الفصل التاسع عشر

تنفيذ أكثر من عملية فى وقت واحد
Threading

Programming Concepts 19

فى هذا الفصل

فى هذا الفصل نتناول موضوع اعداد برنامج ينفذ أكثر من عملية فى نفس الوقت وهو ما يسمى Threading وذلك من خلال النقاط التالية :-

- ما هو الـ Thread
- كيفية إنشاء Thread
- ما هى الفصيلة Thread
- أعضاء الفصيلة Thread
- تغيير درجة أولوية التنفيذ Priority
- التحكم فى تنفيذ الـ Thread

تخيل أنك كنت تجلس على مكتبك تقرأ كتاباً. بإمكانك شرب كوب من الشاى وأنت تقرأ كما يمكنك أيضاً الاستماع إلى موسيقى.

- إذن فى نفس الوقت أمكنك القيام بعدة أعمال وإلا كان لابد أن تكمل قراءة الكتاب ثم تشرب كوب الشاى ثم تستمتع إلى الموسيقى.
- طبعاً كونك أديت جميع هذه الأعمال فى نفس الوقت وفرت عليك وقتاً لتنفيذ جميع هذه العمليات.
- من هنا نستطيع أن نعرف اكثر من مصطلح.

العمليات المتعددة Multiprocessing

- هو القدرة على القيام بأكثر من عملية process فى نفس الوقت.
- فمثلاً يمكن فتح برنامج الكتابة Word وبرنامج الجداول Excel وبرنامج العروض PowerPoint فى نفس الوقت ولكن لاحظ أنه لا توجد علاقة بين برنامج الكتابة Word وبرنامج الجداول Excel وبرنامج العروض PowerPoint بمعنى أن اى برنامج منهم لا يعتمد على الآخر أو ينتظر إتمام عملية من الآخر.

المهام المتعددة Multithreading

- هو القدرة على أداء اكثر من وظيفة داخل نفس العملية process.
- فمثلاً إذا كنت تلعب لعبة تتعلق بالفضاء مثلاً فإنك تحرك طائرتك وتظهر لك أهدافاً لتضربها كما يوجد أيضاً صوت فى اللعبة.
- إذن داخل العملية الواحدة (والتى هي اللعبة) فيمكنك القيام بأكثر من وظيفة فى نفس الوقت وإلا تخيل أنك تضرب ثم تظهر الأهداف ثم تتحرك ثم يظهر الصوت وهي طبعاً عملية غير منطقية لأنك تريد أن تنفذ جميع هذه العمليات فى نفس الوقت.

إذن نستطيع القول أنه عندنا نوعان من التطبيقات:

1. التطبيقات ذات المهمة الواحدة Single-Threaded Applications
2. التطبيقات ذات المهام المتعددة Multi-Threaded Applications

التطبيقات ذات المهمة الواحدة Single-Threaded Applications

هو التطبيق الذى يستطيع أداء وظيفة واحدة فى وقت ما. فإذا كان عندك اكثر من وظيفة تريد تنفيذها فلابد أن ينتهى تنفيذ الوظيفة الأولى حتى يمكنك تنفيذ الثانية ولابد أن ينتهى تنفيذ الوظيفة الثانية حتى تبدأ الثالثة وهكذا حتى ينتهى تنفيذ جميع الوظائف الموجودة بالبرنامج.

التطبيقات ذات المهام المتعددة Multi-Threaded Applications

- هو التطبيق الذى يستطيع أداء أكثر من وظيفة فى نفس الوقت. فإذا كان عندك أكثر من وظيفة فيمكنك تنفيذ هذه الوظيفة مع وظيفة أخرى فى نفس الوقت بدون أن تنتظر انتهاء الوظيفة الأولى.
- فمثلاً يمكنك باستخدام برنامج الكتابة Word كتابة التقارير وطباعتها.
- أيضاً يمكنك باستخدام برنامج مستكشف الإنترنت Internet Explorer أن تنزل برنامج (Download) وأن تستمع إلى موسيقى وأن تتصفح صفحات الوب.
- إمكانية أداء أكثر من وظيفة فى نفس الوقت عملية هامة جداً لأنها أقرب إلى الحياة الواقعية.
- فى هذا الفصل سوف نتحدث عن كيفية أداء اكثر من وظيفة فى نفس الوقت وكيفية الاستفادة من ذلك.

فى لغة VB.NET

ما هو Threading؟

هى طريقة تسمح بتنفيذ اكثر من عملية Task فى وقت واحد كل عملية تنفذ فى Thread مع امكانية التحكم فى هذه العمليات مثل ايقاف الـ Thread مؤقتا أو نهائيا.

