
44
```

Program 11-2 (Lines 45 through 72 of vehicles.py)

```
45 # The Car class represents a car. It is a subclass
46 # of the Automobile class.
47
48 class Car(Automobile):
49     # The __init__ method accepts arguments for the
50     # car's make, model, mileage, price, and doors.
51
52     def __init__(self, make, model, mileage, price, doors):
53         # Call the superclass's __init__ method and pass
54         # the required arguments. Note that we also have
55         # to pass self as an argument.
56         Automobile.__init__(self, make, model, mileage, price)
57
58         # Initialize the __doors attribute.
59         self.__doors = doors
60
61     # The set_doors method is the mutator for the
62     # __doors attribute.
63
64     def set_doors(self, doors):
65         self.__doors = doors
66
67     # The get_doors method is the accessor for the
68     # __doors attribute.
69
70     def get_doors(self):
71         return self.__doors
72
```

Program 11-3 (car_demo.py)

```
1 # This program demonstrates the Car class.  
2  
3 import vehicles  
4  
5 def main():  
6     # Create an object from the Car class.  
7     # The car is a 2007 Audi with 12,500 miles, priced  
8     # at $21,500.00, and has 4 doors.  
9     used_car = vehicles.Car('Audi', 2007, 12500, 21500.00, 4)  
10  
11    # Display the car's data.  
12    print('Make:', used_car.get_make())  
13    print('Model:', used_car.get_model())  
14    print('Mileage:', used_car.get_mileage())  
15    print('Price:', used_car.get_price())  
16    print('Number of doors:', used_car.get_doors())  
17  
18    # Call the main function.  
19    main()
```

Program Output

Make: Audi

Model: 2007

Program 11-4 (Lines 73 through 100 of vehicles.py)

```
73 # The Truck class represents a pickup truck. It is a
74 # subclass of the Automobile class.
75
76 class Truck(Automobile):
77     # The __init__ method accepts arguments for the
78     # Truck's make, model, mileage, price, and drive type.
79
80     def __init__(self, make, model, mileage, price, drive_type):
81         # Call the superclass's __init__ method and pass
82         # the required arguments. Note that we also have
83         # to pass self as an argument.
84         Automobile.__init__(self, make, model, mileage, price)
85
86     # Initialize the drive type attribute.
87     self.__drive_type = drive_type
88
89     # The set_drive_type method is the mutator for the
90     # __drive_type attribute.
91
92     def set_drive_type(self, drive_type):
93         self.__drive = drive_type
94
95     # The get_drive_type method is the accessor for the
96     # __drive_type attribute.
97
98     def get_drive_type(self):
99         return self.__drive_type
100
```

Program 11-5 (Lines 101 through 128 of vehicles.py)

```
101 # The SUV class represents a sport utility vehicle. It
102 # is a subclass of the Automobile class.
103
104 class SUV(Automobile):
105     # The __init__ method accepts arguments for the
106     # SUV's make, model, mileage, price, and passenger
107     # capacity.
108
109     def __init__(self, make, model, mileage, price, pass_cap):
110         # Call the superclass's __init__ method and pass
111         # the required arguments. Note that we also have
112         # to pass self as an argument.
113         Automobile.__init__(self, make, model, mileage, price)
114
115         # Initialize the __pass_cap attribute.
116         self.__pass_cap = pass_cap
117
118     # The set_pass_cap method is the mutator for the
119     # __pass_cap attribute.
120
121     def set_pass_cap(self, pass_cap):
122         self.__pass_cap = pass_cap
123
124     # The get_pass_cap method is the accessor for the
125     # __pass_cap attribute.
126
127     def get_pass_cap(self):
128         return self.__pass_cap
```

Program 11-6 (car_truck_suv_demo.py)

