Classes and Methods: الفئات والطرق:

As explained, instance variables and methods are the constituents of classes. So far, the **Vehicle** class contains data, but no methods. Although data-only classes are perfectly valid, most classes will have methods. Methods are subroutines that manipulate the data defined by the class and, in many cases, provide access to that data.

كما هو موضح ، فإن متغيرات الحالة والأساليب هي مكونات الفئات. حتى الآن ، تحتوي فئة السيارة على بيانات ، ولكن لا توجد طرق. على الرغم من أن فئات البيانات فقط صالحة تمامًا ، إلا أن معظم الفئات لها طرق. الأساليب هي إجراءات فرعية تتعامل مع البيانات المحددة بواسطة الفئة ، وفي كثير من الحالات ، توفر الوصول إلى تلك البيانات.

In most cases, other parts of your program will interact with a class through its methods. A method contains one or more statements. In well-written Java code, each method performs only one task. Each method has a name, and it is this name that is used to call the method.

في معظم الحالات ، ستتفاعل أجزاء أخرى من برنامجك مع الفصل من خلال طرقه. تحتوي الطريقة على جملة واحدة أو أكثر. في كود Java المكتوب جيدًا ، تؤدي كل طريقة مهمة واحدة فقط. كل طريقة لها اسم ، وهذا الاسم هو الذي يستخدم لاستدعاء الطريقة.

In general, you can give a method whatever name you please. However, remember that **main( )** is reserved for the method that begins execution of your program. Also, don't use Java's keywords for method names.

بشكل عام ، يمكنك إعطاء طريقة مهما كان الاسم الذي تريده. ومع ذلك ، تذكر أن main () محجوزة للطريقة التي تبدأ بتنفيذ برنامجك. أيضًا ، لا تستخدم الكلمات الرئيسية لجافا لأسماء الطرق.

A method will have parentheses after its name. For example, if a method's name is **getval**, it will be written **getval( )** when its name is used in a sentence. This notation will help you distinguish variable names from method names.

The general form of a method is shown here:

سيكون للطريقة أقواس بعد اسمها. على سبيل المثال ، إذا كان اسم العملية هو getval ، فسيتم كتابتها getval () عند استخدام اسمها في الجملة. ستساعدك هذه الملاحظة على تمييز أسماء المتغيرات من أسماء الطرق.

الشكل العام للطريقة معروض هنا:

```
ret-type name( parameter-list ) {
// body of method
}
```

Here, *ret-type* specifies the type of data returned by the method. This can be any valid type, including class types that you create. If the method does not return a value, its return type must be **void**. The name of the method is specified by *name.* This can be any legal identifier other than those already used by other items within the current scope. The *parameter-list* is a sequence of type and identifier pairs

separated by commas. Parameters are essentially variables that receive the value of the *arguments* passed to the method when it is called.
هنا ، يحدد نوع ret نوع البيانات التي يتم إرجاعها بواسطة الطريقة. يمكن أن يكون هذا أي نوع صالح ، بما في ذلك أنواع الفئات التي تقوم بإنشائها. إذا لم تُرجع الطريقة قيمة ، فيجب أن يكون نوع الإرجاع الخاص بها باطلاً. يتم تحديد اسم الطريقة بالاسم. يمكن أن يكون هذا أي معرف قانوني بخلاف تلك المستخدمة بالفعل من قبل العناصر الأخرى داخل النطاق الحالي. قائمة المعلمات هي سلسلة من أزواج النوع والمعرف مفصولة بفواصل. المعلمات هي في الأساس متغيرات تتلقى قيمة الوسائط التي تم تمريرها إلى الطريقة عندما يتم استدعاؤها.

If the method has no parameters, the parameter list will be empty.
إذا كانت الطريقة لا تحتوي على معلمات ، فستكون قائمة المعلمات فارغة.

