Constructors:المنشئون :

A *constructor* initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type. Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other startup procedures required to create a fully formed object.

يقوم المُنشئ بتهيئة كائن عند إنشائه. له نفس اسم فئته ويشبه طريقة ما من الناحية التركيبية. ومع ذلك ، ليس للمُنشئين نوع إرجاع صريح. عادةً ، ستستخدم مُنشئًا لإعطاء قيم أولية لمتغيرات المثيل المحددة بواسطة الفئة ، أو لتنفيذ أي إجراءات بدء تشغيل أخرى مطلوبة لإنشاء كائن مكون بالكامل.

All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

تحتوي جميع الفئات على مُنشئين ، سواء قمت بتعريف واحد أم لا ، لأن Java توفر تلقائيًا مُنشئًا افتراضيًا يقوم بتهيئة جميع متغيرات الأعضاء إلى الصفر. ومع ذلك ، بمجرد تحديد المُنشئ الخاص بك ، لن يتم استخدام المُنشئ الافتراضي.

Here is a simple example that uses a constructor:فيما يلي مثال بسيط يستخدم المُنشئ

```
//Asimpleconstructor.منشئ
بسيط



class MyClass {
 int x;
  MyClass() {
   x = 10;
  }
}
class ConsDemo {
public static void main(String args[]) {
MyClass t1 = new MyClass();
MyClass t2 = new MyClass();
System.out.println(t1.x + " " + t2.x);
}
}

In this example, the constructor for MyClass is
MyClass() {
  x = 10;
}
```

This constructor assigns the instance variable **x** of **MyClass** the value 10. This constructor is called by **new** when an object is created. For example, in the line

يقوم هذا المُنشئ بتعيين متغير المثيل x الخاص بـ MyClass بالقيمة 10. يتم استدعاء هذا المُنشئ بواسطة new عند إنشاء كائن. على سبيل المثال ، في السطر

```
MyClass t1 = new MyClass();
```

the constructor **MyClass( )** is called on the **t1** object, giving **t1.x** the value 10. The same is true for **t2**. After construction, **t2.x** has the value 10. Thus, the output from the program is

يتم استدعاء المُنشئ MyClass () على الكائن t1 ، مع إعطاء t1.x القيمة 10. وينطبق الشيء نفسه على t2. بعد الإنشاء ، يكون لـ t2.x القيمة 10. وبالتالي ، يكون ناتج البرنامج

```
10 10
```

Parameterized Constructors المنشئات ذات المعاملات

In the preceding example, a parameter-less constructor was used. Although this is fine for some situations, most often you will need a constructor that accepts one or more parameters.

في المثال السابق ، تم استخدام مُنشئ بدون معلمات. على الرغم من أن هذا أمر جيد في بعض المواقف ، إلا أنك ستحتاج في أغلب الأحيان إلى مُنشئ يقبل معلمة واحدة أو أكثر.

Parameters are added to a constructor in the same way that they are added to a method: just declare them inside the parentheses after the constructor's name. For example, here, **MyClass** is given a parameterized constructor:

تتم إضافة المعلمات إلى المُنشئ بنفس طريقة إضافتها إلى الطريقة: فقط قم بتعريفها داخل الأقواس بعد اسم المنشئ. على سبيل المثال ، هنا ، يتم إعطاء MyClass مُنشئ معلمات:

```
// A parameterized
constructor منشئ ذو معلمات .
```

```
class MyClass {
int x;
MyClass(int i) {
x = i;
}
}
class ParmConsDemo {
public static void main(String args[]) {
MyClass t1 = new MyClass(10);
MyClass t2 = new MyClass(88);
System.out.println(t1.x + " " + t2.x);
}
}
```

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

```
10 88
```

In this version of the program, the **MyClass( )** constructor defines one parameter called **i**, which is used to initialize the instance variable, **x**. Thus, when the line

في هذا الإصدار من البرنامج ، يحدد مُنشئ MyClass () معلمة واحدة تسمى i ، والتي تُستخدم لتهيئة متغير المثيل ، x. وهكذا ، عندما يكون الخط

```
MyClass t1 = new MyClass(10);
```

executes, the value 10 is passed to **i**, which is then assigned to **x**. This constructor has a parameter.

