Object-Oriented Languages: اللغات الشيئية:

There are several object-oriented programming languages available to choose from, including Smalltalk, Eiffel, C++, Objective C, Objective Pascal, Java, Ada, and even a version of Lisp. There are two clear marketplace winners, C++ and Java.

Today, Java is the developing object-oriented language of choice for many programmers and software projects.

هناك العديد من لغات البرمجة الموجهة للكائنات المتاحة للاختيار من بينها ، بما في ذلك Smalltalk و Eiffel و C ++ و Objective C و Objective Pascal و Java و Ada وحتى إصدار من Lisp. يوجد فائزان واضحان في السوق ، C ++ و Java.

اليوم ، تعد Java هي اللغة التطويرية الموجهة للكائنات المختارة للعديد من المبرمجين ومشاريع البرمجيات.

Overview of the Java Language نظرة عامة على لغة جافا

Java is an object-oriented programming language developed by Sun Microsystems. Originally, the developers of Java intended to use C++ for their software development. But they needed a language that could execute on different sets of computer chips to accommodate the ever-changing consumer electronics market. So they decided to design their own language which would be independent of the underlying hardware.

Java هي لغة برمجة موجهة للكائنات طورتها شركة Sun Microsystems. في الأصل ، كان مطورو Java يعتزمون استخدام C ++ لتطوير برامجهم. لكنهم كانوا بحاجة إلى لغة يمكن تنفيذها على مجموعات مختلفة من رقائق الكمبيوتر لاستيعاب سوق الإلكترونيات الاستهلاكية المتغير باستمرار. لذلك قرروا تصميم لغتهم الخاصة والتي ستكون مستقلة عن الأجهزة الأساسية.

It allows a user to receive software from a remote system and execute it on a local system, regardless of the underlying hardware or operating system. An interpreter and runtime are called the Java Virtual Machine which insulates the software from the underlying hardware.

يسمح للمستخدم بتلقي البرامج من نظام بعيد وتنفيذها على نظام محلي ، بغض النظر عن الأجهزة الأساسية أو نظام التشغيل. يُطلق على المترجم الفوري ووقت التشغيل اسم Java Virtual Machine الذي يعزل البرنامج عن الأجهزة الأساسية.

Unlike more traditional languages, Java source code does not get translated into the machine instructions for a particular computer platform. Instead, Java source code (.java) is compiled into an intermediate form called bytecodes which are stored in a .class file. These bytecodes can be executed on any computer system that implements a Java Virtual Machine (JVM). This portability is perhaps one of the most compelling features of the Java language, from a commercial perspective. In the current era of cross-platform application development, any tool that allows programmers to write code once and execute it on many platforms is going to get attention.

على عكس المزيد من اللغات التقليدية ، لا يتم ترجمة شفرة مصدر Java إلى تعليمات الجهاز لمنصة كمبيوتر معينة. بدلاً من ذلك ، يتم تجميع شفرة مصدر Java (.java) في نموذج وسيط يسمى bytecodes يتم تخزينه في ملف .class. يمكن تنفيذ رموز البايت هذه على أي نظام كمبيوتر يقوم بتنفيذ Java Virtual Machine (JVM). ربما تكون قابلية النقل هذه واحدة من أكثر الميزات إلحاحًا للغة Java ، من منظور تجاري. في العصر الحالي لتطوير التطبيقات عبر الأنظمة الأساسية ، فإن أي أداة تسمح للمبرمجين بكتابة التعليمات البرمجية مرة واحدة وتنفيذها على العديد من الأنظمة الأساسية ستلفت الانتباه.

Java Language Characteristics خصائص لغة جافا

- The portable, interpreted nature of Java impacts its performance. While the performance of interpreted Java code is better than scripting languages and fast enough for interactive applications, it is slower than traditional languages whose source code is compiled directly into the machine code for a particular machine. To improve performance, Just-In-Time compilers (JITs) have been developed. A JIT compiler runs concurrently with the Java Virtual Machine and determines what pieces of Java code are called most often.

- تؤثر طبيعة Java المحمولة والمفسرة على أدائها. على الرغم من أن أداء كود Java المفسر أفضل من لغات البرمجة النصية وسريعًا بدرجة كافية للتطبيقات التفاعلية ، إلا أنه أبطأ من اللغات التقليدية التي يتم تجميع كود مصدرها مباشرة في كود الآلة لجهاز معين. لتحسين الأداء ، تم تطوير برامج التحويل البرمجي Just-In-Time (JITs). يعمل مترجم JIT بشكل متزامن مع Java Virtual Machine ويحدد أجزاء كود Java التي يطلق عليها غالبًا.