Adding a Method to the Vehicle Class إضافة طريقة إلى فئة المركبة

As just explained, the methods of a class typically manipulate and provide access to the data of the class. With this in mind, recall that **main( )** in the preceding examples computed the range of a vehicle by multiplying its fuel consumption rate by its fuel capacity. While technically correct, this is not the best way to handle this computation. The calculation of a vehicle's range is something that is best handled by the **Vehicle** class itself. The reason for this conclusion is easy to understand: the range of a vehicle is dependent upon the capacity of the fuel tank and the rate of fuel consumption, and both of these quantities are encapsulated by **Vehicle**. By adding a method to **Vehicle** that computes the range, you are enhancing its object-oriented structure. To add a method to **Vehicle**, specify it within **Vehicle**'s declaration. For example, the following version of **Vehicle** contains a method called **range( )** that displays the range of the vehicle.
كما أوضحنا للتو ، فإن طرق الفصل عادةً ما تتلاعب وتوفر الوصول إلى بيانات الفصل. مع وضع ذلك في الاعتبار ، تذكر أن main () في الأمثلة السابقة تحسب نطاق السيارة بضرب معدل استهلاك الوقود في قدرتها على الوقود. على الرغم من كونها صحيحة من الناحية الفنية ، إلا أن هذه ليست أفضل طريقة للتعامل مع هذا الحساب. يعد حساب نطاق السيارة أمرًا يتم التعامل معه بشكل أفضل بواسطة فئة السيارة نفسها. يسهل فهم سبب هذا الاستنتاج: يعتمد نطاق السيارة على سعة خزان الوقود ومعدل استهلاك الوقود ، ويتم تغليف هاتين الكميتين بواسطة السيارة. من خلال إضافة طريقة إلى المركبة التي تحسب النطاق ، فإنك تعزز هيكلها الموجه للكائنات. لإضافة طريقة إلى السيارة ، حددها في بيان المركبة. على سبيل المثال ، يحتوي الإصدار التالي من السيارة على طريقة تسمى النطاق () التي تعرض نطاق السيارة.

```
// Add range to Vehicle.إضافة نطاق إلى السيارة // .
class Vehicle {
int passengers; // number of passengersعدد الركاب
int fuelcap; // fuel capacity in gallonsسعة الوقود بالغالون

int mpg; // fuel consumption in miles per gallon
                                استهلاك الوقود بالأميال للغالون الواحد //
// Display the range.اعرض النطاق .
void range() {
System.out.println("Range is " + fuelcap * mpg);
}
```

```
}
class AddMeth {
public static void main(String args[]) {
Vehicle minivan = new Vehicle();
Vehicle sportscar = new Vehicle();
int range1, range2;
// assign values to fields in minivan
                                    // تعيين القيم إلى الحقول في شاحنة صغيرة
minivan.passengers = 7;
minivan.fuelcap = 16;
minivan.mpg = 21;
// assign values to fields in sportscar
                                  // تعيين قيم للمجالات في السيارات الرياضية
sportscar.passengers = 2;
sportscar.fuelcap = 14;
sportscar.mpg = 12;
System.out.print("Minivan can carry " + minivan.passengers +
". ");
minivan.range(); // display range of minivanنطاق عرض الميني فان
System.out.print("Sportscar can carry " + sportscar.passengers +
". ");
sportscar.range(); // display range of sportscar. عرض مجموعة من
السيارات الرياضية.
}
}
```

This program generates the following output::يولد هذا البرنامج المخرجات التالية

```
Minivan can carry 7. Range is 336 النطاق 7. يمكن أن تحمل الميني فان

Sportscar can carry 2. Range is 168 يمكن تحمل Sportscar 2 المدى هو .
```

Let's look at the key elements of this program, beginning with the **range( )** method itself.

لنلقِ نظرة على العناصر الأساسية لهذا البرنامج ، بدءًا من طريقة range () نفسها.

The first line of **range( )** is السطر الأول من النطاق () هو

```
void range() {
```

The **range( )** method is contained within the **Vehicle** class. Notice that **fuelcap** and **mpg** are used directly, without the dot operator.