في حالة التنفيذ ، يتم تمرير القيمة 10 إلى i ، والتي يتم تعيينها بعد ذلك إلى x. هذا المنشئ له معلمة

Adding a Constructor to the Vehicle Class إضافة منشئ لفئة المركبة

We can improve the **Vehicle** class by adding a constructor that automatically initializes the **passengers**, **fuelcap**, and **mpg** fields when an object is constructed. Pay special attention to how **Vehicle** objects are created.

يمكننا تحسين فئة السيارة عن طريق إضافة مُنشئ يقوم تلقائيًا بتهيئة حقول الركاب ووقود الوقود و mpg عند إنشاء كائن. انتبه بشكل خاص لكيفية تكوين أجسام المركبة.

```
// Add a constructor.
```

// أضافة المؤنشئ //

```
class Vehicle {
int passengers; // number of passengers
int fuelcap; // fuel capacity in gallons
int mpg; // fuel consumption in miles per gallon

// This is a constructor for Vehicle
```

// هذا مُنشئ للمركبة . . //

```
Vehicle(int p, int f, int m) {
passengers = p;
fuelcap = f;
mpg = m;
}

// Return the range.
```

```
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 VehConsDemo {
public static void main(String args[]) {
// construct complete vehicles
```

// بناء مركبات كاملة

```
Vehicle minivan = new Vehicle(7, 16, 21);
Vehicle sportscar = new Vehicle(2, 14, 12);
double gallons;
int dist = 252;
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.");
}
}
```

Both **minivan** and **sportscar** are initialized by the **Vehicle( )** constructor when they are created. Each object is initialized as specified in the parameters to its constructor. For example, in the following line,

تتم تهيئة كل من الشاحنة الصغيرة والسيارة الرياضية بواسطة مُصمم السيارة () عند إنشائها. تتم تهيئة كل كائن كما هو محدد في معلمات منشئه. على سبيل المثال ، في السطر التالي ،

```
Vehicle minivan = new Vehicle(7, 16, 21);
```

The values 7, 16, and 21 are passed to the **Vehicle( )** constructor when **new** creates the object. Thus, **minivan**'s copy of **passengers**, **fuelcap**, and **mpg** will contain the values 7, 16, and 21, respectively. The output from this program is the same as the previous version.

يتم تمرير القيم 7 و 16 و 21 إلى مُنشئ السيارة () عندما ينشئ الكائن الجديد. وبالتالي ، ستحتوي نسخة الميني فان الخاصة بالركاب ووقود الوقود وميل لكل جالون على القيم 7 و 16 و 21 على التوالي. الإخراج من هذا البرنامج هو نفس الإصدار السابق.

Controlling Access to Class Members:التحكم في الوصول إلى أعضاء الفصل الدراسي :

In essence, there are two basic types of class members: public and private. A public member can be freely accessed by code defined outside of its class. This is the type of class member that we have been using up to this point.

في الجوهر ، هناك نوعان أساسيان من أعضاء الفصل: العام والخاص. يمكن الوصول إلى عضو عام بحرية من خلال رمز محدد خارج فئته. هذا هو نوع أعضاء الفصل الذي كنا نستخدمه حتى هذه النقطة.

A private member can be accessed only by other methods defined by its class. It is through the use of private members that access is controlled.

يمكن الوصول إلى عضو خاص فقط من خلال طرق أخرى محددة بواسطة فئته. من خلال استخدام الأعضاء الخاصين يتم التحكم في الوصول.

Restricting access to a class's members is a fundamental part of object-oriented programming because it helps prevent the misuse of an object. By allowing access to private data only through a well-defined set of methods, you can prevent improper values from being assigned to that data—by performing a range check, for example. It is not possible for code outside the class to set the value of a private member

يعد تقييد الوصول إلى أعضاء الفصل جزءًا أساسيًا من البرمجة الموجهة للكائنات لأنه يساعد في منع إساءة استخدام الكائن. بالسماح بالوصول إلى البيانات الخاصة فقط من خلال مجموعة طرق محددة جيدًا ، يمكنك منع تعيين القيم غير الصحيحة لتلك البيانات — عن طريق إجراء فحص النطاق ، على سبيل المثال. لا يمكن للكود خارج الفصل تعيين قيمة عضو خاص

directly. You can also control precisely how and when the data within an object is used. Thus, when correctly implemented, a class creates a “black box” that can be used, but the inner workings of which are not open to tampering.

Up to this point, you haven’t had to worry about access control because Java provides a default access setting in which the members of a class are freely available to the other code in your program. (Thus, the default access setting is essentially public.) .