- The bytecode portability is what enables Java to be transported across a network and executed on any target computer system. Java applets are small Java programs designed to be included in an HTML (HyperText Markup Language) Web document. HTML tags specify the name of the Java applet +and its Uniform Resource Locator (URL). The URL is the location on the Internet at which the applet bytecodes reside. When a Java-enabled Web browser displays an HTML document containing an applet tag, the Java bytecodes are downloaded from the specified location and the Java Virtual Machine interprets or executes the bytecodes. Java applets are what enable Web pages to contain animated graphics and interactive content.

- قابلية نقل الرمز الثانوي هي ما يمكّن من نقل Java عبر شبكة وتنفيذها على أي نظام كمبيوتر مستهدف. تطبيقات Java الصغيرة هي برامج Java صغيرة مصممة ليتم تضمينها في مستند ويب بتنسيق HTML (لغة ترميز النص التشعبي). تحدد علامات HTML اسم برنامج Java الصغير + ومحدد موقع المعلومات (URL) الخاص به. عنوان URL هو الموقع على الإنترنت حيث توجد أكواد بايت صغيرة. عندما يعرض مستعرض ويب ممكّن لـ Java مستند HTML يحتوي على علامة تطبيق صغير ، يتم تنزيل أكواد Java bytecodes من الموقع المحدد ويفسر Java Virtual Machine أو ينفذ الرموز البايتية. تطبيقات Java الصغيرة هي التي تمكّن صفحات الويب من احتواء رسومات متحركة ومحتوى تفاعلي.

- Because Java applets can be downloaded from any system, security mechanisms exist within the Java Virtual Machine to protect against malicious or errant applets.

- نظرًا لأنه يمكن تنزيل تطبيقات Java الصغيرة من أي نظام ، توجد آليات أمان داخل Java Virtual Machine للحماية من التطبيقات الخبيثة أو الضالة.

- Java is an object-oriented programming language, borrowing heavily from Smalltalk, Objective C, and C++. It is characterized by many as a better, safer C++. Java uses C++ syntax and is readily accessible to the large existing C++ development community.
- Java هي لغة برمجة موجهة للكائنات ، تقترض بشكل كبير من Smalltalk و Objective C و C ++. يتميز من قبل الكثيرين بأنه C ++ أفضل وأكثر أمانًا. يستخدم Java بناء جملة C ++ ويمكن الوصول إليه بسهولة من قبل مجتمع تطوير C ++ الكبير الحالي.

- Java, however, does not drag along the legacy of C. It does not allow global variables, functions, or procedures. With the exception of a few primitive data types like integers or floating-point numbers, everything in Java is an object.

- ومع ذلك ، فإن Java لا تسحب إرث C. فهي لا تسمح بالمتغيرات أو الوظائف أو الإجراءات العالمية. باستثناء بعض أنواع البيانات البدائية مثل الأعداد الصحيحة أو أرقام الفاصلة العائمة ، فإن كل شيء في Java هو كائن.

- Object references are not pointers, and pointer manipulation is not allowed. This contributes to the general robustness of Java programs since pointer operations tend to be particularly nasty and bug-prone. Java also manages memory itself, thereby avoiding problems with allocation and deallocation of objects. It does not allow multiple inheritance like C++ does, but supports another type of reuse through the use of formal interface definitions.

- مراجع الكائنات ليست مؤشرات ، ولا يُسمح بمعالجة المؤشر. يساهم هذا في القوة العامة لبرامج Java نظرًا لأن عمليات المؤشر تميل إلى أن تكون سيئة بشكل خاص وعرضة للأخطاء. تدير Java أيضًا الذاكرة نفسها ، وبالتالي تتجنب مشاكل تخصيص الكائنات وإلغاء تخصيصها. لا يسمح بالميراث المتعدد مثل C ++ ، ولكنه يدعم نوعًا آخر من إعادة الاستخدام من خلال استخدام تعريفات الواجهة الرسمية.

- Java is similar enough to C and C++ that it already feels familiar to most of the existing programming community. But it is different enough in important ways

(memory management and cross-platform portability) that it is worth it for programmers to switch to a new language.

- تشبه Java بدرجة كافية C و C ++ لدرجة أنها تشعر بالفعل بأنها مألوفة لمعظم مجتمع البرمجة الحالي. لكن الأمر مختلف بما فيه الكفاية من نواحٍ مهمة (إدارة الذاكرة وقابلية النقل عبر الأنظمة الأساسية) لدرجة أن الأمر يستحق أن يتحول المبرمجون إلى لغة جديدة.