```
1 # This program creates a Car object, a Truck object,
2 # and an SUV object.
3
4 import vehicles
5
6 def main():
7     # Create a Car object for a used 2001 BMW
8     # with 70,000 miles, priced at $15,000, with
9     # 4 doors.
10    car = vehicles.Car('BMW', 2001, 70000, 15000.0, 4)
11
12    # Create a Truck object for a used 2002
13    # Toyota pickup with 40,000 miles, priced
14    # at $12,000, with 4-wheel drive.
15    truck = vehicles.Truck('Toyota', 2002, 40000, 12000.0, '4WD')
16
17    # Create an SUV object for a used 2000
18    # Volvo with 30,000 miles, priced
19    # at $18,500, with 5 passenger capacity.
20    suv = vehicles.SUV('Volvo', 2000, 30000, 18500.0, 5)
21
22    print('USED CAR INVENTORY')
23    print('=====')
24
25    # Display the car's data.
26    print('The following car is in inventory:')
27    print('Make:', car.get_make())
28    print('Model:', car.get_model())
29    print('Mileage:', car.get_mileage())
30    print('Price:', car.get_price())
31    print('Number of doors:', car.get_doors())
32    print()
33
34    # Display the truck's data.
35    print('The following pickup truck is in inventory.')
```

Program 11-6 (continued)

```
36     print('Make:', truck.get_make())
37     print('Model:', truck.get_model())
38     print('Mileage:', truck.get_mileage())
39     print('Price:', truck.get_price())
40     print('Drive type:', truck.get_drive_type())
41     print()
42
43     # Display the SUV's data.
44     print('The following SUV is in inventory.')
45     print('Make:', suv.get_make())
46     print('Model:', suv.get_model())
47     print('Mileage:', suv.get_mileage())
48     print('Price:', suv.get_price())
49     print('Passenger Capacity:', suv.get_pass_cap())
50
51 # Call the main function.
52 main()
```

Program Output

```
USED CAR INVENTORY
=====
The following car is in inventory:
Make: BMW
Model: 2001
Mileage: 70000
Price: 15000.0
Number of doors: 4

The following pickup truck is in inventory.
Make: Toyota
Model: 2002
Mileage: 40000
Price: 12000.0
Drive type: 4WD

The following SUV is in inventory.
Make: Volvo
Model: 2000
Mileage: 30000
Price: 18500.0
```

Checkpoint

11.1 In this section we discussed super classes and subclasses. Which is the general class, and which is the specialized class?

11.2 What does it mean to say there is an “is a” relationship between two objects?

11.3 What does a subclass inherit from its superclass?

11.4 Look at the following code, which is the first line of a class definition. What is the name of the superclass? What is the name of the subclass?

class Canary(Bird)

11.1 في هذا القسم ، يتم مناقشة الفصول الفائقة والفئات الفرعية . الذي

الفئة العامة وأي فئة متخصصة؟

11.2 ماذا يعني أن هناك علاقة “is a“ بينهما

شيئين؟

11.3 ما الذي ترثه الفئة الفرعية من صنفها الفائق؟

11.4 ابحث عن الكود التالي ، وهو السطر الأول من الفصل التعريف ، ما هو اسم الطبقة العليا؟

هذه الفئة الفرعية؟

(**class Canary**)

Concept: Polymorphism allows subclasses to have methods with the same names as methods in their superclasses. It gives the ability for a program to call the correct method depending on the type of object that is used to call it.

المفهوم: يعطي القدرة على البرنامج لاستدعاء الطريقة الصحيحة بناءً على نوع الكائن. يسمح تعدد الأشكال للفئات الفرعية بأن يكون لها طرق بنفس الطريقة الخاصة بهم . أسماء كطرق في المستخدم نسميها superclasses.

The term polymorphism refers to an object's ability to take different forms. It is a powerful feature of object-oriented programming. In this section, we will look at two essential ingredients of polymorphic behavior:

1. The ability to define a method in a superclass, and then define a method with the same name in a subclass. When a subclass method has the same name as a superclass method, it is often said that the subclass method **overrides** the superclass method.

2. The ability to call the correct version of an overridden method, depending on the type of object that is used to call it. If a subclass object is used to call an overridden method, then the subclass's version of the method is the one that will execute. If a superclass object is used to call an overridden method, then the superclass's version of the method is the one that will execute.

يشير مصطلح تعدد الأشكال إلى قدرة الكائن على اتخاذ أشكال مختلفة. إنه قوي من سمات البرمجة الشبيهة: في هذا القسم ، سننظر إلى أمران أساسيين

مكونات السلوك متعدد الأشكال: 1. القدرة على تحديد طريقة في الطبقة العليا ، وطريقة التحديد مع نفس الطريقة الاسم في فئة فرعية . عندما يكون أسلوب فئة فرعية هو الاسم الذي يطلق على طريقة الطبقة الفائقة ، فإنه يكون كذلك كثيراً ما ذكر أن طريقة الطبقة الفرعية تتجاوز أسلوب الفصل الدراسي الفائق .

2-إمكانية استدعاء النسخة الصحيحة من طريقة تم تجاوزها ، اعتماداً على نوع الكائن المستخدم في تسميته . إذا تم استخدام كائن فئة فرعية لاستدعاء طريقة تم تجاوزها ، فعندئذ نسخة فئة فرعية من المنهجية التي سيتم تنفيذها. استدعاء طريقة تم تجاوزها ، ثم نسخة الطبقة العليا من الطريقة التي سوف

Program 11-9 (Lines 1 through 22 of animals.py)