يتم تضمين طريقة المدى () في فئة السيارة. لاحظ أن غطاء الوقود و mpg يتم استخدامهما مباشرة ، دون عامل التشغيل النقطي.

This line declares a method called **range** that has no parameters. Its return type is **void**. Thus, **range( )** does not return a value to the caller.

يعلن هذا السطر عن طريقة تسمى النطاق لا تحتوي على معلمات. نوع عودته باطل. وبالتالي ، فإن النطاق () لا يُرجع قيمة إلى المتصل.

The body of **range( )** consists solely of this line:يتكون جسم النطاق () فقط من هذا الخط :

```
System.out.println("Range is " + fuelcap * mpg);
```

This statement displays the range of the vehicle by multiplying **fuelcap** by **mpg**. Since each object of type **Vehicle** has its own copy of **fuelcap** and **mpg**, when **range( )** is called, the range computation uses the calling object's copies of those variables.
The **range( )** method ends when its closing curly brace is encountered. This causes program control to transfer back to the caller.
يعرض هذا البيان نطاق السيارة بضرب غطاء الوقود في ميلا في الغالون. نظرًا لأن كل كائن من نوع مركبة له نسخته الخاصة من غطاء الوقود و mpg ، عند استدعاء النطاق () ، يستخدم حساب النطاق نسخ كائن الاستدعاء من تلك المتغيرات.
تنتهي طريقة النطاق () عند مصادفة قوس الإغلاق المتعرج. يؤدي هذا إلى تحويل التحكم في البرنامج مرة أخرى إلى المتصل.

Next, look closely at this line of code from inside **main( )**:
بعد ذلك ، انظر عن كثب إلى هذا السطر من التعليمات البرمجية من داخل main ():

```
minivan.range();
```

This statement invokes the **range( )** method on **minivan**. That is, it calls **range( )** relative to the **minivan** object, using the object's name followed by the dot operator. When a method is called, program control is transferred to the method. When the method terminates, control is transferred back to the caller, and execution resumes with the line of code following the call. In this case, the call to **minivan.range( )** displays the range of the vehicle defined by **minivan**.
تستدعي هذه العبارة طريقة النطاق () على الميني فان. أي أنه يستدعي النطاق () بالنسبة إلى كائن الميني فان ، باستخدام اسم الكائن متبوعًا بمعامل النقطة. عندما يتم استدعاء طريقة ، يتم نقل التحكم في البرنامج إلى الطريقة. عندما تنتهي الطريقة ، يتم نقل التحكم مرة أخرى إلى المتصل ، ويستأنف التنفيذ مع سطر التعليمات البرمجية الذي يلي المكالمة. في هذه الحالة ، يعرض استدعاء minivan.range () نطاق السيارة المحدد بواسطة الميني فان.
In similar fashion, the call to **sportscar.range( )** displays the range of the vehicle defined by **sportscar**. Each time **range( )** is invoked, it displays the range for the specified object.
بطريقة مماثلة ، تعرض المكالمة إلى نطاق السيارة الرياضية () نطاق السيارة المحدد بواسطة السيارة الرياضية. في كل مرة يتم استدعاء نطاق زمني () ، فإنه يعرض النطاق الخاص بالكائن المحدد.

There is something very important to notice inside the **range( )** method: the instance variables **fuelcap** and **mpg** are referred to directly, without preceding them with an object name or the dot operator. When a method uses an instance variable that is defined by its class, it does so directly, without explicit reference to an object and without use of the dot operator. This is easy to understand if you think about it. A method is always invoked relative to some object of its class. Once this invocation