ومثال لذلك تطبيق الانترنت الذى يأتيه أكثر من طلب إذا أمكن التعامل مع أكثر من طلب فى نفس الوقت زادت كفاءة التطبيق ويتم ذلك بإنشاء أكثر من Thread باستعمال الفصيلة thread. ولتوضيح كيفية استعمال فكرة إنشاء Thread وتنفيذ عملية داخله تابع المثال التالى

مثال

1- قم باعداد تطبيق جديد من النوع Console Application

2- اكتب به السطور التالية.

```
Imports System
Imports System.Threading
Module Module1
    Sub Main()
        Dim t1 As System.Threading.Thread
        t1 = New System.Threading.Thread(AddressOf ThreadEntryMethod)
        t1.Name = "Hello"
        t1.Start()

        Dim t2 As System.Threading.Thread
        t2 = New System.Threading.Thread(AddressOf ThreadEntryMethod)
        t2.Name = "Hello"
        t2.Start()
    End Sub

    Public Sub ThreadEntryMethod()
        Dim r As Random = New Random(Thread.CurrentThread.GetHashCode())
        Do While (True)

            Console.WriteLine(Thread.CurrentThread.Name)
            Dim rount As Integer = r.Next(1000)
            Thread.Sleep(rount)
        Loop
    End Sub
End Module
```

فى C#

```
class MyThreadClass
{
    //ThreadEntryMethod method
    public void ThreadEntryMethod()
    {
        Random r = new Random(
            Thread.CurrentThread.GetHashCode());
        while (true)
        {
            Console.WriteLine(Thread.CurrentThread.Name);
            int rount = r.Next(1000);
            Thread.Sleep(rount);
        }
    }
}
class CreateThread
{
    public static void Main()
    {
        MyThreadClass mtc = new MyThreadClass();

        //establish thread entry method
        ThreadStart ts = new ThreadStart(mtc.ThreadEntryMethod);

        //create first thread, set name to "Hello", start
        Thread t1 = new Thread(ts);
        t1.Name = "First";
        t1.Start();

        //create second thread, set name to "Hello", start
        Thread t2 = new Thread(ts);
        t2.Name = "Second";
        t2.Start();
    }
}
```

فى هذه السطور

- فى السطر رقم 2 يتم عمل Imports للمكتبة (NameSpace) الموجودة بالاسم System.Threading وذلك لتعريف الفصيلة Thread بها.
- فى السطر رقم 5 تم الاعلان المعرفة فى Threading المعرفة فى System.
- فى السطر رقم 6 تم إنشاء المتغير (الهدف) Object المسمى t1 باستخدام الكلمة New مع ارسال معامل الدالة البناء هو عنوان الدالة ThreadEntryMethod وهى الدالة التى تنفذ فى هذا الـ Thread.
- فى السطر رقم 7 تم تسجيل اسم الـ Thread عن طريق تسجيل قيمة الخاصية Name للهدف t1.
- فى السطر رقم 8 تم استدعاء الدالة ()Start العضوة فى الفصيلة Thread والتى تؤدى إلى بدء عمل الـ Thead وبالتالى تنفيذ الدالة المعرفة فيه ولمشار اليها عند إنشاء هدف من الـ Thead.
- والدالة المعرفة مع الهدف t1 هى الدالة ()ThreadEntryMethod و بالتالى يبدأ تنفيذ هذه الدالة وهذة الدالة معرف فى السطور من 16 إلى 24.
- فى السطر رقم 10 تم إنشاء المتغير (الهدف) Object المسمى t2 باستخدام الكلمة New مع ارسال معامل الدالة البناء هو عنوان الدالة ThreadEntryMethod وهى الدالة التى تنفذ فى هذا الـ Thread.
- فى السطر رقم 11 تم تسجيل اسم الـ Thread عن طريق تسجيل قيمة الخاصية Name للهدف t1.
- فى السطر رقم 12 تم استدعاء الدالة ()Start العضوة فى الفصيلة Thread والتى تؤدى إلى بدء عمل الـ Thead وبالتالى تنفيذ الدالة المعرفة فيه ولمشار إليها عند إنشاء هدف من الـ Thread.
- فى السطور من 16 إلى 24 تم تعريف الدالة ()ThreadEntryMethod و التى تنفذ مع كلا من t1,t2 المعرفين من الفصيلة Thread.

