

- كتاب برمجي عربي متخصص في مخاطبة الدارسين من الكليات المتوسطة.
- أسلوب مبسط وميسّر دون ترجمة حرفية أو ألفاظ عربية غريبة.
- أكثر من 150 شكل توضيحي لإيصال المعلومة بأسلوب أسهل.
- أكثر من 500 مثال توضيحي وتمرين ذاتي الحل.
- تنبیهات متواصلة حول الأخطاء الشائعة.
- تلميحات للتركيز على المعلومات المهمة.
- ملفات عرض داعمة لكل وحدة.

## المفيد

# في الخوارزميات ومبادئ البرمجة

بلغة جافا

أ. محمود رفيق الفرا

المحاضر بقسم علوم الحاسوب وتكنولوجيا المعلومات  
الكلية الجامعية للعلوم والتكنولوجيا - خان يونس- فلسطين

الطبعة الثانية 2015

اقتداءً بأئمَّة السلف الصالح في تصنيفهم للمصنفات؛ ابدأ  
هذه الطبعة بحديث النِّيَّات تذكيرًا النفسي وإخواني وأسائل  
الله لنا ولكلم الإخلاص والقبول.

عَنْ أَمِيرِ الْمُؤْمِنِينَ أَبِي حَفْصٍ عُمَرَ بْنِ الْخَطَّابِ - رضي  
الله عنه - قَالَ: سَمِعْتَ رَسُولَ اللَّهِ - صلَّى اللهُ عَلَيْهِ  
وَسَلَّمَ - يَقُولُ: "إِنَّمَا الْأَعْمَالُ بِالنِّيَّاتِ، وَإِنَّمَا لِكُلِّ امْرِئٍ  
مَا نَوَى، فَمَنْ كَانَتْ هِجْرَتُهُ إِلَى اللَّهِ وَرَسُولِهِ فَهِجْرَتُهُ إِلَى  
اللَّهِ وَرَسُولِهِ، وَمَنْ كَانَتْ هِجْرَتُهُ لِدُنْيَا يُصِيبُهَا أَوْ امْرَأَةٌ  
يَنْكِحُهَا فَهِجْرَتُهُ إِلَى مَا هَاجَرَ إِلَيْهِ".

حديث صحيح

هذا الكتاب مجهود بشرٍ خالص وهو بالتالي  
مُعَرَّض للخطأ من أي نوع؛ فرجاءً من يلحظ أي  
خطأً من أي نوع أن يقوم بمراسلتي وفق البريد  
الإلكتروني التالي:

[m.farra@cst.ps](mailto:m.farra@cst.ps)

داعياً الله أن يجزيَه كُلَّ خير

## مقدمة المؤلف للطبعة الأولى :

الحمد لله وحده والصلوة والسلام على من لا نبي بعده محمد صلى الله عليه وسلم على آله وأصحابه وأزواجه أمهات المؤمنين وسلم تسليماً كثيراً وبعد ، ،

قناعني بالتعلم الإلكتروني كوسيلة داعمة للعملية التعليمية في شتى الجامعات والكليات، لا يمنع قناعني بأن الكتاب الجامعي يظل عاملًا لا بد من توفره بين يدي الطالب، لما لوجوده من أهمية كبيرة لدى كثيرون من الطلبة والمعلمين من حيث تعود الطلبة على التعامل معه والاستفادة منه.

لهذا بحثت طلابي كثيراً عن كتاب دراسي لأهم مساقات تخصصهم فلم أجد كتاباً يلبي الاحتياجات الأساسية من حيث الأسلوب الميسر، البعد عن الترجمة الحرفية للكتب الأجنبية وتتوفر عدد كبير من الأمثلة المحلولة والتمارين ذاتية الحل بأفكار تختلف وتنعد، وهو ما دفعني وبقناعة كبيرة أن أسخر جل وقتى لخدمة طلابي والمجتمع العربي من خلال تأليف كتاب ميسر في تسلسله وأسلوبه، متخصص في كونه يخاطب طلبة الدبلوم المتوسط أو المبتدئين في عالم البرمجة، فكان كتاب "المفید في الخوارزمیات و مبادئ البرمجة بلغة جافا". ولأنني أفتتح كامل القناعة بأن لا نجاح بلا تفكير سليم وخطيب صحيح، فقد جعلت تركيزى ينصب على إيصال هذه القناعة لكل من يقرأ في هذا الكتاب بداية من تخصيص باب كامل للخوارزميات مدعوماً بأكثر من خمسين فكرة مختلفة. كما وضعت في هذا الكتاب ما يزيد عن خمسين تدريبيات والأمثلة والتمارين ذاتية الحل، وجعلت أضع الحلول لبعضها وأنترك الأخرى توفيراً لمساحة مناسبة بيني وبين الدارس للتفاعل والتبادل.

هذا الكتاب هو عمل متواضع أقدمه لطلابي وزملائي، قد يحتوي على الكثير من الأخطاء، لذا أرجو أن يساهموا معي في تقييده وتصويبه من خلال مراسلتي لتكبر هذه التجربة وتصبح أكثر نفعاً للجميع ولتكنون بداية لمزيد من الكتب الدراسية النافعة والمخصصة للطلبة في هذه المرحلة .

وأخيراً أسأل الله العظيم أن يجعل كل ما في هذا الكتاب مفيداً للإسلام والمسلمين وأن يفيد به طلابنا وطالباتنا وعلى الله التكلال ومنه التوفيق والقبول. والحمد لله رب العالمين .