has occurred, the object is known. Thus, within a method, there is no need to specify the object a second time. This means that **fuelcap** and **mpg** inside **range( )** implicitly refer to the copies of those variables found in the object that invokes **range( )**.
هناك شيء مهم للغاية يجب ملاحظته داخل طريقة النطاق (): تتم الإشارة إلى متغيرات المثيل Fuelcap و mpg مباشرة ، دون أن يسبقهما اسم كائن أو عامل النقطة. عندما تستخدم طريقة متغير حالة يتم تحديده بواسطة فئته ، فإنه يفعل ذلك مباشرة ، دون إشارة صريحة إلى كائن ودون استخدام عامل التشغيل النقطي. من السهل فهم هذا إذا فكرت في الأمر. يتم استدعاء عملية دائمًا بالنسبة إلى كائن ما في فئتها. بمجرد حدوث هذا الاستدعاء ، يُعرف الكائن. وبالتالي ، ضمن طريقة ما ، ليست هناك حاجة لتحديد الكائن مرة ثانية. هذا يعني أن غطاء الوقود و mpg داخل النطاق () يشيران ضمنًا إلى نسخ تلك المتغيرات الموجودة في الكائن الذي يستدعي النطاق ().

Returning from a Methodالعودة من طريقة

In general, there are two conditions that cause a method to return—first, as the **range( )** method in the preceding example shows, when the method's closing curly brace is encountered. The second is when a **return** statement is executed. There are two forms of **return**—one for use in **void** methods (those that do not return a value) and one for returning values. The first form is examined here. The next section explains how to return values.
بشكل عام ، هناك شرطان يتسببان في إرجاع طريقة ما - أولاً ، كما توضح طريقة النطاق () في المثال السابق ، عندما يتم مصادفة قوس الإغلاق المتعرج الخاص بالطريقة. والثاني هو عندما يتم تنفيذ تعليمة العودة. هناك نوعان من أشكال الإرجاع - أحدهما للاستخدام في الطرق الباطلة (تلك التي لا تُرجع قيمة) والآخر لإرجاع القيم. يتم فحص النموذج الأول هنا. يوضح القسم التالي كيفية إرجاع القيم.

In a **void** method, you can cause the immediate termination of a method by using this form of **return**:
في طريقة باطلة ، يمكنك التسبب في الإنهاء الفوري للطريقة باستخدام هذا الشكل من الإرجاع:

```
return ;
```

When this statement executes, program control returns to the caller, skipping any remaining code in the method. For example, consider this method:
عند تنفيذ هذه العبارة ، يعود التحكم في البرنامج إلى المتصل ، متخطياً أي رمز متبقي في الطريقة. على سبيل المثال ، ضع في اعتبارك هذه الطريقة:

```
void myMeth() {
int i;
for(i=0; i<10; i++) {
if(i == 5) return; // stop at 5
System.out.println();
 }
}
```

Here, the **for** loop will only run from 0 to 5, because once **i** equals 5, the method returns. It is permissible to have multiple return statements in a method, especially when there are two or more routes out of it. For example:
هنا ، لن تعمل الحلقة for إلا من 0 إلى 5 ، لأنه بمجرد أن يساوي 5 i ، تعود الطريقة. يجوز الحصول على عبارات إرجاع متعددة في طريقة ما ، خاصةً عندما يكون هناك مساران أو أكثر للخروج منه. على سبيل المثال:

```
void myMeth() {
// ...
if(done) return;
// ...
if(error) return;
}
```

Here, the method returns if it is done or if an error occurs. Be careful, however, because having too many exit points in a method can destructure your code; so avoid using them casually. A well-designed method has well-defined exit points.
To review: a **void** method can return in one of two ways—its closing curly brace is reached, or a **return** statement is executed.
هنا ، تعود الطريقة إذا تم القيام بها أو في حالة حدوث خطأ. كن حذرًا ، على الرغم من ذلك ، لأن وجود عدد كبير جدًا من نقاط الخروج في طريقة ما يمكن أن يدمر التعليمات البرمجية الخاصة بك ؛ لذا تجنب استخدامها بشكل عرضي. تتميز الطريقة المصممة جيدًا بنقاط خروج محددة جيدًا.
للمراجعة: يمكن أن تعود طريقة الفراغ بإحدى طريقتين - يتم الوصول إلى قوس الإغلاق المتعرج ، أو تنفيذ تعليمة الإرجاع.