The Java Programming Language Basics أساسيات لغة برمجة جافا

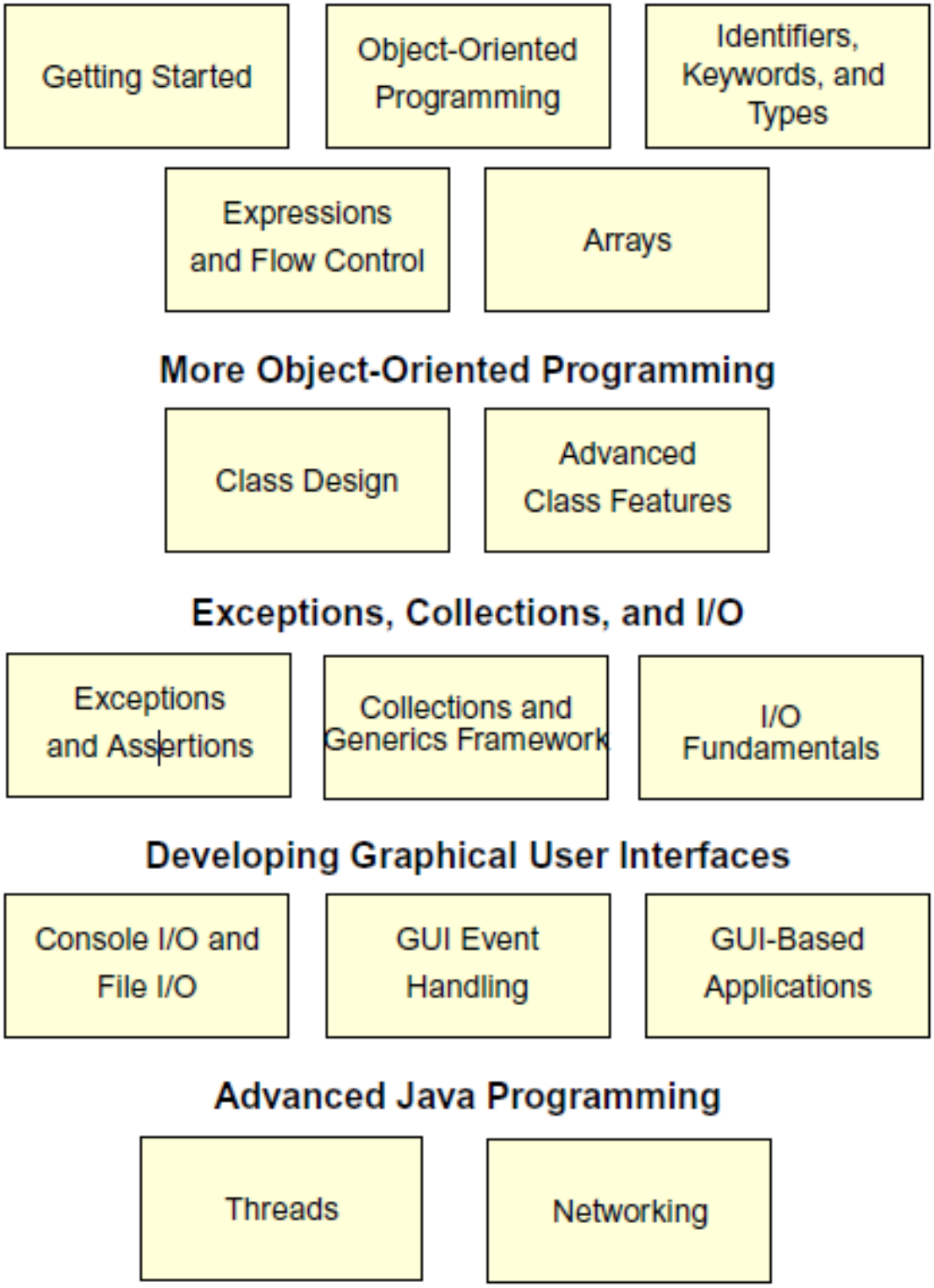


Fig 1: Java programming Basics الشكل 1: أساسيات برمجة جافا

The Java Runtime Environment بيئة جافا وقت التشغيل

The Java application environment performs as follows:

تعمل بيئة تطبيق Java على النحو التالي:

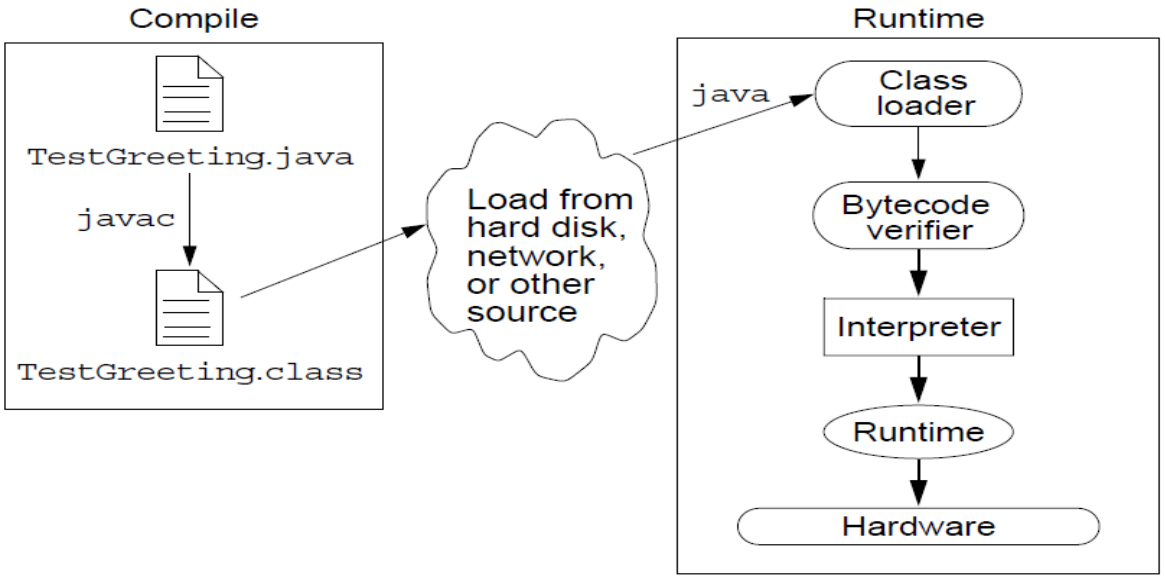


Fig 2: Java Runtime Environment (JRE)

الشكل 2: بيئة تشغيل Java Runtime Environment (JRE)

In some Java technology runtime environments, apportion of the verified bytecode is compiled to native machine code and executed directly on the hardware platform. What is the difference to use that?

في بعض بيئات وقت تشغيل تقنية Java ، يتم تجميع تقسيم الرمز الثانوي الذي تم التحقق منه إلى رمز الجهاز الأصلي ويتم تنفيذه مباشرةً على النظام الأساسي للأجهزة. ما هو الفرق لاستخدام ذلك؟

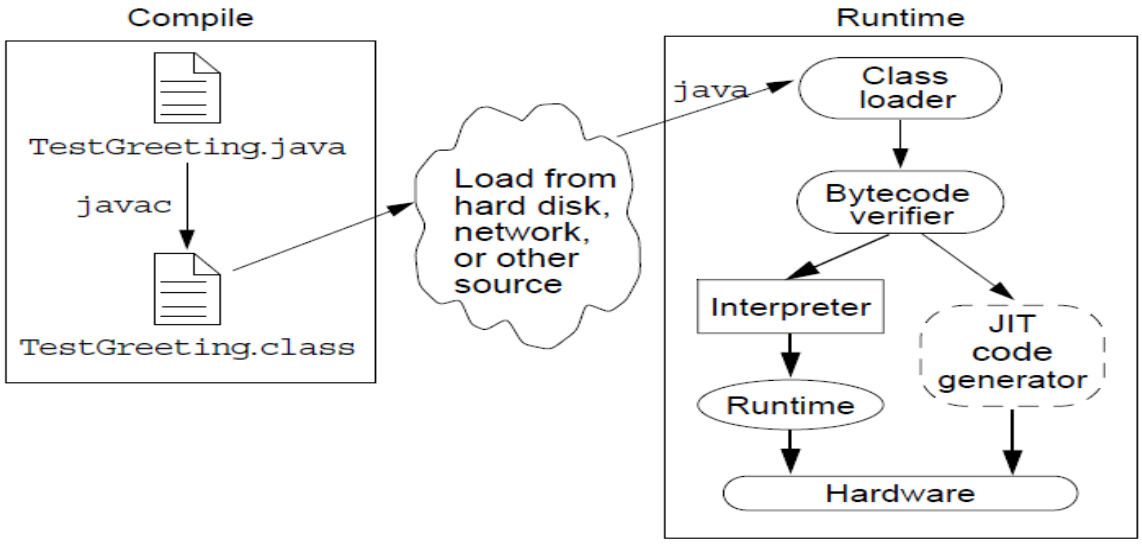


Fig 3: Java Runtime Environment (JRE) with Just in Time (JIT)

Classes and Objects: الفئات والكائنات:

The class is the essence of Java. It is the foundation upon which the entire Java language is built because the class defines the nature of an object. As such, the class forms the basis for object oriented programming in Java. Within a class are defined data and code that acts upon that data. The code is contained in methods.