محمود رفيق الفرا

السابع والعشرين من صفر لعام 1432 هجري

الموافق الأول من فبراير لعام 2011 ميلادي

## مقدمة المؤلف للطبعة الثانية :

الحمد لله الذي يُشَغِّلُ مَنْ يُحِبُّ فِيمَا يُحِبُّ والصلوة والسلام على محمد صلى الله عليه وعلى آله وأصحابه وأزواجه أمهات المؤمنين وسلم تسليماً كثيراً وبعد ،،،

لعل من أثيل وأروع المبادئ التي يمكن للإنسان أن يعيش لأجلها هي غرس الخير، لاسيما في التعليم وخاصة في تأليف وإعداد المراجع العلمية بلغة القرآن الكريم، لإثراء المكتبة العربية العلمية وتطوير المحتوى العربي، وفي ظل الحاجة المتواصلة لتوفير مراجع عربية أصيلة تتحدث في العلوم التطبيقية وخاصة علوم الحاسوب وتكنولوجيا المعلومات، تشجعت على تطوير محتوى هذا الكتاب وصولاً لهذه الطبعة الثانية بمواضيع إضافية تتافق مع احتياجات سوق العمل والفئة المستهدفة؛ فبالإضافة إلى الأسلوب الميسر والبعد عن الترجمة الحرافية للكتب الأجنبية، وتتوفر ما يزيد عن خمسمائة من الأمثلة المحلولة والتمارين ذاتية الحل والتي تميزت بها الطبعة الأولى، اجتهدت في هذه الطبعة إلى أن توفر مواد إلكترونية مساعدة للكتاب؛ من أبرزها شرائح العرض لمعظم الأبواب وسلسلة محاضرات مرئية موازية لأكثر مواضيع الكتاب، مما يجعل هذا الكتاب منهاجاً متكاملاً يساعد الدارسين والمحاضرين على حد سواء كما يُشجع ويُساعد الراغبين في خوض غمار التعلم الذاتي.

في الوقت ذاته حافظت على أسلوبى في الطبعة الأولى في التركيز على الخوارزميات وإيصال المعلومات والأفكار اعتماداً على أسلوب التحليل المتسلسل والمنطقى لكل فكرة أو مشكلة برمجية مقتضاها أن هذا هو الأسلوب الأمثل والأفضل في شرح علوم البرمجة للطلبة المبتدئين عاملاً وطلبة الكليات المتوسطة بشكل خاص. ورغم ذلك يبقى هذا العمل متواضعاً، أقدمه لطلبتي وزملائي، وقد يحتوى على أخطاء أو هفوات، وهنا أرجو أن يساهموا جميعاً معي في تقييده وتصويبه من خلال مراسلتي لتكرر هذه التجربة وتصبح أكثر نضجاً وفعلاً للجميع لنواصل تأليف وإعداد الكتب الدراسية النافعة والمخصصة للطلبة في هذه المرحلة.

وأخيراً أسأل الله العظيم أن يجعل هذا الكتاب مفتاح خير الإسلام والمسلمين وأن يفيد به الجميع، والحمد لله رب العالمين.

## المؤلف

الحادي والعشرين من محرم لعام 1437 هجري  
الموافق الأول من نوفمبر لعام 2015 ميلادي

## إهداء وشكر

أهدى هذا الجهد العلمي إلى والدي الحبيبين الرحيمين و زوجتي الصبوره وقره عيوني غدير ورفيق وأهلي وأحبابي في كل مكان؛ كما أهديه لكل من له فضل علىِّ ممن علموني أو ساعدوني أو دعوا الله لي بظهر الغيب.

ولكلَّ محبٍ ومتبعٍ لنهج محمد صلى الله عليه وسلم.  
سائلًا المولى جلَّ شأنه أن يجمعني بهم بصحبة النبئين  
و الصديقين يوم الدين .

تقديري إلى كل من ساهم في هذا العمل بمراجعة أو نصيحة وأخص بالذكر الزميل العزيز د. جمال العطار رئيس وحدة البحث العلمي بكلية الجامعية للعلوم والتكنولوجيا. وأ. محمود قشطة أحد خريجي تخصص تكنولوجيا المعلومات من الكلية والباحث في جامعة أسطنبول ايدن. وأ. محمد أبوشاويش المصمم الجرافيكى بوحدة التعليم الإلكتروني والوسائط المتعددة بكلية على جهودهم في تحرير الكتاب ومراجعته وتصميمه.

جزاكم الله خيراً

**قبل أن تبدأ الدراسة في هذا الكتاب**

عزيزي الدارس من المفضل قبل أن تبدأ الدراسة في هذا الكتاب أن تأخذ مجموعة من الأمور بعين الاعتبار وهي على النحو التالي:

**أولاً، أسلوب كتابة المفاهيم والرسوم التوضيحية /**

لقد قمت باستخدام مجموعة من الرموز والأشكال التي لها معنى معين لإعطاء الشرح في الكتاب مزيداً من الوضوح والسلامة وهي للتوضيح على النحو التالي:

1. تم تمييز القطع البرمجية في جدول مظلل، مع ترقيم أسطر الجمل البرمجية ليسهل الشرح ويسهل تتبع الجمل البرمجية وذلك بشكل خاص في الأمثلة التوضيحية.

2. تم تمييز مجموعة من المعلومات البرمجية المهمة ومجموعة من التبيهات حول الأخطاء الشائعة بأشكال توضيحها كما يلي:

وهذا الشكل يشير إلى وجود معلومة برمجية مهمة قد تكون ذكرت خلال الشرح ولكنني فضلت تمييزها لأهميتها ولضرورة أن يهتم الدارس بمعرفتها وفهمها بشكل دقيق.



3. اجتهدت عند ذكر المصطلحات أن أذكر المرادف لها باللغة الإنجليزية وذلك نظراً لوجود عدة ترجمات لها باللغة العربية.

4. في بداية كل باب وضع لك قائمة بالوسائل الإلكترونية المتاحة مع هذا الكتاب من خلال الشكل التوضيحي التالي:

يتوفر لهذا الباب شرائح العرض "PowerPoint" وكذلك ملفات مرئية. يمكنك الحصول عليها من خلال [mfarra.cst.ps](http://mfarra.cst.ps)

**ثانياً، الأمثلة والتمارين /**

نتيجة للصعوبة الكبيرة التي يجدها الدارسون المبتدئون في تعلم البرمجة والناتجة في الغالب عن قلة الأمثلة أو التدريبات التي يحاولون الإجابة عليها أو قلة الأفكار التي يتمرنون عليها؛ فقد قمت وبفضل الله تعالى في هذا الكتاب بتوفير ثلاثة أنواع من الأمثلة والتمارين المختلفة على النحو التالي:

1. الأمثلة التوضيحية: وقد استخدمتها خلال الشرح لتوضيح أساسيات موضوع الشرح، فيتم شرح كل جزئية بتفصيل كامل، وهذا النوع عدده قليل نسبياً في الكتاب، والهدف منه تفصيل كافة الجزئيات للدارس.