مباشرة. يمكنك أيضًا التحكم بدقة في كيفية ووقت استخدام البيانات الموجودة داخل الكائن. وهكذا ، عندما يتم تطبيق الفصل بشكل صحيح ، فإنه يخلق "صندوقًا أسود" يمكن استخدامه ، ولكن الأعمال الداخلية منه ليست مفتوحة للعبث.

حتى هذه اللحظة ، لم يكن لديك ما يدعو للقلق بشأن التحكم في الوصول لأن Java توفر إعداد وصول افتراضيًا يكون فيه أعضاء الفصل متاحين مجانًا للرمز الآخر في برنامجك. (وبالتالي ، فإن إعداد الوصول الافتراضي هو في الأساس عام.).

Java’s Access Specifiers: محددات الوصول إلى Java:

Member access control is achieved through the use of three access specifiers: **public**, **private**, and **protected**. As explained, if no access specifier is used, the default access setting is assumed.

يتحقق التحكم في وصول الأعضاء من خلال استخدام ثلاثة محددات وصول: عام ، خاص ، ومحمي. كما هو موضح ، إذا لم يتم استخدام محدد وصول ، فسيتم افتراض إعداد الوصول الافتراضي.

When a member of a class is modified by the public specifier, that member can be accessed by any other code in your program. This includes methods defined inside other classes.

عندما يتم تعديل عضو في فئة بواسطة المحدد العام ، يمكن الوصول إلى هذا العضو بواسطة أي رمز آخر في برنامجك. يتضمن ذلك الطرق المحددة داخل الفئات الأخرى.

When a member of a class is specified as **private**, that member can be accessed only by other members of its class. Thus, methods in other classes cannot access a private member of another class.

عندما يتم تحديد عضو من فئة على أنه خاص ، لا يمكن الوصول إلى هذا العضو إلا من قبل أعضاء آخرين في فئته. وبالتالي ، لا يمكن للطرق في الفئات الأخرى الوصول إلى عضو خاص من فئة أخرى.

The default access setting (in which no access specifier is used) is the same as public unless your program is broken down into packages. A package is, essentially, a grouping of classes.

يعد إعداد الوصول الافتراضي (الذي لا يتم فيه استخدام محدد وصول) هو نفسه الإعداد العام ما لم يتم تقسيم البرنامج إلى حزم. الحزمة ، في الأساس ، هي مجموعة من الطبقات.

An access specifier precedes the rest of a member’s type specification. That is, it must begin a member’s declaration statement.

يسبق محدد الوصول باقي مواصفات نوع العضو. أي أنه يجب أن يبدأ بيان إعلان العضو.

Constructor and Set and Get Methods:منشئ وتعيين الطرق والحصول عليها :

Constructor Used to initialize an object of the class and any members. Called only once for the lifetime of the object being initialized:

المُنشئ يُستخدم لتهيئة كائن للفئة وأي أعضاء. تم استدعاؤه مرة واحدة فقط طوال مدة تهيئة الكائن:

```
public class Person
{
   String person_name;

      public Person(String name)
   {
      person _name = name;
   }
}
```

Example usage:

```
// Initialize object of Person class by calling constructor.
Person p = new Person("John Smith");
```

The Get and Set Methods :طرق الحصول على وتعيين:

Used to get or set values of object members respectively. Unlike constructors, set methods can be used to initialize member values more than once:

تستخدم للحصول على أو تعيين قيم أعضاء الكائن على التوالي. على عكس المُنشئ ، يمكن استخدام طرق المجموعة لتهيئة قيم الأعضاء أكثر من مرة:

```
public class Person
{
    // Member.
   Private String person_name;
    // Constructor.
   public Person(String name)
   {
      person_name = name;
   }

    // Get member value.
    public String getName()
    {
       return person_name;
    }

    // Set member value to given value.
    public void setName(String name)
    {
       person_name = name;
    }
}
```

Example usage:

```
// Initialize object of Person class.
Person p = new Person("John Smith");
String name;
// Get name of this person.
name = p.getName();
System.out.println(name);

// Set new name for this person.
p.setName("John Doe");
name = p.getName();
System.out.println(name);
```

Output:

```
John Smith
John Doe
```

The toString method :طريقة_ toString:

- Java provides an inbuilt method called toString()
- This is called whenever an object is processed as a string. For example, being printed out:

• توفر Java طريقة مضمنة تسمى toString ()
• يسمى هذا عندما تتم معالجة كائن كسلسلة. على سبيل المثال ، يتم طباعتها:

- If you provide a toString() method in your class, this will be automatically called when the object is converted to a string.
- The toString() method must return a string and take no arguments.