Returning a Value إرجاع قيمة

You can use a return value to improve the implementation of **range( )**. Instead of displaying the range, a better approach is to have **range( )** compute the range and return this value. Among the advantages to this approach is that you can use the value for other calculations. The following example modifies **range( )** to return the range rather than displaying it.
يمكنك استخدام القيمة المرجعة لتحسين تنفيذ النطاق (). بدلاً من عرض النطاق ، تتمثل الطريقة الأفضل في جعل النطاق () يحسب النطاق ويعيد هذه القيمة. من بين مزايا هذا الأسلوب أنه يمكنك استخدام القيمة لعمليات حسابية أخرى. يعدل المثال التالي النطاق () لإرجاع النطاق بدلاً من عرضه.

```
// Use a return value.استخدم قيمة الإرجاع // .
class Vehicle {
int passengers; // number of passengers// عدد الركاب
int fuelcap; // fuel capacity in gallons// سعة الوقود بالجالون
int mpg; // fuel consumption in miles per gallon
                                                    // استهلاك الوقود بالأميال للغالون الواحد
// Return the range.إرجاع النطاق .
int range() {
return mpg * fuelcap;
}
}
class RetMeth {
public static void main(String args[]) {
```

```
Vehicle minivan = new Vehicle();
Vehicle sportscar = new Vehicle();
int range1, range2;
// assign values to fields in minivan
                                                    // تعيين القيم إلى الحقول في شاحنة صغيرة
minivan.passengers = 7;
minivan.fuelcap = 16;
minivan.mpg = 21;
// assign values to fields in sportscar
                                                    // تعيين قيم للمجالات في السيارات الرياضية
sportscar.passengers = 2;
sportscar.fuelcap = 14;
sportscar.mpg = 12;
// get the ranges// احصل على النطاقات
range1 = minivan.range();
range2 = sportscar.range();
System.out.println("Minivan can carry " + minivan.passengers +
" with range of " + range1 + " Miles");
System.out.println("Sportscar can carry " + sportscar.passengers +
" with range of " + range2 + " miles");
}
}
```

The output is shown here:

```
Minivan can carry 7 with range of 336 Miles
Sportscar can carry 2 with range of 168 miles
```

In the program, notice that when **range( )** is called, it is put on the right side of an assignment statement. On the left is a variable that will receive the value returned by **range( )**.

يظهر الإخراج هنا:
يمكن أن تحمل الميني فان 7 بمدى يصل إلى 336 ميلاً
يمكن أن تحمل Sportscar 2 بمدى 168 ميلاً
في البرنامج ، لاحظ أنه عند استدعاء النطاق () ، يتم وضعه على الجانب الأيمن من ملف
بيان التنازل. على اليسار يوجد متغير يتلقى القيمة المعادة بواسطة النطاق ().

Thus, after

```
range1 = minivan.range();
```

executes, the range of the **minivan** object is stored in **range1**.

Notice that **range( )** now has a return type of **int**. This means that it will return an integer value to the caller. The return type of a method is important because the type of data returned by a method must be compatible with the return type specified by the method. Thus, if you want a method to return data of type **double**, its return type must be type **double**.

في حالة التنفيذ ، يتم تخزين نطاق كائن الميني فان في النطاق 1.
لاحظ أن النطاق () يحتوي الآن على نوع إرجاع من int. هذا يعني أنه سيعيد قيمة عدد صحيح للمتصل. يعد نوع الإرجاع للطريقة مهمًا لأن نوع البيانات التي يتم إرجاعها بواسطة طريقة ما يجب أن يكون متوافقًا مع نوع

الإرجاع المحدد بواسطة الطريقة. وبالتالي ، إذا كنت تريد طريقة ما لإرجاع بيانات من النوع مزدوج ، فيجب أن يكون نوع الإرجاع الخاص بها من النوع double.