2. الأمثلة والتدريبات العامة: أقوم بطرحها وحلها مع شرح موجز للفكرة وطريقة الحل وأحياناً لكيفية الوصول للحل، هذه الأمثلة والتمارين تأخذ أنماطاً مختلفة مثل (إكمال الفراغ، تحديد صحة الإجابة أو خطتها، التعبير عن الجمل العربية بجمل تناسب قواعد لغة جافا بالإضافة إلى تدريبات برمجية، ...).

3. التدريبات ذاتية الحل: وهي مجموعة من التدريبات التي أطربها للدارس ليتمكن على حلها ويحاول البحث والتفكير في حلها مما يزيد من قدرته على اكتساب المهارة في حل المسائل المختلفة وبالتالي تزيد قدراته البرمجية.

### ثالثاً، أهداف ومهارات /

سعياً لتوفير بيئة تعليمية مميزة للدارس من خلال هذا الكتاب، اجتهدت في تحديد الأهداف التعليمية والمهارات الفنية لكل باب من أبواب الكتاب، وذلك ليتسنى للدارس أن يراقب تحصيله الدراسي أولاً بأول وأن يقيّم معرفته ومماراته بنهاية كل باب، كما يمكن للمحاضر الاعتماد عليها عند وضع الخطة التدريسية، وكذلك عند وضع الامتحانات.

### رابعاً، البرمجيات الازمة /

اعتماداً على التشابه الواضح بين مفاهيم البرمجة الأساسية في اللغات المختلفة وخاصة تلك اللغات المعتمدة على مبدأ البرمجة شبيهة التوجه (*Object-Oriented Programming*)، فإنَّ هذا الكتاب يمكن اعتماده كمراجع لتعلم أكثر من لغة فيما يتعلق بالمفاهيم والتفكير والأمثلة والتمارين ونحوه، لكننا نعتمد شرح قواعد البرمجة لغة جافا (*Java*) فقط من أجل التركيز، وبالتالي من الضروري خلال دراستك لهذا الكتاب أن تقوم في البداية بتنصيب البرمجيات الازمة لعمل لغة جافا وهي *Java Development Kit* (*JDK*) ومحرر مناسب مثل (*JCreator*) أو (*Eclipse*) أو (*NetBeans*) أو (*YouTube*).  
وبكل تأكيد هذا مهم جداً لفهمك لكافة البرامج والتطبيقات التي سنقوم بشرحها خلال أبواب الفصل المختلفة، ويمكنك التعرف على كيفية تنصيب هذه البرمجيات من خلال زيارة موقعي الأكاديمي أو قناتي على

*YouTube*

يمكنك مشاهدة كيفية تنصيب البرنامج من خلال زيارة [mfarra.cst.ps](http://mfarra.cst.ps) أو قناتي على

[www.youtube.com/mralfarra1](http://www.youtube.com/mralfarra1)



### خامساً، منهجي في اعتماد المصطلحات العربية البرمجية المرادفة للأجنبية /

يحتوي علم البرمجة على كثيرٍ من المصطلحات الأجنبية التي يتوفر لها أكثر من رديف عربي ولكنه لا اعتمد أي رديف إلا بعد التحقق من أنه مناسب، حيث أنَّ الترجمة الحرافية لهذه المصطلحات يفقدها معناها الحقيقي،

وتحل فهمها أمر بالغ الصعوبة على الدارس العربي، لأن بعضها يفقد قيمته عندما يتم ترجمته دون ربطه بطبيعة عمله، ولذلك اجتهد في دراسة الخيارات المتعددة لكل مصطلح، واخترت الأفضل من وجهة نظرى، وهنا أسرد أبرز هذه المصطلحات وهي ليست بالكثيرة.

سبب الاختيار	الخيارات العربية	المصطلح باللغة الانجليزية
كتاب خيار		
أن هذه الطريقة تستخدم في وصف الحل المقترن لتطوير الحل البرمجي لأى مشكلة، وبالتالي فهي أقرب لكونها وصفية من كونها مزيفة أو مجرد، وإن كانت مجرد أيضاً مقبولة إن قصدنا مجردة من الأوامر البرمجية المتعلقة بلغة محددة.	• الشيفرة المزيفة • الشيفرة الوصفية • الشيفرة المجردة	<b>pseudocode</b>
كلا المصطلحين مقبول من وجهة نظرى وتم اعتماد جمل التكرار في الكتاب لأنه الأقرب لفهم من الناحية الوظيفية.	• جمل الدوران • جمل التكرار	<b>Repetition statements</b>
(الطرق والدوال) كلاهما مقبول من وجهة نظرى، لكن "الدوال" أقرب لفهم حيث أنها في لغة جافا تعتبر مهمة محددة جاهزة للعمل بمجرد استدعائهما أما كلمة "طريقة" هي أقرب إلى وصفة معينة يمكننا السير على خطواتها لتصل إلى ما نريد دون إرجاع قيم بعض الدوال التي تتمكن من إعادة القيم.	• الطرق • الدوال • الاقتران • الإجراء	<b>Methods</b>
أن هذا الأسلوب يتعامل مع أي مشكلة على أنها مجموعة من العناصر الحية أو غير الحية وبالتالي فإن أقرب مفهوم عربي لكلمة ( <i>Object</i> ) هي شيء وليس كائن، وجود كلمة ( <i>Oriented</i> ) يجعلنا بحاجة لوجود كلمة التوجه في المصطلح العربي.	• البرمجة الشيئية • البرمجة الهدفية • البرمجة كائنية التوجه • البرمجة شيئية التوجه • البرمجة غرضية التوجيه	<b>Object Oriented Programming</b>
المصطلح يعني أن المتغير مشترك بين كافة الكائنات للصنف الواحد ويحق لهم التعديل عليه، لذا لا يمكن ترجمته (ثابت أو ساكن) لأن هذا يتعارض مع طبيعته الوظيفية.	• المتغير الثابت • متغير الصنف • المتغير المشترك • المتغير الساكن	<b>static variable</b>
حيث أن المقصود بها هي الدوال التي يمكن استخدامها مباشرة من الصنف دون الحاجة لاستقاص كائن من الصنف، وهي وبالتالي لا يصح القول أنها ثابتة ولكن يصح نسبها للصنف حيث يوجد رديف أجنبى آخر لها يطلق عليه ( <i>Class Method</i> ).	• الدوال الثابتة • دوال الصنف	<b>static Method</b>