```
1 # The Mammal class represents a generic mammal.  
2  
3 class Mammal:  
4  
5     # The __init__ method accepts an argument for  
6     # the mammal's species.  
7  
8     def __init__(self, species):  
9         self.__species = species  
10  
11    # The show_species method displays a message  
12    # indicating the mammal's species.  
13  
14    def show_species(self):  
15        print('I am a', self.__species)  
16  
17    # The make_sound method is the mammal's  
18    # way of making a generic sound.  
19  
20    def make_sound(self):  
21        print('Grrrrr')  
22
```

Program 11-9 (Lines 23 through 38 of animals.py)

```
23 # The Dog class is a subclass of the Mammal class.  
24  
25 class Dog(Mammal):  
26  
27     # The __init__ method calls the superclass's  
28     # __init__ method passing 'Dog' as the species.  
29  
30     def __init__(self):  
31         Mammal.__init__(self, 'Dog')  
32  
33     # The make_sound method overrides the superclass's  
34     # make_sound method.  
35  
36     def make_sound(self):  
37         print('Woof! Woof!')  
38
```

Program 11-9 (Lines 39 through 53 of animals.py)

```
39 # The Cat class is a subclass of the Mammal class.  
40  
41 class Cat(Mammal):  
42  
43     # The __init__ method calls the superclass's  
44     # __init__ method passing 'Cat' as the species.  
45  
46     def __init__(self):  
47         Mammal.__init__(self, 'Cat')  
48  
49     # The make_sound method overrides the superclass's  
50     # make_sound method.  
51  
52     def make_sound(self):  
53         print('Meow')
```

The is instance Function

Polymorphism gives us a great deal of flexibility when designing programs. For example,

look at the following function:

```
def show_mammal_info(creature):
    creature.show_species()
    creature.make_sound()
```

We can pass any object as an argument to this function, and as long as it has a `show_species` method and a `make_sound` method, the function will call those methods. In essence, we can pass any object that “is a” Mammal (or a subclass of Mammal) to the function. Program 11-10 demonstrates

وظيفة is instance

يمنح تعدد الأشكال قدرًا كبيرًا من المرونة عند تصميم البرامج مثل،

انظر إلى الوظيفة التالية:

```
def show_mammal_info(creature):
    creature.show_species()
    creature.make_sound()
```

حجة لهذه الوظيفة ، و `asan` كائن `we can pass any` `as long as it has a` `show_species` `method` `and a` `make_sound` `method` ،
ستستدعي الوظيفة هذين في الجوهر ، يمكننا تمرير أي كائن "هو" الثدييات .
أو فئة فرعية يوضح البرنامج ١١-١٠ (من الثدييات) للوظيفة

Program 11-10 (polymorphism_demo.py)

```
1 # This program demonstrates polymorphism.  
2  
3 import animals  
4  
5 def main():  
6     # Create a Mammal object, a Dog object, and  
7     # a Cat object.  
8     mammal = animals.Mammal('regular animal')  
9     dog = animals.Dog()  
10    cat = animals.Cat()  
11  
12    # Display information about each one.  
13    print('Here are some animals and')  
14    print('the sounds they make.')  
15    print('-----')  
16    show_mammal_info(mammal)  
17    print()  
18    show_mammal_info(dog)  
19    print()  
20    show_mammal_info(cat)  
21  
22    # The show_mammal_info function accepts an object  
23    # as an argument, and calls its show_species  
24    # and make_sound methods.  
25  
26    def show_mammal_info(creature):  
27        creature.show_species()  
28        creature.make_sound()  
29  
30    # Call the main function.  
31    main()
```

Program Output (continued)

Here are some animals and
the sounds they make.

I am a regular animal
Grrrrr
I am a Dog
Woof! Woof!
I am a Cat
Meow