الفصل هو جوهر جافا. إنه الأساس الذي بنيت عليه لغة Java بأكملها لأن الفئة تحدد طبيعة الكائن. على هذا النحو ، يشكل الفصل الأساس للبرمجة الموجهة للكائنات في Java. داخل الفصل يتم تعريف البيانات والرمز الذي يعمل على تلك البيانات. يتم تضمين الرمز في الأساليب.

Class Fundamentals:أساسيات الفصل :

A class is a template that defines the form of an object. It specifies both the data and the code that will operate on that data. Java uses a class specification to construct *objects*. Objects are *instances* of a class. Thus, a class is essentially a set of plans that specify how to build an object. It is important to be clear on one issue: a class is a logical abstraction. It is not until an object of that class has been created that a physical representation of that class exists in memory.

One other point: recall that the methods and variables that constitute a class are called *members* of the class. The data members are also referred to as *instance variables.*

الفئة هي قالب يحدد شكل الكائن. يحدد كلاً من البيانات والرمز الذي سيعمل على تلك البيانات. تستخدم Java مواصفات الفئة لبناء كائنات. الكائنات هي حالات من فئة. وبالتالي ، فإن الفئة هي في الأساس مجموعة من الخطط التي تحدد كيفية بناء كائن. من المهم أن نكون واضحين بشأن مسألة واحدة: الفصل هو تجريد منطقي. لا يوجد تمثيل مادي لتلك الفئة في الذاكرة حتى يتم إنشاء كائن من تلك الفئة.

نقطة أخرى: تذكر أن الطرق والمتغيرات التي تشكل فئة تسمى أعضاء الفصل. يشار إلى أعضاء البيانات أيضًا باسم متغيرات الحالة.

The General Form of a Class الشكل العام للفصل

When you define a class, you declare its exact form and nature. You do this by specifying the instance variables that it contains and the methods that operate on them. Although very simple classes might contain only methods or only instance variables, most real-world classes contain both.

A class is created by using the keyword **class**. The general form of a **class** definition is shown here:

عندما تحدد فئة ، فإنك تعلن عن شكلها وطبيعتها بالضبط. يمكنك القيام بذلك عن طريق تحديد متغيرات الحالة التي تحتوي عليها والطرق التي تعمل عليها. على الرغم من أن الفئات البسيطة جدًا قد تحتوي فقط على طرق أو متغيرات حالة فقط ، إلا أن معظم فئات العالم الحقيقي تحتوي على كليهما.

يتم إنشاء فئة باستخدام فئة الكلمات الأساسية. الشكل العام لتعريف الطبقة هو

يظهر هنا:

```
class classname {
// declare instance variables إعلان متغيرات الحالة
type var1;
type var2;
// ...
type varN;

// declare methods طرق التصريح
type method1(parameters) {

// body of method
}
type method2(parameters) {
// body of method
}
// ...
type methodN(parameters) {
// body of method
}}
```

Defining a Class تحديد فئة

To illustrate classes we will develop a class that encapsulates information about vehicles, such as cars, vans, and trucks. This class is called **Vehicle**, and it will store three items of information about a vehicle: the number of passengers that it can carry, its fuel capacity, and its average fuel consumption (in miles per gallon).

The first version of **Vehicle** is shown next. It defines three instance variables: **passengers**, **fuelcap**, and **mpg**. Notice that **Vehicle** does not contain any methods. Thus, it is currently a data-only class. (Subsequent sections will add methods to it.)

لتوضيح الفصول الدراسية ، سنقوم بتطوير فصل دراسي يحتوي على معلومات حول المركبات ، مثل السيارات والشاحنات الصغيرة والشاحنات. يُطلق على هذه الفئة اسم "مركبة" ، وستقوم بتخزين ثلاثة عناصر من المعلومات حول السيارة: عدد الركاب الذين يمكنها نقلها ، وسعة الوقود ، ومتوسط استهلاك الوقود (بالأميال لكل جالون).

يتم عرض الإصدار الأول من السيارة بعد ذلك. يحدد ثلاثة متغيرات حالة: الركاب ، والوقود ، و ميلا في الغالون. لاحظ أن السيارة لا تحتوي على أي طرق. وبالتالي ، فهي حاليًا فئة بيانات فقط. (ستضيف الأقسام اللاحقة طرقًا إليها.)

```
class Vehicle {
int passengers; // number of passengers عدد الركاب
int fuelcap; // fuel capacity in gallons سعة الوقود بالغالون
int mpg; // fuel consumption in miles per gallon استهلاك الوقود
بالأميال للغالون الواحد
}
```

A **class** definition creates a new data type. In this case, the new data type is called **Vehicle**. You will use this name to declare objects of type **Vehicle**. Remember that a **class** declaration is only a type description; it does not create an actual object. Thus, the preceding code does not cause any objects of type **Vehicle** to come into existence.