## المحتويات

1

الباب الأول: مقدمة للحاسوب والحوسبة

1. مقدمة
2. ما هو الحاسوب؟
3. تنظيم الحاسوب ومكوناته
4. أنظمة العد
5. مفهوم الحوسبة وال الحاجة
6. تدريبات عامة
7. تمارين ذاتية الحل

14

الباب الثاني: الخوارزميات وطرق حل المشاكل

1. مقدمة
2. خطوات حل المشكلة
3. طريقة الشيفرة الوصفية (*Pseudocode*)
4. طريقة مخطط سير العمليات (*Flowchart Diagram*)
5. أمثلة وتدريبات عامة
6. تمارين ذاتية الحل

52

الباب الثالث: مبادئ البرمجة

1. مقدمة
2. الحاجة للبرمجة وللغاتها
3. تصنیفات لغات البرمجة
4. مراحل حل المشاكل باستخدام الحاسوب
5. مميزات لغة جافا
6. مكونات لغة جافا
7. العمليات الحسابية
8. جمل الإدخال والإخراج
9. أمثلة وتدريبات عامة
10. تمارين ذاتية الحل

96

الباب الرابع : جمل الاختيار (*Condition Statements*)

1. مقدمة

2. أسس بناء الشروط المنطقية

3. فهم منطق اتخاذ القرار وتطبيقاته العملية

4. أنواع الجمل البرمجية لاتخاذ القرار

✓ جملة الشرط ذات النتيجة الواحدة (*Simple if*)✓ جملة الشرط ذات النتيجتين (*if .. else*)✓ جملة الشرط متعددة الشروط (*nested if*، *if .. else if*)✓ جملة (*switch case*)

✓ الشروط المركبة

✓ معامل الشرط (:?)

5. أمثلة وتدريبات عامة

6. تمارين ذاتية الحل

130

الباب الخامس: جمل التكرار (*Repetition Statements*)

1. مقدمة

2. فهم منطق التكرار في البرمجة والتطبيقات العملية لها

3. استخدام جملة الدوران (*while*)4. استخدام جملة الدوران (*do ... while*)5. استخدام جملة الدوران (*for*)6. استخدام جمل (*break & continue*) وتوسيع أثراها

7. أنواع الأخطاء

8. جمل الدوران المتداخلة

9. أمثلة وتدريبات عامة

10. تمارين ذاتية الحل

163

الباب السادس: المصفوفات (*Arrays*)

1. مقدمة

2. مفهوم المصفوفات وال الحاجة لها

3. تعريف وإنشاء المصفوفات أحادية الأبعاد

4. جملة التكرار المحسنة للمصفوفات (*foreach*)
5. تعريف وإنشاء المصفوفات متعددة الأبعاد
6. أمثلة وتدريبات عامة
7. تمارين ذاتية الحل

185

الباب السابع: الدوال

1. مقدمة
2. مفهوم الدوال وال الحاجة إليها
3. تعريف وإنشاء الدوال
4. استدعاء الدوال
5. تمرير المصفوفات للدوال
6. المتغيرات المشتركة للكائنات (*static variables*)
7. دوال الأصناف (*static Methods*)
8. استخدام الدوال الجاهزة
9. مفهوم المتغيرات المحلية والعامة (*Local and Global variables*)
10. الاستدعاء الذاتي (*Recursion*)
11. مفهوم التحميل الزائد للدوال (*Overloading Methods*)
12. أمثلة وتدريبات عامة
13. تمارين ذاتية الحل

222

الباب الثامن: خوارزميات البحث والترتيب

1. مقدمة
2. الحاجة إلى عملية البحث وتطبيقاتها العملية
3. خوارزميات البحث
  - ✓ خوارزمية البحث الخطي (*Linear Search Algorithms*)
  - ✓ خوارزمية البحث الثنائي (*Binary Search Algorithms*)
4. الحاجة إلى عملية الترتيب وتطبيقاتها العملية
5. خوارزميات الترتيب
  - ✓ خوارزمية الفقاعات الهوائية (*Bubble Sort Algorithm*)

✓ خوارزمية الاختيار (*Selection Sort Algorithm*)

6. أمثلة وتدريبات عامة

7. تمارين ذاتية الحل

239

الباب التاسع: أساسيات البرمجة شبيهة التوجه

1. مقدمة

2. مفهوم البرمجة شبيهة التوجه (*Object-Oriented Programming*)

3. مميزات البرمجة شبيهة التوجه

4. مفهوم الأصناف والكائنات

5. الدوال البنائية (*Constructors Methods*)6. محددات الوصول (*Access modifiers*)7. تطبيق مصفوفة الكائنات (*Array Of Objects*)

8. تطبيق الوراثة في لغة جافا ومفاهيمها

9. تطبيق تعدد الأشكال في لغة جافا ومفاهيمه

10. أمثلة وتدريبات عامة

11. تمارين ذاتية الحل

286

المراجع

287

المواد الإلكترونية المساعدة

إنَّ هذا الكتاب قد تم تجهيزه بجهودٍ كبيرٍ لا يعلمه إلا الله تعالى  
 وعليه فإنَّ حقوق النسخ والطباعة والتسويق له لا تعود إلا للمؤلف ،،،

المؤلف

احرص دائمًا على توفير عوامل التركيز عند الدراسة؛ فالأصل في الدراسة التحصيل الكامل لكل ما تدرس لا مجرد الدراسة فقط.