- فى السطر رقم 17 يتم الاعلان عن متغير r من النوع Random وباستعمال الدالة Random لتوليد أرقام عشوائية وكل مرة يسجل رقم فى المتغير r.
- فى السطر رقم 18 يبدأ التكرار Do While بالشرط True الذى يؤدى إلى استمرار التكرار.
- فى السطر رقم 20 يتم استعمال ()Writeline مع الفصيلة Console لطباعة اسم الـ Thread.
- فى السطر رقم 21 يتم أخذ رقم تالى من الأرقام العشوائية ثم ارسال هذا الرقم للدالة ()Sleep فيؤدى إلى توقف الـ Thread لمدة المتغير Count الذى يأخذ قيمته من المتغير r.

نفذ البرنامج F5 تحصل على نتيجة التنفيذ كما فى الشكل (1-19).

الشكل (1-19)

فى هذا الشكل نلاحظ طباعة الكلمة Heads الخاص بالـ Thread الأول t1 ثم الكلمة Tails الخاص بالـ Thread الثانى t2 بشكل غير منتظم ويرجع ذلك إلى قيام الكمبيوتر بتنفيذ ها بشكل غير منظم حسب تفرغ المشغل processor مما أدى إلى ظهور العبارات بشكل غير منظم.

ما هى الفصيلة Thread؟

هى الفصيلة المسؤلة عن تعريف Thread جديد مع توفير الأعضاء (الدوال Methods والخصائص Properties) التى تتحكم فى جميع عمليات الأت Thread مثل بدء تشغيل الـ Thread (الدالة Start()) وايقاف التشغيل Stop() والكثير من العمليات.

تغيير درجة أولوية التنفيذ Priority

عند تشغيل أكثر من Thread يقوم الكمبيوتر بتشغيلها حسب الوقت المالح لمشغل Processor ولكن قد تحتاج لتشغيل Thread قبل أخر وذلك لاهميته أو قد تحتاج لإعادة ترتيب الأولويات وهذا ما يسمى Priority وهى عبارة عن خاصية يتم اعطائها قيمة حسب الأولوية التى تريدها ، وتأخذ هذه الخاصية أحد مجموعة قيم.

ولتوضيح ذلك تابع المثال التالى:

1- قم بإعداد تطبيق جديد من النوع Console Application

2- اكتب به السطور التالية.

```
Imports System
Imports System.Threading
Module Module1
   Sub Main()
      Dim t1 As System.Threading.Thread
      Dim t2 As System.Threading.Thread
      Dim t3 As System.Threading.Thread
      Dim t4 As System.Threading.Thread

t1 = New System.Threading.Thread(AddressOf threadEntryMethod)
t2 = New System.Threading.Thread(AddressOf ThreadEntryMethod)
t3 = New System.Threading.Thread(AddressOf ThreadEntryMethod)
t4 = New System.Threading.Thread(AddressOf ThreadEntryMethod)

1.  t1.Priority = ThreadPriority.Normal
2.  t2.Priority = ThreadPriority.Lowest
3.  t3.Priority = ThreadPriority.Highest
4.  t4.Priority = ThreadPriority.AboveNormal
      t1.Name = "Normal"
      t2.Name = "Lowest"
      t3.Name = "Highest"
      t4.Name = "AboveNormal"
```

```
        t1.Start()
        t2.Start()
        t3.Start()
        t4.Start()
    End Sub

    Public Sub ThreadEntryMethod()
        Dim r As Random = New
Random(Thread.CurrentThread.GetHashCode())
        Do While (True)

            Console.WriteLine(Thread.CurrentThread.Name)
            Dim count As Integer = r.Next(1000)
            Thread.Sleep(count)
        Loop
    End Sub
End Module
```

فى لغة C#

```
using System;
using System.Threading;

class ThreadPriority
{
public static void Main()
{
MyThreadClass mtc = new MyThreadClass();
ThreadStart ts = new ThreadStart(mtc.ThreadStart);

//create the threads and set the priorities
Thread threadLowest = new Thread(ts);
threadLowest.Priority =
System.Threading.ThreadPriority.Lowest;
Thread threadBelowNormal = new Thread(ts);
threadBelowNormal.Priority =
System.Threading.ThreadPriority.BelowNormal;
Thread threadNormal = new Thread(ts);
threadNormal.Priority =
System.Threading.ThreadPriority.Normal;
Thread threadAboveNormal = new Thread(ts);
threadAboveNormal.Priority =
System.Threading.ThreadPriority.AboveNormal;
Thread threadHighest = new Thread(ts);
threadHighest.Priority =
System.Threading.ThreadPriority.Highest;
```

```
//start thread race
threadLowest.Start();
threadBelowNormal.Start();
threadNormal.Start();
threadAboveNormal.Start();
threadHighest.Start();
}
}

class MyThreadClass
{
public void ThreadStart()
{
for (long count = 0; count <10000000; count++);
Console.WriteLine(
"Priority " + Thread.CurrentThread.Priority);
}
}
```

الجديد فى هذه السطور:

- فى هذه السطور تم تغيير درجة أولوية تنفيذ الـ Theads وذلك بتغيير قيمة الخاصية Priority كما فى السطور 1، 2، 3، 4.