Specifically, there is no need for the **range1** or **range2** variables. A call to **range( )** can be used in the **println( )** statement directly, as shown here:
على وجه التحديد ، ليست هناك حاجة لمتغيرات range1 أو range2. يمكن استخدام استدعاء النطاق () في عبارة println () مباشرةً ، كما هو موضح هنا:

```
System.out.println("Minivan can carry " + minivan.passengers +
" with range of " + minivan.range() + " Miles");
```

In this case, when **println( )** is executed, **minivan.range( )** is called automatically and its value will be passed to **println( )**. Furthermore, you can use a call to **range( )** whenever the range of a **Vehicle** object is needed. For example, this statement compares the ranges of two vehicles:
في هذه الحالة ، عند تنفيذ println () ، يتم استدعاء minivan.range () تلقائيًا وسيتم تمرير قيمتها إلى println (). علاوة على ذلك ، يمكنك استخدام استدعاء النطاق () كلما دعت الحاجة إلى نطاق جسم السيارة. على سبيل المثال ، يقارن هذا البيان نطاقات سيارتين:

```
if(v1.range() > v2.range()) System.out.println("v1 has greater
range");
```

<u>باستخدام المعلماتUsing Parameters</u>

It is possible to pass one or more values to a method when the method is called. As explained, a value passed to a method is called an *argument.* Inside the method, the variable that receives the argument is called a *parameter*. Parameters are declared inside the parentheses that follow the method's name. The parameter declaration syntax is the same as that used for variables.
من الممكن تمرير قيمة واحدة أو أكثر إلى طريقة عندما يتم استدعاء الطريقة. كما هو موضح ، تسمى القيمة التي يتم تمريرها إلى طريقة الوسيطة. داخل العملية ، يُطلق على المتغير الذي يستقبل الوسيطة معلمة. يتم التصريح عن المعلمات داخل الأقواس التي تتبع اسم الطريقة. صيغة إعلان المعلمة هي نفسها المستخدمة مع المتغيرات.

A parameter is within the scope of its method, and aside from its special task of receiving an argument, it acts like any other local variable.
تقع المعلمة ضمن نطاق طريقتها ، وبغض النظر عن مهمتها الخاصة لتلقي وسيطة ، فإنها تعمل مثل أي متغير محلي آخر.

Here is a simple example that uses a parameter. Inside the **ChkNum** class, the method **isEven( )** returns **true** if the value that it is passed is even. It returns **false** otherwise. Therefore, **isEven( )** has a return type of **boolean**.
فيما يلي مثال بسيط يستخدم معلمة. داخل صنف ChkNum ، تُرجع الطريقةEven () صحيحًا إذا كانت القيمة التي تم تمريرها زوجية. ترجع كاذبة على خلاف ذلك. لذلك ، يحتوي isEven () على نوع إرجاع منطقي.

```
// A simple example that uses a parameter.
```

// مثال بسيط يستخدم المعامل.

```
class ChkNum {
// return true if x is even// إرجاع صحيحًا إذا كان x زوجيًا
boolean isEven(int x) {
if((x%2) == 0) return true;
else return false;
}
}
class ParmDemo {
public static void main(String args[]) {
ChkNum e = new ChkNum();
if(e.isEven(10)) System.out.println("10 is even.");
if(e.isEven(9)) System.out.println("9 is even.");
if(e.isEven(8)) System.out.println("8 is even.");
}
```

Here is the output produced by the program:هنا هو الناتج الناتج عن البرنامج :

```
10 is even.
8 is even.
```

In the program, **isEven( )** is called three times, and each time a different value is passed. Let's look at this process closely. First, notice how **isEven( )** is called. The argument is specified between the parentheses. When **isEven( )** is called the first time, it is passed the value 10. Thus, when **isEven( )** begins executing, the parameter **x** receives the value 10. In the second call, 9 is the argument, and **x**, then, has the value 9. In the third call, the argument is 8, which is the value that **x** receives. The point is that the value passed as an argument when **isEven( )** is called is the value received by its parameter, **x**.