## الباب الأول

### مقدمة للحاسوب والحوسبة

(Introduction To Computer and Computing)

يتوقع من الدارس في نهاية هذا الباب أن:

- يَتَعَرَّفُ على منطق عمل الحاسوب.
- يُعْدَدُ مكونات الحاسوب من حيث وحدات العمل.
- يَتَعَرَّفُ على طبيعة اللغة التي يفهمها الحاسوب.
- يستخدم أنظمة العد.
- يُحدِّدُ علاقة انظمة العد بفهم الحاسوب لاحتياجات الإنسان.
- يُعْدَدُ أسباب حاجة الإنسان للحوسبة.
- يُقدِّرُ أهمية الحوسبة في حياة الإنسان.

## الباب الأول: مقدمة للحاسوب والحوسبة

### 1.1 مقدمة

ما نعيشه اليوم من تطورٍ كبير في قطاع تكنولوجيا المعلومات والاتصالات لم يولد مباشرةً، وإنما نتيجةً لـتعدد المحاولات لربطِ الإمكانيات المادية للحاسوب (*Hardware*) بـمفاهيم البرمجيات (*Software*). فالعلاقة بينهما تتغير وثيقةً ولا يمكن تجاهل أيٍ منها. لهذا سنقوم في الفصل 1.2 بشرح موجز لمفهوم الحاسوب بنظرةٍ برمجيةٍ ثم ننتقل لتوضيح المكونات المادية والبرمجية للحاسوب في الفصل 1.3 من هذا الباب.

في الفصل 1.4 سنقوم بعرضٍ موجزٍ لأنظمة العد وعلاقتها بعمل الحاسوب وتأثيرها على التفاعل بين الإنسان والحاسوب وهو مفهومٌ يُساهم في عملية الحوسبة التي سنتناقشها في الفصل 1.5، بينما نختَم هذا الباب بالتدريبات والتمارين.

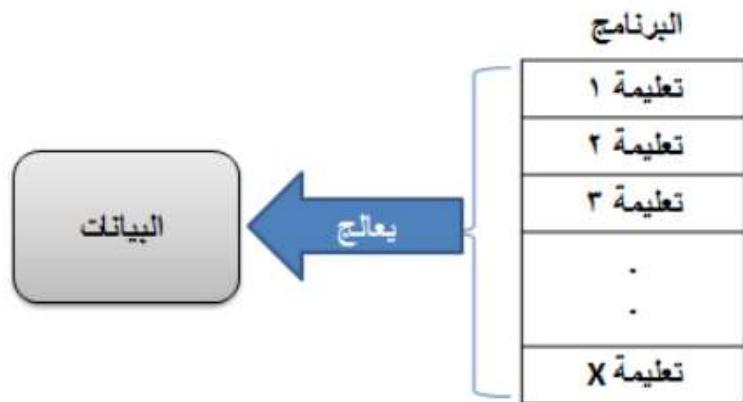
### 1.2 ما هو الحاسوب؟

توالت التعريفات للحاسوب على أنه آلة يمكنها إنجاز العمليات الحسابية والمنطقية بسرعةٍ تفوق سرعة الإنسان كثيراً من خلال استقبال البيانات ومعالجتها بطريقةٍ ما وإخراج النتائج بأشكالٍ مختلفة. ويكون الحاسوب عموماً من مكوناتٍ ماديةٍ يتم استخدامها أو إدارتها أو الاستفادة منها بـواسطة المكونات البرمجية. ومثالٌ لهذا تطبيق الآلة الحاسبة في جهاز الحاسوب حيث يتطلب من التطبيق إجراء عمليات حسابيةٍ مختلفةٍ من حيث التعقيد أو البساطة، فيستفيد وقتها التطبيق من المعالج والذاكرة المؤقتة ولوحة المفاتيح والفأرة والشاشة لإنجاز المهمة، وهذا يوضح أنَّ أي مهمةٌ تحتاج إلى إنجازها في الحاسوب سنكون بحاجةٍ إلى استخدام البرمجيات جنباً إلى جنب مع المكونات المادية.

فالحاسوب يقوم بمعالجة البيانات بناءً على مجموعةٍ من تعليماتٍ برمجيةٍ (*Instructions*)، وهي شكلٌ في مجموعةٍ ما يُعرف بالبرنامج أو التطبيق (*Computer program*) كما يظهر بالشكل 1.1، والذي يقوم بكتابته وتطويره شخصٌ يُعرف باسم مبرمج (*Programmer*) أو مطور نظم (*System Development*). هذه التعليمات سنتعلم كيف نكتبها في هذا الكتاب بلغةٍ جافا، وهي اللغة التي سنتعامل معها في الأبواب القادمة، وتتغير التعليمات نسبياً عند كتابتها بلغاتٍ أخرى.

**التطبيق:** هو عبارة عن مجموعةٍ من التعليمات أو الأوامر التي يتم كتابتها من قبل المبرمج، وهي توجه وتحكم في عملية استقبال ومعالجة وإخراج وتخزين البيانات في الحاسوب.





شكل 1.1: مفهوم المعالجة في الحاسوب

**المبرمج:** هو الشخص الذي يقوم بكتابة التعليمات اللازمة لعمل الحاسوب ويكتبها بطريقة تتوافق مع فهم الحاسوب وفي الوقت ذاته بما يضمن خروج نتائج صحيحة.