ينشئ تعريف الفئة نوع بيانات جديدًا. في هذه الحالة ، يُطلق على نوع البيانات الجديد اسم مركبة. ستستخدم هذا الاسم للإعلان عن كائنات من نوع مركبة. تذكر أن إعلان الفئة ليس سوى وصف للنوع ؛ لا يُنشئ كائنًا حقيقيًا. وبالتالي ، فإن الكود السابق لا يتسبب في ظهور أي كائنات من نوع مركبة.

To actually create a **Vehicle** object, you will use a statement like the following:

لإنشاء كائن مركبة فعليًا ، ستستخدم عبارة مثل ما يلي:

```
Vehicle minivan = new Vehicle(); // create a Vehicle object called minivan إنشاء كائن مركبة يسمى الميني فان
```

How Objects Are Created كيف يتم إنشاء الكائنات

In the preceding programs, the following line was used to declare an object of type **Vehicle**:

في البرامج السابقة ، تم استخدام السطر التالي للإعلان عن كائن من نوع مركبة:

```
Vehicle minivan = new Vehicle( );
```

This declaration performs two functions. يؤدي هذا الإعلان وظيفتين.

First, it declares a variable called **minivan** of the class type **Vehicle**. This variable does not define an object. Instead, it is simply a variable that can *refer to* an object.

Second, the declaration creates a physical copy of the object and assigns to **minivan** a reference to that object. This is done by using the **new** operator.

The **new** operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it. This reference is, more or less, the address in memory of the object allocated by **new**. This reference is then stored in a variable. Thus, in Java, all class objects must be dynamically allocated.

The two steps combined in the preceding statement can be rewritten like this to show each step individually:

أولاً ، يعلن عن متغير يسمى minivan من نوع الفئة Vehicle. هذا المتغير لا يعرّف الكائن. بدلاً من ذلك ، إنه ببساطة متغير يمكن أن يشير إلى كائن. ثانيًا ، يُنشئ الإعلان نسخة مادية من الكائن ويعين للشاحنة الصغيرة إشارة إلى هذا الكائن. يتم ذلك باستخدام عامل التشغيل الجديد.

يخصص المشغل الجديد بشكل ديناميكي (أي يخصص في وقت التشغيل) ذاكرة لملف الكائن وإرجاع إشارة إليه. هذا المرجع هو ، بشكل أو بآخر ، العنوان الموجود في ذاكرة الكائن المخصص بواسطة new. ثم يتم تخزين هذا المرجع في متغير. وبالتالي ، في Java ، يجب تخصيص جميع كائنات الفئة ديناميكيًا.

يمكن إعادة كتابة الخطوتين معًا في العبارة السابقة بهذا الشكل لإظهار كل خطوة على حدة:

```
Vehicle minivan; // declare reference to object تعلن الإشارة إلى الكائن
minivan = new Vehicle( ); // allocate a Vehicle object تخصيص كائن مركبة
```

The first line declares **minivan** as a reference to an object of type **Vehicle**. Thus, **minivan** is a variable that can refer to an object, but it is not an object, itself. At this point, **minivan** contains the value **null**, which means that it does not refer to an object. The next line creates a new **Vehicle** object and assigns a reference to it to **minivan**. Now, **minivan** is linked with an object.

يعلن السطر الأول الميني فان كمرجع لكائن من نوع مركبة. وبالتالي ، فإن minivan هو متغير يمكن أن يشير إلى كائن ، لكنه ليس كائنًا في حد ذاته. في هذه المرحلة ، تحتوي الميني فان على القيمة فارغة ، مما يعني أنها لا تشير إلى كائن. ينشئ السطر التالي كائنًا جديدًا للمركبة ويعين مرجعًا له إلى الميني فان. الآن ، الميني فان مرتبطة بجسم ما.

After this statement executes, **minivan** will be an instance of **Vehicle**. Each time you create an instance of a class, you are creating an object that contains its own copy of each instance variable defined by the class. Thus, every **Vehicle** object will contain its own copies of the instance variables **passengers**, **fuelcap**, and **mpg**.

بعد تنفيذ هذا البيان ، ستكون الميني فان مثالاً للمركبة. في كل مرة تقوم فيها بإنشاء مثيل لفئة ، فإنك تقوم بإنشاء كائن يحتوي على نسخته الخاصة من كل متغير مثيل محدد بواسطة الفئة. وبالتالي ، سيحتوي كل كائن مركبة على نسخه الخاصة من متغيرات الحالة ، الركاب ، غطاء الوقود ، وميل في الغالون.