```
10     cat = animals.Cat()
11
12     # Display information about each one.
13     print('Here are some animals and')
14     print('the sounds they make.')
15     print('-----')
16     show_mammal_info(mammal)
17     print()
18     show_mammal_info(dog)
19     print()
20     show_mammal_info(cat)
21     print()
22     show_mammal_info('I am a string')
23
24 # The show_mammal_info function accepts an object
25 # as an argument and calls its show_species
26 # and make_sound methods.
27
28 def show_mammal_info(creature):
29     if isinstance(creature, animals.Mammal):
30         creature.show_species()
31         creature.make_sound()
32     else:
33         print('That is not a Mammal!')
34
35 # Call the main function.
36 main()
```

استنتاج

- To avoid spaghetti code, we use object oriented programming which is built on four pillars. These pillars are
- Encapsulation.
- Abstraction.
- Inheritance.
- Polymorphism.
- **Encapsulation** - This is a grouping methods and properties that perform the same function into object.

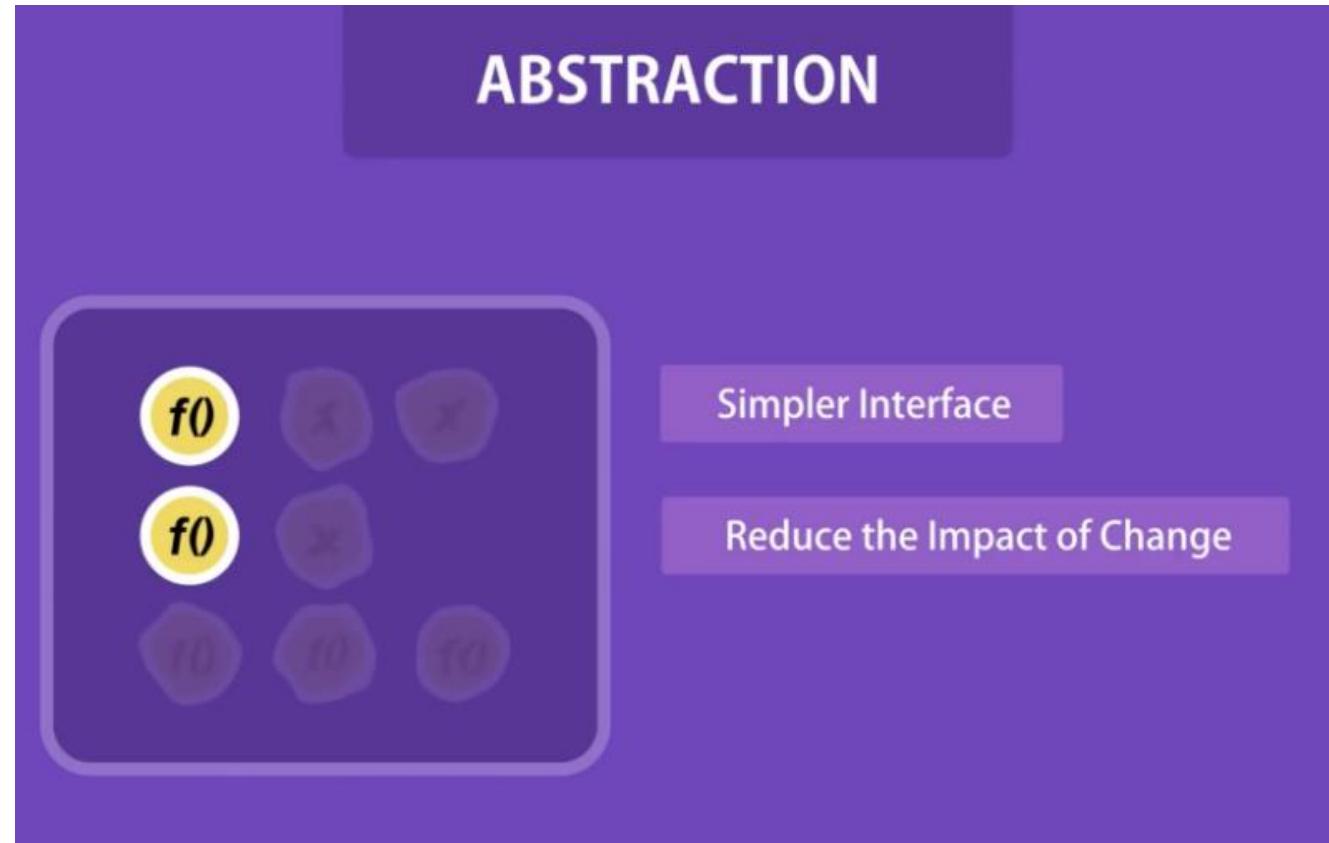
لتجنب كود السبايغىتى ، نستخدم البرمجة الشيئية وهي مبني على أربع ركائز . هذه الركائز

- التغليف.
- التجريد.
- ميراث.
- تعدد الأشكال