### 1.3 تنظيم الحاسوب و مكوناته

تنظيم الحاسوب أو مكوناته يمكن ذكرها بناءً على وجهات مختلفة سواء مادية أو برمجية؛ في هذا الكتاب سنفهم بتفصيل الأمر من جهة التنظيم الظاهري للحاسوب والذي ينقسم فيه الحاسوب إلى خمسة أجزاء هي كما يلي:

**أ- وحدة الإدخال:** تعتبر هذه الوحدة مسؤولة عن استقبال البيانات في الحاسوب، حيث يتم استقبال مدخلات (سواء كانت تعليمات أو أوامر أو بيانات تحتاج معالجة)، ويتم استقبالها من خلال أدوات الإدخال مثل (لوحة المفاتيح، الفأرة، الماسح الضوئي، لاقط الصوت وغيرها من أدوات الإدخال).

**ب- وحدة الإخراج:** هذه هي وحدة الإظهار للنتائج التي يصل إليها الحاسوب بعد إجراء عملية المعالجة على البيانات، ويتم إظهارها من خلال أدوات الإخراج المختلفة مثل الشاشة والطابعة والسماعات.

**ت- وحدة الذاكرة المؤقتة (Random Access Memory):** وفيها يتم تخزين البيانات التي يحتاجها الحاسوب وقت معالجة البيانات المطلوبة منه، وهي لا تحفظ بالبيانات طيلة الوقت، حيث أنها تفقد بياناتها بمجرد إغلاق التطبيق لأي سبب كان. وتُعرف هذه الوحدة بأسماء أخرى مثل (وحدة الذاكرة العشوائية، الذاكرة، الذاكرة الأساسية). هذه الذاكرة سنتعامل معها دائمًا خلال أبواب هذا الكتاب دون الإشارة الصريحة إليها عندما نتعامل مع ما يعرف بتعريف متغيرات وإنشاء كائنات وغير ذلك، إلا أننا

سنعرض لها بوضوح في الفصل 1.4 خلال الحديث عن أنظمة العد ثم كيف يتم تخزين القيم في الذاكرة لنشاهد كيفية تقسيمها واستغلالها.

**ث - وحدة المعالجة المركزية (CPU)**: هي عقل الحاسوب الذي يتم من خلاله إدارة كافة العمليات المطلوب معالجتها من خلال الحاسوب بما في ذلك العمليات الحسابية أو المنطقية كما يظهر ذلك في شكل 1.2، وتقسام هذه الوحدة إلى قسمين هما:

- **وحدة الحساب والمنطق (ALU)**: وتقوم بتنفيذ العمليات

الحسابية الرئيسية على البيانات المخزنة داخل الذاكرة الرئيسية متبعةً في ذلك الطريقة والمنهج الموضح في البرنامج الذي تم إدخاله إلى الحاسوب. كما تتفذ العمليات المنطقية وعمليات المقارنة التي يحتاج الحاسوب إلى تنفيذها خلال معالجة البيانات.

- **وحدة التحكم والسيطرة (CU)**: وتقوم هذه الوحدة بتنظيم سير نقل

التعليمات بين مختلف وحدات الحاسوب.



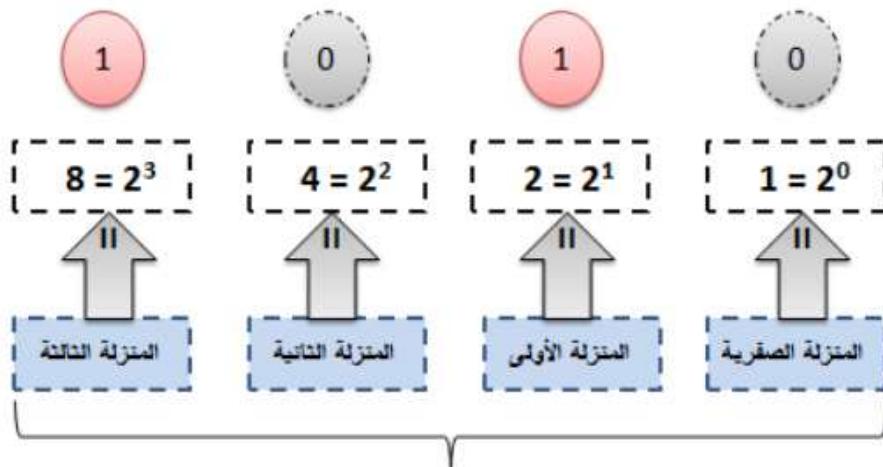
شكل 1.2: مبدأ عمل الحاسوب

**ج - الذاكرة الثانوية**: وهي الذاكرة الدائمة التي يتم فيها تخزين البيانات من المستخدم لفترات طويلة ويحتفظ بها الحاسوب بالرغم من إطفاء الجهاز لأي سبب كان.

#### 1.4 أنظمة العد

عرفت البشرية منذ بدايتها مجموعة من أنظمة العد مثل (النظام الأحادي Unitary system)، النظام العشري Hexa decimal system، النظام الثنائي Binary system، والنظام السادس عشر Decimal system) بالإضافة إلى العديد من الأنظمة الأخرى، والتي سنناقش أهمها في هذا الفصل.

**1.4.1 النظام الثنائي:** الأرقام في هذا النظام يتم التعبير عنها بسلسلة من الأرقام مكونة فقط من (0،1)، ومثال ذلك (1010) وهو الرقم الثنائي المكافئ للعدد 10، ويتم التعبير عن الأرقام بهذه السلسلة من خلال استخدام أقل عدد ممكن من الخانات التي أساسها 2 الأسس لها متسلسل 0,1,2,3,... ولتمثيل الرقم 10 مثلاً بالنظام الثنائي تابع معك الشكل التوضيحي 1.3.



نرى أن العدد  $10 = 8 + 2$  وبالتالي نختار الخفات التي تعطينا هذه القيم ونعرض مكانتها بالرقم 1 بينما الأخرى نعرض عنها بالرقم 0

شكل 1.3: تمثيل الأرقام في النظام الثنائي

وكما تتابع معك في الشكل 1.3 فإننا نضع الرتبة في النظام الثنائي متوافقة مع ترتيب الخانة بداية من الصفر وبزيادة متسلسلة بمقدار 1، وهذه الرتبة تكون هي الأسس للأساس الذي مقداره 2 في النظام الثنائي.