حيث يتم إعطاء كل Thread درة أولوية مختلفة.

3- نفذ البرنامج F5 لحصل على نتجة التنفيذ كما فى الشكل (2-19).

الشكل (2-19)

التحكم فى تنفيذ الـ Thread

يمكن التحكم فى تنفيذ الـ Thread بحيث يمكن تغيير حالته من Start اى ابتداء تشغيله إلى إيقافه مؤقت إلى استعادة تشغيله إلى إيقافه نهائيا وهكذا ولتوضيح ذلك تابع المثال التالى

1- قم بإعداد تطبيق جديد من النوع Console Application.

2- اكتب به السطور التالية.

```
Imports System
Imports System.Threading
Module Module1
Sub Main()
Dim t1 As System.Threading.Thread
t1 = New System.Threading.Thread(AddressOf ThreadEntryMethod)
t1.Name = "Normal"
1.  t1.Start()
2.  t1.Suspend()
3.  Thread.Sleep(1000)
4.  t1.Resume()
5.  t1.Suspend()
6.  Thread.Sleep(1000)
7.  t1.Resume()
8.  t1.Suspend()
9.  Thread.Sleep(1000)
10. t1.Resume()
11. t1.Abort()
End Sub
Public Sub ThreadEntryMethod()
Dim r As Random = New Random(Thread.CurrentThread.GetHashCode())
Do While (True)
Dim i
Console.WriteLine("*****************")
Dim count As Integer = r.Next(1000)
Thread.Sleep(count)
Loop
End Sub
End Module
```

فى لغة C#

```
using System;
using System.Threading;
class ControlThread
{
    public static void Main()
    {
        MyThreadClass mtc = new MyThreadClass();
        ThreadStart ts = new ThreadStart(mtc.ThreadStart);
        Thread t = new Thread(ts);
        t.Start();

        for (int i=0; i<3; i++)
        {
        t.Suspend();
            Thread.Sleep(1000);
            t.Resume();
            Thread.Sleep(1000);
            Console.WriteLine();
        }
        t.Abort();
    }
}

class MyThreadClass
{
    public void ThreadStart()
    {
        while (true)
        {
            Console.Write("*");

            //sleep for 0.1 second
            Thread.Sleep(100);
        }
    }
}
```

فى هذه السطور:

- تم تغيير حالة الـ Thread من حالة إلى حلة وذلك من السطر رقم 1 إلى السطر رقم 11 كما يلى.
- فى السطر رقم 1 تم بدء تشغيل الـ Thread باستدعاء الدالة start().
- فى السطر رقم 2 تم إيقاف تشغيل الـ Thread ايقاف مؤقتا فى وضع الاستعداد وذلك باستدعاء الدالة Suspend().

- فى السطر رقم 3 يتم تعطيل العمليات لمدة 1000 ملى ثانية وذلك باستدعاء الدالة Sleep().
- فى السطر رقم 4 يتم استئناف تشغيل الـ Thread وذلك باستدعاء الدالة Resume().
- وهكذا باقى السطور حتى يتم انهاء عمل الـ Thead ايقاف نهائى فى السطر رقم 11.

فى لغة Java

أنواع المهام Threads فى لغة الجافا Java

يوجد نوعان من المهام Threads فى لغة الجافا Java وهما:

1. مهام المستخدم user Threads.
2. المهام الحارسة daemon threads.

إنشاء المهام Threads

توجد طريقتان لإنشاء المهام threads فى لغة الجافا Java:

1. أن ترث من الفصيلة class المسماة Thread.
2. أن تنفذ الـ interface المسمى Runnable.

كل طريقة لها مميزاتها وعيوبها واستخدام اى طريقة منهما تعتمد على احتياجك. فى الصفحات التالية سنعطى أمثلة على استخدام كلاً منهما ثم نبين مميزات وعيوب كلا الطريقتين.