To access these variables, you will use the dot (.) operator. The *dot operator* links the name of an object with the name of a member. The general form of the dot operator is shown here:

للوصول إلى هذه المتغيرات ، ستستخدم عامل النقطة (.). يربط عامل النقطة اسم كائن باسم عضو. يظهر الشكل العام لعامل النقطة هنا:

```
object.member
```

Thus, the object is specified on the left, and the member is put on the right. For example, to assign the **fuelcap** variable of **minivan** the value 16, use the following statement:

وهكذا ، يتم تحديد الكائن على اليسار ، والعضو على اليمين. على سبيل المثال ، لتعيين متغير غطاء الوقود لشاحنة صغيرة بقيمة 16 ، استخدم العبارة التالية:

```
minivan.fuelcap = 16;
```

In general, you can use the dot operator to access both instance variables and methods.

بشكل عام ، يمكنك استخدام عامل التشغيل dot للوصول إلى كل من متغيرات الحالة والطرق.

Here is a complete program that uses the **Vehicle** class:

إليك برنامج كامل يستخدم فئة السيارة:

```
/* A program that uses the Vehicle class.برنامج يستخدم فئة السيارة .
Call this file VehicleDemo.javaاستدعاء هذا الملف عرض السيارة. جافا
*/
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
}
// This class declares an object of type Vehicle.
                                    //تعلن هذه الفئة عن كائن من نوع مركبة.
class VehicleDemo {
public static void main(String args[]) {
Vehicle minivan = new Vehicle();
int range;
// assign values to fields in minivan تعيين القيم إلى الحقول في
شاحنة صغيرة
minivan.passengers = 7;
minivan.fuelcap = 16;
minivan.mpg = 21;
// compute the range assuming a full tank of gasاحسب النطاق بافتراض
وجود خزان غاز ممتلئ
range = minivan.fuelcap * minivan.mpg;

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

The following output is displayed::يتم عرض الإخراج التالي

Minivan can carry 7 with a range of 336 يمكن أن تحمل الميني فان 7 بمدى

Before moving on, let's review a fundamental principle: each object has its own copies of the instance variables defined by its class. Thus, the contents of the variables in one object can differ from the contents of the variables in another. There is no connection between the two objects except for the fact that they are both objects of the same type. For example, if you have two **Vehicle** objects, each has its own copy of **passengers**, **fuelcap**, and **mpg**, and the contents of these can differ between the two objects. The following program demonstrates this fact. (Notice that the class with **main( )** is now called **TwoVehicles**.)

قبل المضي قدمًا ، دعنا نراجع مبدأ أساسيًا: كل كائن له نسخه الخاصة من متغيرات الحالة المحددة بواسطة فئته. وبالتالي ، يمكن أن تختلف محتويات المتغيرات في كائن واحد عن محتويات المتغيرات في كائن آخر. لا توجد علاقة بين الكائنين باستثناء حقيقة أنهما كائنان من نفس النوع. على سبيل المثال ، إذا كان لديك جسمان للسيارة ، فلكل منهما نسخته الخاصة من الركاب ، وغطاء الوقود ، و mpg ، ويمكن أن تختلف محتويات هذين العنصرين. البرنامج التالي يوضح هذه الحقيقة. (لاحظ أن الفصل الذي يحتوي على main () يسمى الآن TwoVehicles.)

```
// This program creates two Vehicle objects.
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 //
}
// This class declares an object of type Vehicle.
                                تعلن هذه الفئة عن كائن من نوع مركبة. //
class TwoVehicles {
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;
// compute the ranges assuming a full tank of gas
                                حساب النطاقات بافتراض وجود خزان غاز ممتلئ //
range1 = minivan.fuelcap * minivan.mpg;
range2 = sportscar.fuelcap * sportscar.mpg;
System.out.println("Minivan can carry " + minivan.passengers +
" with a range of " + range1);
System.out.println("Sportscar can carry " + sportscar.passengers +
```