• إذا قمت بتوفير طريقة toString () في الفصل الدراسي الخاص بك ، فسيتم استدعاء هذا تلقائيًا عند تحويل الكائن إلى سلسلة.
• يجب أن تُرجع طريقة toString () سلسلة نصية ولا تأخذ أي وسيطات.

- **public String toString() { }**

```
public class Person {
    private  String firstname;
    private  String surname;
    private int age;

    public Person(String firstname, String surname, int age) {
        this.firstname = firstname;
        this.surname = surname;
        this.age = age;
    }

    public String toString() {
        return this.age + " year old called " +
            this.firstname + " " + this.surname;

    }
}
```

```
public static void main(String[] args) {
    Person person1 = new Person("John",
                        "Smith",25);

    System.out.println(person1);
}
```

```
Output:
25 year old called John Smith
```

To understand the effects of **public** and **private**, consider the following program:

لفهم تأثيرات القطاعين العام والخاص ، ضع في اعتبارك البرنامج التالي:

```
// Public vs private access
```

الوصول العام مقابل الوصول الخاص .

```
.class MyClass {
private int alpha; // private access
public int beta; // public access
int gamma; // default access (essentially public)
/* Methods to access alpha. It is OK for a
member of a class to access a private member of the same class.
*/
```

/ * طرق الوصول إلى alpha. لا بأس بالنسبة لـ
عضو في فئة للوصول إلى عضو خاص من نفس الفئة.
* /

```
void setAlpha(int a) {
alpha = a;
}
int getAlpha() {
return alpha;
}
}

class AccessDemo {
public static void main(String args[]) {
MyClass ob = new MyClass();

/* Access to alpha is allowed only through
its accessor methods. */
ob.setAlpha(-99);
System.out.println("ob.alpha is " + ob.getAlpha());
// You cannot access alpha like this:
// ob.alpha = 10; // Wrong! alpha is private!
// These are OK because beta and gamma are public.
ob.beta = 88;
ob.gamma = 99;
}
}
```

As you can see, inside the **MyClass** class, **alpha** is specified as **private**, **beta** is explicitly specified as **public**, and **gamma** uses the default access, which for this example is the same as specifying **public**. Because **alpha** is private, it cannot be accessed by code outside of its class.

كما ترى ، داخل فئة MyClass ، يتم تحديد alpha على أنه خاص ، ويتم تحديد beta بشكل صريح على أنه عام ، ويستخدم gamma الوصول الافتراضي ، والذي يكون في هذا المثال هو نفسه تحديد public. نظرًا لأن alpha خاص ، فلا يمكن الوصول إليه برمز خارج فئته.

Therefore, inside the **AccessDemo** class, **alpha** cannot be used directly. It must be accessed through its public accessor methods: **setAlpha( )** and **getAlpha( )**. If you were to remove the comment symbol from the beginning of the following line,

لذلك ، داخل فئة AccessDemo ، لا يمكن استخدام alpha مباشرة. يجب الوصول إليه من خلال طرق الوصول العامة الخاصة به: setAlpha () و getAlpha (). إذا كنت تريد إزالة رمز التعليق من بداية السطر التالي ،

```
// ob.alpha = 10; // Wrong! alpha is private!
```

You would not be able to compile this program because of the access violation. Although access to **alpha** by code outside of **MyClass** is not allowed, methods defined within **MyClass** can freely access it, as the **setAlpha( )** and **getAlpha( )** methods show. The key point is this: a private member can be used freely by other members of its class, but it cannot be accessed by code outside its class.

لن تتمكن من تجميع هذا البرنامج بسبب انتهاك الوصول. على الرغم من أن الوصول إلى alpha by code خارج MyClass غير مسموح به ، إلا أن الطرق المحددة داخل MyClass يمكنها الوصول إليها بحرية ، كما تظهر الطرق setAlpha () و getAlpha (). النقطة الأساسية هي: يمكن استخدام العضو الخاص بحرية من قبل أعضاء فئته الآخرين ، لكن لا يمكن الوصول إليه عن طريق رمز خارج فئته.