بعد ذلك نقوم باختيار أقل عدد من الأرقام الثنائية ( $2^{الأس}$ ) التي تعطينا في مجموعها الرقم الذي نريده مع استبدالهم بالرقم 1، بينما الأرقام التي لم نقم باختيارها نستبدلها بالرقم 0.

**مثال توضيحي 1.1:** قم بتمثيل الرقم 15 بالنظام الثنائي.

الحل :

بداية يظهر أن الرقم صغير وبالتالي نكتب سلسلة الأرقام الثنائية الخامسة الأولى لنرى الناتج:

$(2^4 + 2^3 + 2^2 + 2^1 + 2^0) = (16 + 8 + 4 + 2 + 1)$  كما تشاهد فإن الرقم 15 نحصل عليه من جمع الأرقام 1، 2، 4، 8 وهي متتالية و وبالتالي نضع مكانها الرقم 1 كما نرى (1111)

**الرقم الثنائي:** هو عبارة عن الرقم الناتج من الأساس 2 مرفوعنا للرتبة ولها اسم مختصر هو **البت (Bit)**



تتضح أهمية النظام الثنائي في أن الحاسوب يعتمد في عمله على هذا النظام.



**مثال توضيحي 1.2:** قم بتمثيل الرقم 129 بالنظام الثنائي.

**الحل :**

بداية يظهر أن الرقم كبير نسبياً و بالتالي نكتب سلسلة الأرقام الثنائية العشرة الأولى لنرى الناتج:

$$(32 + 16 + 8 + 4 + 2 + 1) = (2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) + 128 + 64 \text{ كما تشاهد فإن الرقم 129 نحصل عليه من جمع الأرقام 1 و 128 و بالتالي نضع مكانهم الرقم 1 بينما الأرقام الأخرى نضع مكانها 0 كما ترى (1000000001).$$

ولكن كيف لي أن أعرف عدد الخانات التي احتاجها لتمثيل رقم ما؟

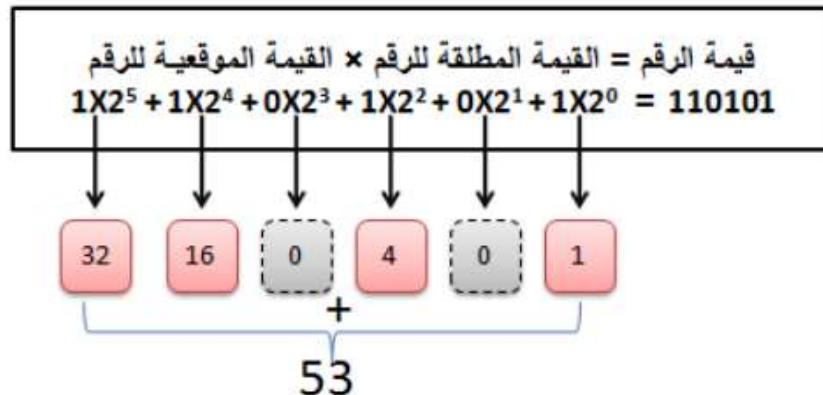
هذا السؤال يمكن الإجابة عليه من خلال معرفة سلسلة الأرقام المميزة المتسلسلة وهي (1 ، 2 ، 4 ، 8 ، 16 ، 32 ، 64 ، 128 ، 256 ، 512 ، 1024 ، 2048 ....)، وهذه السلسلة هي نواتج مضاعفات العدد 2، ومن خلالها

تعرف الرقم المراد تمثيله بالنظام الثنائي هل هو كبير أم صغير؟

ولا بد في هذا النظام أن نوضح نقطة واحدة وهي أن الحزمة الواحدة من الخانات العشرية يساوي 8 وهو ما يُعرف بالبايت (Byte) وعند تمثيل أي عدد بالنظام الثنائي لا يمكن لي استخدام إلا وحدات كاملة لا وحدات عشرية منه فمثلاً الرقم 127 يحتاج إلى تمثيله ثمانية أرقام ثنائية وهو ما يقابل بايت واحد.

يتبقى أن نعرف كيفية معرفة قيمة العدد الثنائي، فمثلاً الرقم الثنائي 110101 ما قيمته؟

للإجابة على هذا السؤال انظر للشكل التوضيحي 1.4 والذي يبين أننا نضرب كل خانة من الرقم الثنائي بقيمتها الموقعة ( $2^{(\text{الرتبة})}$ ) ونجمع حاصل الجميع.



شكل 1.4: كيفية حساب قيمة رقم من النظام الثنائي

**1.4.2 النظام العشري:** يعتمد هذا النظام على عشرة أرقام هي (0 ... 9) حيث يمكن من خلالها تمثيل أي عدد على أنه حاصل ضرب كل رقم منه في الأساس 10 مرفوعاً إلى رتبته (موقعه) التي تبدأ من 0 وتزداد بزيادة منتظمة بواحد.

**مثال توضيحي 1.3:** قم بتمثيل الرقم 273 بالنظام العشري.

كل خانة من خانات هذا الرقم يساوي قيمته المطلقة × قيمة موقعه من حيث (أحاد، عشرات، ...) وعلى ذلك :

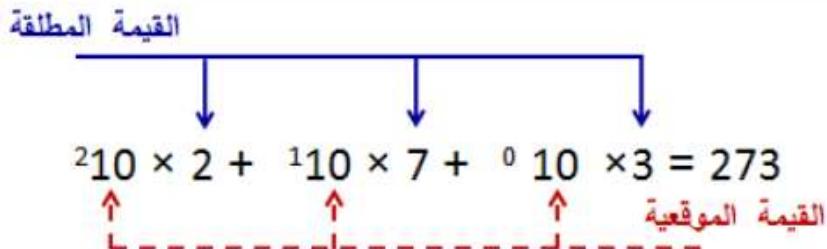
الخانة الأولى بها الرقم 3 وهي خانة أحداد قيمتها  $1 \times 10^0$