أولاً: الوارثة من الفصيلة class المسماة Thread

يمكنك الوارثة من الفصيلة Thread مباشرة كما تعلمنا من قبل. فإذا أردت إنشاء فصيلة class اسمها MyThread وتريدها أن تنفذ مهمة Thread معينة فإنك ترث من الفصيلة Thread كالآتي

```
class  MyThread extends Thread
{
}
```

كما ذكرنا فإن المهمة thread هى وظيفة يمكنها أن تعمل فى نفس الوقت مع عدة وظائف أخرى فإذا أردت أن تكتب كود البرمجة الخاص بالوظيفة التى تريدها، فلابد من إضافة دالة تسمى () run وبداخلها تكتب الكود الذى ينفذ الوظيفة التى تريدها أى كالآتى:

```
class  MyThread extends Thread
{
public void run ()
        {
            \\ put your code here
         }
}
```

- الوظيفة التى تريد تنفيذها سوف تضيف الكود الخاص بها فى الدالة ()run.
- اتفقنا ان دالة الـ ()main تعتبر مهمة thread وهى الوحيدة التى تبدأ أوتوماتيكياً بدون تدخل المبرمج. إذن من داخل دالة الـ ()main فإنك تريد ان تبدأ المهمة thread التى عرفتها فى الكود الخاص بالدالة () run.
- أولاً لابد أن تنشئ هدف object من الفصيلة class المسماة MyThread كالآتى

```
MyThread firstThread = new Mythread ();
```

- الآن إذا أردت بدأ المهمة thread المعرفة فى الدالة () run فإنك تستخدم الدالة () start المعرفة فى الفصيلة class المسماة Thread. عندما تستدعى الدالة ()start فإن لغة الجافا Java سوف تستدعى الدالة ()run اى انك تبدأ المهمة thread كالآتى:

```
firstThread.start ();
```

- قد تسأل سؤالاً: لماذا لا نستدعى الدالة ()run مباشرة بدلاً من استدعاء الدالة ()start؟
- الإجابة هى أن الدالة () start هى التى تعنى بدء المهمة thread بالنسبة للغة الجافا Java فإذا استدعيت الدالة () run فستظن لغة الجافا Java أنك تستدعى دالة عادية عرفتها أنت كمبرمج. لاحظ أن الفصيلة MyThread هي فصيلة class عادية ويمكنك وضع أى دالة فيها بجانب الدالة ()run ولذلك تم وضع الدالة ()start ليبان ان الكود الذى سينفذ عند استدعائها هو مهمة thread بمعنى انه يمكن تنفيذ وظيفة أخرى معها. فإذا استدعيت الدالة ()run مباشرة فلن تكون مهمة thread فى هذه الحالة ولابد أن ينتهى الكود الخاص بها قبل استدعاء وتنفيذ اى دالة أخرى.
- لذلك تعود انك تستدعى الدالة ()start وهذه الدالة تنفذ الكود الموجود فى الدالة run () كمهمة thread أى انه سيتم تنفيذ هذه الدالة مع إمكانية تنفيذ وظيفة أخرى معها.
- الآن نريد مثالاً يوضح لنا كيف يمكننا تنفيذ أكثر من وظيفة فى نفس الوقت.

مثال 1 بدون المهام thread:

هذا المثال لا يمثل مهمة thread ولكننا وضعناه لبيان مشكلة ما وسوف نرى كيف يتم حلها باستخدام المهمة thread.

واليك كود البرمجة:

```
1:  class MyThread
2:  {
3:   public void start()
4:   {
5:        run();
6:   }
7:
8:   public void run()
9:   {
10:       while(true)
11:       {
12:            System.out.println("This is my thread");
13:       }
14: }
15:}
16:
17:class Main
18:{
19: public static void main (String[]args)
20: {
21:       MyThread firstThread = new MyThread ();
22:       firstThread.start();
23:
24:       while(true)
25:       {
26:            System.out.println("This is main thread");
27:       }
28: }
29:}
```

شرح الكود:

- في السطر 1 يتم إنشاء فصيلة class تسمى MyThread. هذه الفصيلة class تحتوى على دالة تسمى () start كما فى سطر 3.
- هذه الدالة () start تستدعى الدالة الأخرى المسماة () run كما فى سطر 5.

- في السطر 8 يتم تعريف الدالة () run.
- في السطر 10 يتم إنشاء دوارة لا نهائية infinite loop باستخدام while.
- في السطر 12 يتم طباعه رسالة على الشاشة.
- في السطر 17 يتم تعريف فصيلة class اسمها Main وهي تحتوى على الدالة () main.
- في السطر 21 يتم إنشاء هدف object من نوع الفصيلة class المسماة MyThread.
- في السطر 22 يتم استدعاء الدالة () start المعرفة فى الفصيلة class المسماة MyThread.
- في السطر 24 نبدأ دوارة لا نهائية infinite loop باستخدام while.
- في السطر 26 يتم طباعه رسالة على الشاشة.
- الآن يبدأ تنفيذ البرنامج باستدعاءك الدالة ()start كما فى سطر 22 وكما ذكرنا من قبل فإن هذه الدالة تستدعى الدالة ()run التى تطبع رسالة على الشاشة بصورة لا نهائية.
- إذا نفذت البرنامج فستجد طباعة الرسالة "This is My Thread" بصورة لا نهائية. انظر شكل (3-19).