التغليف - هذه هي طرق التجميع والخصائص التي أداء نفس الوظيفة في الكائن

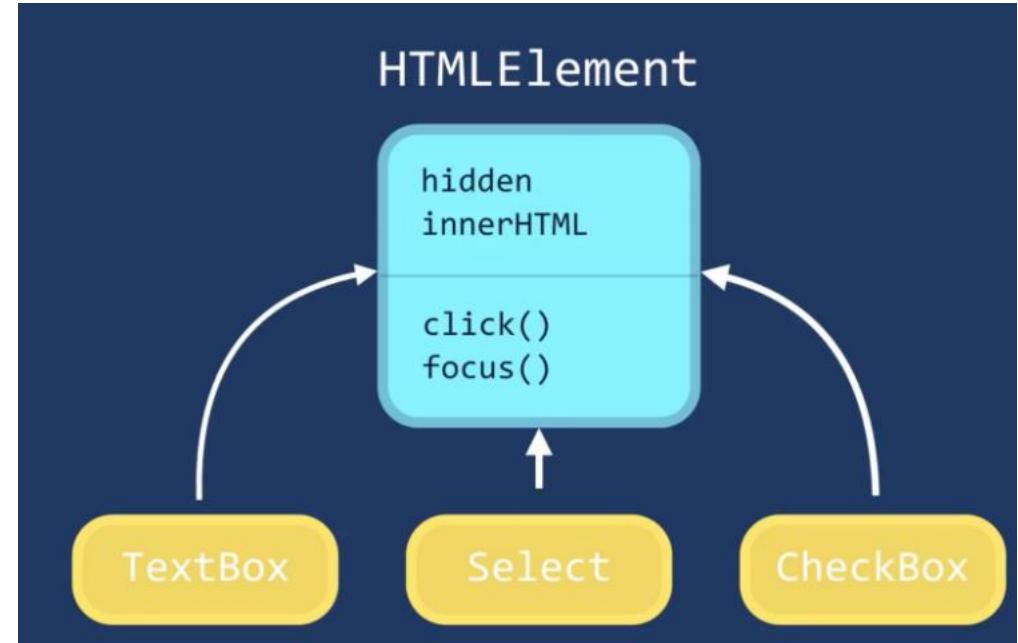
- **Abstraction** — A good example is a DVD player or a remote control. It has complex wiring on the inside and few buttons on the outside. All complexity is hidden from the user and only few buttons are made available to the user
- .

- **التجريد**
 على سبيل المثال هو مشغل DVD أو ملف
 جهاز التحكم . لديها
 الأسلام المعقدة من الداخل
 وعدد قليل من الأزرار على
 في الخارج . كل التعقيد
 مخفى عن المستخدم و
 يتم إجراء عدد قليل فقط من الأزرار
 متاح للمستخدم .



- **Inheritance** — Lets you remove redundancy in code. This is the situation where by objects can inherit from properties and attributes from other objects.

• الميراث - يتيح لك الإزالة التكرار في التعليمات البرمجية . هذا الوضع الذي يمكن أن ترث فيه الأشياء من الخصائص والسمات من أشياء أخرى.



Polymorphism — Lets you avoid and get round long if/else cases.

- تعدد الأشكال - يتيح لك تجنب حالات if / else والتغلب عليها.

| BENEFITS OF OOP | |
|-----------------|---|
| Encapsulation | Reduce complexity + increase reusability |
| Abstraction | Reduce complexity + isolate impact of changes |
| Inheritance | Eliminate redundant code |
| Polymorphism | Refactor ugly switch/case statements |