```
" with a range of " + range2);
}
}
```

The output produced by this program is shown here:
Minivan can carry 7 with a range of 336
Sportscar can carry 2 with a range of 168
As you can see, **minivan**'s data is completely separate from the data contained in **sportscar**.
The following illustration depicts this situation.

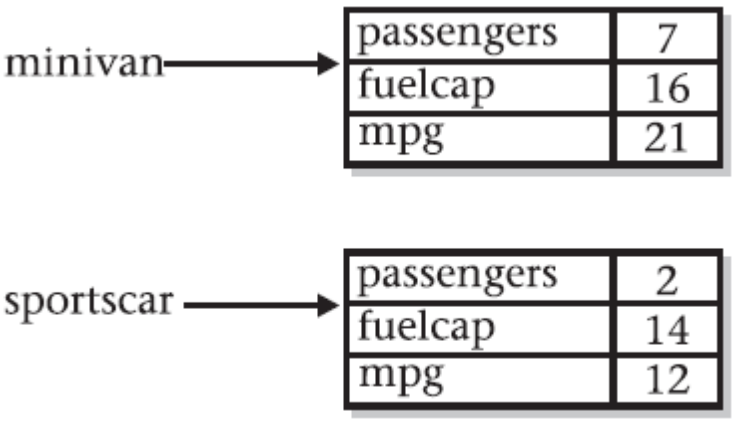
يتم عرض الإخراج الناتج عن هذا البرنامج هنا:

يمكن أن تحمل الميني فان 7 بمدى 336

يمكن أن تحمل Sportscar 2 مع مدى من 168

كما ترى ، فإن بيانات الميني فان منفصلة تمامًا عن البيانات الموجودة في السيارة الرياضية.

يوضح الرسم التوضيحي التالي هذا الموقف.



Reference Variables and Assignment المتغيرات المرجعية والتخصيص

In an assignment operation, object reference variables act differently than do variables of a primitive type, such as **int**. When you assign one primitive-type variable to another, the situation is straightforward. The variable on the left receives a *copy* of the *value* of the variable on the right. When you assign an object reference variable to another, the situation is a bit more complicated because you are changing the object that the reference variable refers to.

The effect of this difference can cause some counterintuitive results. For example, consider

في عملية الإسناد ، تعمل متغيرات مرجع الكائن بشكل مختلف عن متغيرات النوع البدائي ، مثل int. عندما تقوم بتعيين متغير من النوع الأولي إلى متغير آخر ، يكون الموقف مباشرًا. المتغير الموجود على اليسار يتلقى نسخة من قيمة المتغير على اليمين. عندما تقوم بتعيين متغير مرجع كائن لآخر ، يكون الموقف أكثر تعقيدًا بعض الشيء لأنك تقوم بتغيير الكائن الذي يشير إليه المتغير المرجعي.

يمكن أن يؤدي تأثير هذا الاختلاف إلى بعض النتائج غير البديهية. على سبيل المثال ، ضع في اعتبارك

the following fragment:الجزء التالي :

```
Vehicle car1 = new Vehicle();
Vehicle car2 = car1;
```

At first glance, it is easy to think that **car1** and **car2** refer to different objects, but this is not the case. Instead, **car1** and **car2** will both refer to the *same* object. The assignment of **car1** to **car2** simply makes **car2** refer to the same object as does **car1**. Thus, the object can be acted

upon by either **car1** or **car2**. For example, after the assignment

للوهلة الأولى ، من السهل التفكير في أن car1 و car2 يشيران إلى كائنات مختلفة ، لكن هذا ليس هو الحال. بدلاً من ذلك ، سيشير كل من car1 و car2 إلى نفس الكائن. يؤدي تخصيص car1 إلى car2 ببساطة إلى جعل car2 تشير إلى نفس الكائن مثل car1. وبالتالي ، يمكن أن يتصرف الكائن

عليها إما السيارة 1 أو السيارة 2. على سبيل المثال ، بعد المهمة

```
car1.mpg = 26;
```

executes, both of these **println( )** statements

```
System.out.println(car1.mpg);
System.out.println(car2.mpg);
```

display the same value: 26.

Although **car1** and **car2** both refer to the same object, they are not linked in any other way. For example, a subsequent assignment to **car2** simply changes the object to which **car2** refers. For example:

عرض نفس القيمة: 26.

على الرغم من أن كلاً من car1 و car2 يشيران إلى نفس الكائن ، إلا أنهما غير مرتبطين بأي طريقة أخرى. على سبيل المثال ، يؤدي تعيين لاحق لـ car2 ببساطة إلى تغيير الكائن الذي تشير إليه car2. على سبيل المثال:

```
Vehicle car1 = new Vehicle();
Vehicle car2 = car1;
Vehicle car3 = new Vehicle();
car2 = car3; // now car2 and car3 refer to the same object.
                                // الآن تشير car2 و car3 إلى نفس الكائن.
```

After this sequence executes, **car2** refers to the same object as **car3**. The object referred to by **car1** is unchanged.

بعد تنفيذ هذا التسلسل ، تشير car2 إلى نفس الكائن مثل car3. الكائن المشار إليه بواسطة car1 لم يتغير.

Translated by :- M.J