شكل (3-19) المثال الأول

- إذن ما العمل إذا أردت طباعة الرسالة "This is My Thread" ثم فى نفس الوقت تنفذ باقى الكود الموجود فى دالة الـ ()main الذى يطبع الرسالة "This is Main Thread"؟
- الحل هو استخدام الوارثة من الفصيلة class المسماة Thread كما فى المثال التالى:

مثال 2: استخدام الفصيلة Thread

كود البرمجة:

```
1:  class MyThread extends Thread
2:  {
3:   public void run()
4:   {
5:        while(true)
6:        {
7:        System.out.println("This is my thread");
8:        }
9:   }
10:)
11:
12:class Main
13:{
14: public static void main (String[]args)
15: {
16:       MyThread firstThread = new MyThread ();
17:       firstThread.start();
18:
19:       while(true)
20:       {
21:            System.out.println("This is main thread");
22:       }
23: }
24:}
```

شرح الكود:

- لاحظ عدم وجود تعديل في الفصيلة class المسماة Main.
- التعديل حدث في الفصيلة class المسماة MyThread فأولاً جعلناها ترث من الفصيلة Thread ثم حذفنا الدالة ()start تماماً.
- لاحظ أن سطر 17 يستدعي الدالة ()start وقد تظن أنها غير موجودة بعد أن حذفناها ولكن تذكر حيث أننا ورثنا من الفصيلة Thread والتي يحتوي على الدالة ()start وهذه الدالة ()start هي التي تستدعي الدالة () run لتنفيذها كمهمة thread.
- الآن نفذ البرنامج ولاحظ طباعة الرسالتين "This is My Thread" و "This is main Thread" بطريقة لا نهائية.

النتيجة:

- أنه بالرغم من أن الدالة ()run تمثل دوارة لا نهائية ولكن أمكن تنفيذ دالة أخرى معها وهو باقى الكود فى الدالة ()main. اى أننا لم ننتظر انتهاء الدالة ()run لتنفيذ باقى الكود ولكن تم تنفيذ الاثنين فى نفس الوقت وهذه هي المهمة thread. انظر شكل (4-19).
- طبعاً ستقول أن هذا البرنامج السابق بلا فائدة؟ معك حق ولكننا وضعناه فقط للتوضيح وسترى فى أخر الفصل تطبيق عملى للمهام threads .

شكل (4-19) المثال الثانى

تنفيذ الـ interface المسمى Runnable

ستجد اختلافاً بسيطاً بين الوراثة من الفصيلة Thread وتنفيذ الـ interface المسمي Runnable وهذا سيتم توضيحه فى المثال التالى.

مثال 3: استخدام الـ interface المسمي Runnable

كود البرمجة:

```
1:  class MyThread implements Runnable
2:  {
3:   public void run()
4:   {
5:        while(true)
6:        {
```

```
7:              System.out.println("This is my thread");
8:      }
9:  }
10:}
11:
12:class Main
13:{
14: public static void main (String[]args)
15: {
16:      MyThread firstThread = new MyThread ();
17:      Thread t = new Thread(firstThread);
18:      t.start();
19:
20:      while(true)
21:      {
22:          System.out.println("This is main thread");
23:      }
24: }
25:          }
```

شرح الكود:

- اختلاف بسيط حدث بالنسبة للفصيلة class المسماة MyThread فبدلاً من الوراثة من الفصيلة Thread فهي تنفذ الـ interface المسمي Runnable كما في سطر 1 ولا يوجد تغيير آخر.
- في السطر 16 يتم تعريف هدف object من نوع الفصيلة MyThread ولا جديد هنا.
- الجديد هو سطر 17 وله أهمية قصوى وذلك لأن الـ interface المسمى Runnable لا يحتوى على الدالة start() ولذلك لا يمكن أن نكتب.

```
firstThread.start();
```

- وإلا سيعطى رسالة خطأ. إذن ما العمل؟
- لاحظ أن الدالة start() موجودة في الفصيلة class المسماة Thread ولذلك لابد أن ننشئ هدف object من نوع الفصيلة Thread ونمرر لدالة البناء constructor الخاصه بالفصيلة Thread، هدف object من نوع الـ interface المسمي Runnable اى كالآتي:

```
Thread t = new Thread (firstThread);
```

- إذن هنا أنشأنا هدف object من نوع الفصيلة Thread ومررنا له الفصيلة class الفعليه التي تحتوى على الكود الخاص بالمهمة Thread التي تريد تنفيذها كما في سطر 17.