في البرنامج ، يتم استدعاء isEven () ثلاث مرات ، وفي كل مرة يتم تمرير قيمة مختلفة. دعونا نلقي نظرة على هذه العملية عن كثب. أولاً ، لاحظ كيف يتم استدعاء Even (). يتم تحديد الوسيطة بين قوسين. عندما يتم استدعاء isEven () في المرة الأولى ، يتم تمرير القيمة 10. وهكذا ، عندما يبدأ تنفيذ isEven () ، يتلقى المعامل x القيمة 10. في الاستدعاء الثاني ، 9 هي الوسيطة ، و x ، إذن ، لديها القيمة 9. في الاستدعاء الثالث ، تكون الوسيطة 8 ، وهي القيمة التي يتلقاها x. النقطة هي أن القيمة التي يتم تمريرها كوسيطة عندما يتم استدعاء isEven () هي القيمة التي تتلقاها المعلمة ، x.

Adding a Parameterized Method to Vehicle إضافة طريقة معلمة للمركبة

You can use a parameterized method to add a new feature to the **Vehicle** class: the ability to compute the amount of fuel needed for a given distance. This new method is called **fuelneeded( )**. This method takes the number of miles that you want to drive and returns the number of gallons of gas required. The **fuelneeded( )** method is defined like this:

يمكنك استخدام طريقة معلمات لإضافة ميزة جديدة إلى فئة السيارة: القدرة على حساب كمية الوقود اللازمة لمسافة معينة. تسمى هذه الطريقة الجديدة بالوقود (). تأخذ هذه الطريقة عدد الأميال التي تريد قيادتها وتعيد عدد غالونات الغاز المطلوبة. يتم تعريف طريقة Fuelneeded () على النحو التالي:

```
double fuelneeded(int miles) {
return (double) miles / mpg;
}
```

Notice that this method returns a value of type **double**. This is useful since the amount of fuel needed for a given distance might not be an even number.
The entire **Vehicle** class that includes **fuelneeded( )** is shown here:

لاحظ أن هذه الطريقة تُرجع قيمة من النوع double. هذا مفيد لأن كمية الوقود اللازمة لمسافة معينة قد لا تكون عددًا زوجيًا.

يتم عرض فئة السيارة بالكامل التي تتضمن الوقود المطلوب () هنا:

```
class Vehicle {
int passengers; // number of passengersعدد الركاب
int fuelcap; // fuel capacity in gallonsسعة الوقود بالغالون
int mpg; // fuel consumption in miles per gallon
                                                    // استهلاك الوقود بالأميال للغالون الواحد
// Return the range.إرجاع النطاق .
int range() {
return mpg * fuelcap;
}
// Compute fuel needed for a given distance.
                                                    // احسب الوقود المطلوب لمسافة معينة.
double fuelneeded(int miles) {
return (double) miles / mpg;
 }
}
class CompFuel {
public static void main(String args[]) {
Vehicle minivan = new Vehicle();
Vehicle sportscar = new Vehicle();
double gallons;
int dist = 252;
// assign values to fields in minivanتعيين القيم إلى الحقول في شاحنة صغيرة
minivan.passengers = 7;
minivan.fuelcap = 16;
minivan.mpg = 21;
// assign values to fields in sportscarتعيين قيم للمجالات في السيارات الرياضية
sportscar.passengers = 2;
sportscar.fuelcap = 14;
sportscar.mpg = 12;
gallons = minivan.fuelneeded(dist);
System.out.println("To go " + dist + " miles minivan needs " +
gallons + " gallons of fuel.");
gallons = sportscar.fuelneeded(dist);
System.out.println("To go " + dist + " miles sportscar needs " +
gallons + " gallons of fuel.");
}
}
```

The output from the program is shown here:يظهر ناتج البرنامج هنا :

```
To go 252 miles minivan needs 12.0 gallons of fuel.
To go 252 miles sportscar needs 21.0 gallons of fuel.
```

للذهاب 252 ميلا الميني فان يحتاج 12.0 جالون من الوقود.
للذهاب 252 ميلاً ، تحتاج السيارة الرياضية إلى 21.0 جالونًا من الوقود.

Translated by :- M.J