- حيث أن t هو هدف object من نوع الفصيلة Thread، إذن فهو يحتوى على الدالة ()start ولذلك نستطيع استدعاء هذه الدالة كما فى سطر 18.
- نفذ البرنامج وستلاحظ الحصول على نفس نتيجة البرنامج السابق الذى تم تنفيذه باستخدام الوارثة من الفصيلة Thread.

الرسم التالى يوضح عملية بدء الـthread فى البرنامج السابق.

النتيجة:

- لاحظت ظهور نفس النتيجة باستخدام الوراثة من الفصيلة Thread وتنفيذ الـ interface المسمي Runnable إذن ما الفرق بينهما ومتى نستخدمها؟
- تذكر أن لغة الجافا Java لا تدعم التوارث المتعدد Multiple Inheritance ولذلك إذا ورثت من الفصيلة Thread، فلا يمكنك أن ترث من أى فصيلة class أخري وهذا يعتبر عائقاً لك في بعض الحالات.
- أما إذا كنت تريد أن تنفذ مهمة thread وفى نفس الوقت ترث من فصيلة class أخري، ففي هذه الحالة لا بد من استخدام الـ interface المسمي Runnable.
- ولكن الفصيلة class المسماة Thread تحتوى على الدالة ()start التى تبدأ المهمة thread كما تحتوى الفصيلة Thread على عدة دوال أخرى مفيدة ستذكر لاحقاً كما أن استخدام الفصيلة Thread يعتبر أسهل من استخدام الـ interface المسمي Runnable.
- كملخص: إذا كنت ستنشئ الفصيلة class الخاصة بإنشاء المهمة thread ولن تحتاج أن

ترث من فصيلة class أخري فاستخدم الفصيلة Thread لأنها أسهل وتوفر لك الدوال التى تريدها مباشرة أما إذا كنت سترث من فصيلة class أخري فاستخدم الـ interface المسمي Runnable.

بعض الدوال المهمة للمهام Threads

1. الدالة ()currentThread

عادة يكون عندك أكثر من مهمة thread تعمل فى نفس الوقت فإذا أردت الحصول على هذف object من المهمة thread الحالية فاستخدم الدالة ()currentThread كما سنري في المثال القادم.

2. الدالة ()getName

كل مهمة thread لها اسم فمثلاً مهمة دالة الـ ()main اسمها ايضاً main أما المهمة thread التى تنشئها بنفسك فتأخذ اسم Thread-0,Thread-1 ...وهكذا.

3. الدالة ()setName

إذا لم تكن تعجبك أسماء Thread-0,Thread-1 وهكذا، كأسماء للمهمة thread، فبإمكانك تغيير اسمها عن طريق الدالة ()setName.

4. الدالة () sleep

وهى دالة لجعل المهمة thread تتوقف مؤقتاً عدداً من الثوانى يتم تحديده بمعرفة المبرمج. هذه الدالة تستقبل معامل من نوع long لتحدد كم ألف جزء من الثانية تريد أن تتوقف بمعنى أنك إذا أردت أن تجعل المهمة thread تتوقف لثانية واحدة فتمرر للدالة ()sleep القيمة 1000.

أى أن وحدة القياس هنا هى المللى ثانية وليس ثانية. لاحظ أنك تمرر قيمة long أى لا يوجد بها كسور أو أرقام عشرية.

5. الدالة () setPriority

حيث أنه عندك عدة مهام تنفذ جميعها فى نفس الوقت، فمن الطبيعى أن يكون لكل

مهمة أولوية معينة فبعض المهام تكون حرجة ولابد أن تتم فى أسرع وقت أما البعض الأخر فلا تمثل السرعة فارقاً ولا يهمنا أن تتم سريعاً.

الدالة () setPriority تحدد أولوية تنفيذ المهمة thread بحيث أنك تمرر للدالة معامل من نوع int وهذا المعامل لا بد أن يكون رقماً من 1 إلى 10 بحيث أن القيمة 1 تمثل الأقل أهمية والقيمة 10 تمثل الأهمية القصوى والقيمة 5 تمثل الأهمية العادية.

- إذا لم تحدد أولوية المهمة فإن لغة الجافا Java تعتبر الأولوية تساوى 5 أى تعطى لكل مهمة أولوية عادية ولا تكمل مهمة معينة على حساب مهمة أخرى.

الأمثله التالية توضح استخدام هذه الدوال

مثال 4: توضيح استخدام بعض الدوال المهمة للمهام Threads

كود البرمجة:

```
1:  class MyThread implements Runnable
2:  {
3:   public void run()
4:   {
5:         Thread myThread = Thread.currentThread();
6:         myThread.setName("MyThread");
7:         System.out.println(myThread.getName());
8:   }
9:  }
10:
11: class Main
12: {
13: public static void main (String[]args)
14: {
15:         MyThread firstThread = new MyThread ();
16:         Thread t = new Thread(firstThread);
17:         t.start();
18:
19:         Thread myThread = Thread.currentThread();
20: myThread.setName("Main Thread");
21: System.out.println(myThread.getName());
22: }
23:          }
```

شرح الكود:

- في السطر 1 يتم تعريف الفصيلة class المسماة MyThread والتي سنستخدمها لإنشاء مهمة thread ولذلك فهي تنفذ الـ interface المسمي Runnable وتنشئ فيها الدالة run () كما في سطر 3.
- في السطر 5 يتم إنشاء هدف object من نوع الفصيلة Thread اسمها myThread . ولجعل الهدف myThread يشير إلى المهمة thread الحالية، استخدمنا الدالة currentThread() . لاحظ ان الدالة currentThread() هي دالة static ولذلك استخدمنا اسم الفصيلة class المسماة Thread مباشرة دون الحاجه إلى إنشاء هدف object منها.
- في السطر 6 يتم إعطاء المهمة thread الحالية اسم "My Thread" وبالطبع يمكنك اختيار أي اسم .
- في السطر 7 يتم طباعه اسم المهمة thread الحالية. وقد حصلنا على اسم المهمة thread الحالية عن طريق الدالة getName().
- السطور 5،6،7 كررناها مرة أخرى في سطور 19،20،21 ولكن بالنسبة لدالة الـ main(). يمكنك الآن تنفيذ البرنامج ولاحظ طباعة اسم كل مهمة thread كما حددناه. انظر شكل (5-19).

شكل (5-19) المثال الرابع

المثال التالى يوضح الدالة ()sleep

مثال 5: توضيح استخدام الدالة ()sleep باستخدام الـ interface المسمى Runnable

نريد أن نطبع الأرقام من صفر إلى 10 مع وجود فارق زمنى يساوي ثانية واحدة بين طباعة كل رقم والذى يليه.

كود البرمجة:

```
1:  class MyThread implements Runnable
2:  {
3:   public void run()
4:   {
5:        Thread myThread = Thread.currentThread();
6:
7:        for(int i = 0 ; i <= 10 ; i++)
8:        {
9:             System.out.println(i);
10:
11:            try
12:            {
13:            myThread.sleep(1000);
14:            }
15:
16:            catch (InterruptedException e)
17:            {
18:            System.out.println(e);
19:            }
20:       }
21: }
22:}
23:
24:class Main
25:{
26: public static void main (String[]args)
27: {
28:       MyThread firstThread = new MyThread ();
29:       Thread t = new Thread(firstThread);
30:       t.start();
31: }
32:      }
```

شرح الكود:

- المشكلة التي تحدث عند تنفيذ الـ interface المسمى Runnable أنه لا يحتوى على الدالة ()sleep ولكنها موجودة في الفصيلة Thread، إذن نريد إنشاء هدف object من نوع الفصيلة Thread ليشير إلى المهمة thread الحالية ولذلك استخدمنا الدالة ()currentThread كما في سطر 5.
- في السطر 7 يتم إنشاء دوارة من صفر إلى 10.
- في السطر 9 يتم طباعه قيمة المتغير i والذي يمثل قيمة العداد.
- في السطر 13 يتم جعل المهمة thread تتوقف مؤقتاً لمدة ثانية واحدة.
- لاحظ أن الهدف object المسمي myThread (سطر 5) يشير إلى المهمة thread الحالية وأنه من نوع الفصيلة Thread إذن فهو يحتوى على الدالة ()sleep وبذلك نستطيع ان نوقف المهمة thread مؤقتاً عن طريق استدعاء الدالة ()sleep.
- محاولة استخدام الدالة ()sleep قد ينتج عنها حدوث استثناء Exception واسمه InterruptedException ولذلك استخدمنا try و catch لمعالجة هذا الاستثناء Exception
- أخيراً بعد تنفيذ البرنامج لاحظ طباعة الرقم صفر ثم التوقف لمدة ثانية ثم طباعة الرقم 1 ثم التوقف لمدة ثانية وهكذا حتى رقم 10. انظر شكل (6-19).

شكل (6-19) المثال الخامس