الخانة الثانية بها الرقم 7 وهي خانة عشرات قيمتها  $1 \times 10^1$

الخانة الثالثة بها الرقم 2 وهي خانة مئات قيمتها  $1 \times 10^2$

إذن الرقم  $273 = 10^2 \times 2 + 10^1 \times 7 + 10^0 \times 3$

$$200 + 70 + 3 = 273$$



شكل 1.5: توضيح مفهوم القيمة الموقعة والقيمة المطلقة

### ١.٤.٣ النظام السادس عشر:

هذا النظام يتكون من 16 رمزاً وهي الأرقام من (0 إلى 9) ثم الأحرف الإنجليزية من (A إلى F) حيث يتم من خلالها تمثيل كافة الأرقام، فالرقم 10 هو A وهكذا، و ما تحتاج معرفته أن عناوين البيانات التي تُخزن داخل ذاكرة الحاسوب يتم التعبير عنها بهذا النظام، ولكننا كمبرمجين أو مستخدمين لا نستخدم هذا النظام مباشرة، ولكننا نعرف طريقة عمله لنفهم منطق عمل الحاسوب.

### ١.٥ مفهوم الحوسبة وال الحاجة إليها؟

عندما نعمل على إنجاز أي مهمة من المهام التي تقوم بها في حياتنا من خلال الحاسوب بطريقة منهجية وواضحة الخطوات والترتيب فهذا ما يعرف باسم الحوسبة، فالحوسبة هي محاولة إنجاز المهام التي يقوم بها الإنسان من خلال الحاسوب، ومثال ذلك أن يقوم الحاسوب بطباعة جدول الضرب للأعداد من (1 إلى 100) للعدد 12 ، فهذه العملية يُعتبر الإنسان قادرًا على القيام بها ولكنها تأخذ من وقتها زمن أطول وقد تكون النتائج غير دقيقة خاصة إذا ما حاول الإنسان الإسراع أو كان يشعر بالملل، في المقابل فإن الحاسوب يقوم بهذه المهمة في زمن قصير جداً والنتائج تكون متوافقة تماماً مع الطريقة التي يضعها الإنسان (المبرمج) للحاسوب.

**الhosبة :** هي مفهوم يطلق على كافة العمليات التي يقوم بها الحاسوب بدلاً من الإنسان وكان الإنسان قادرًا على فعلها بمجهود و زمن أكبر ودقة أقل.  
ويمكن تعريف **الhosبة** بأنها استخدام الحاسوب لإنجاز المهام بارشاد و إشراف من الإنسان.



### ومن مميزات الحوسبة:

- أ- السرعة الفائقة في أداء المهام حيث تزيد سرعة الحاسوب عن سرعة الإنسان بشكل ملموس.
  - ب- الدقة المتناهية في النتائج، فالحاسوب يُخرج نتائج متوافقة تماماً مع التعليمات التي يُدخلها الإنسان للحاسوب، وإن ظهر خطأ في النتائج فيكون ذلك من خلال خطأ مسبق في التعليمات أو البيانات التي أدخلها الإنسان للحاسوب.
  - ت- ليس من عيوبها مزاجية الإنسان وشعوره بالملل أو الإرهاق مما يجعل معالجة البيانات لها دقة واحدة.
- وبناءً على ما تقدم تظهر لنا بشكل واضح مدى الحاجة للhosبة فهذا المفهوم يساعد على سرعة وزيادة الإنتاج في مجالات الحياة المختلفة.

**1.6 تدريبات عامة**

1. استخدم الكلمات المناسبة لتعبئة الفراغات في الجمل التالية:

- أ- يتكون الحاسوب من خمسة مكونات ظاهرية هي \_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_.
- ب- \_\_\_\_\_ هي مفهوم يطلق على استخدام الحاسوب في تنفيذ المهام بدلاً من الإنسان.
- ت- الشخص الذي يقوم بكتابية التعليمات للحاسوب بطريقة تتوافق مع فهم الحاسوب يعرف ب\_\_\_\_\_.
- ث- من أدوات الإدخال للحاسوب \_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_.
- ج- الحاسوب يقوم بمعالجة البيانات من خلال مجموعة \_\_\_\_\_ برمجية تكون في مجموعها تطبيق الحاسوب.

2. حدد صحة أو خطأ كل من العبارات التالية مع التعليق:

- ( ) أ- يتاثر الحاسوب بالعوامل النفسية مما يجعله يُخرج نتائج خاطئة في أوقات مختلفة.
- ( ) ب- الذاكرة الثانوية يمكن الاستغناء عنها في الحاسوب بعكس الذاكرة الرئيسية.
- ( ) ت- يمكن الحاسوب من إنجاز المهام بسرعة تزيد عن سرعة الإنسان ملايين المرات.
- ( ) ث- يمكن تمثيل الرقم 18 بالنظام الثنائي بالرقم 11001.
- ( ) ج- الأساس في النظام الثنائي والنظام العشري هو الرقم 10.

3. أذكر باختصار، مميزات الحوسبة.

4. قم بتمثيل الأعداد التالية بالنظام الثنائي:

- أ- 112
- ب- 245
- ت- 897
- ث- 231
- ج- 1287

5. قم بتمثيل الأعداد التالية بالنظام العشري:

أ- 767

ب- 875

ت- 1207

ث- 986

ج- 54

### 1.7 تمارين ذاتية الحل

1. عرّف الحاسوب مع ذكر الحاجة إليه.

2. أذكر مميزات الحوسبة مع مقارنتها بأداء الإنسان.

3. قارن بين أنظمة العد الثلاثة التي تم التعرض لها في هذا الباب مع ضرب الأمثلة.

4. تعتبر وحدة المعالجة المركزية بمثابة المدير والعقل للحاسوب، في ضوء هذه العبارة ناقش دور الوحدة مستخدماً الرسم لتوضيح العلاقة بين أجزائها.

5. عبّر عن الأعداد التالية بالنظام الثنائي:

أ- 786

ب- 432

ت- 21

ث- 543

ج- 32

ح- 12

خ- 231

6. مثل الأعداد التالية بالنظام العشري بالطريقة الموضحة في الشكل 1.5

أ- 865

ب- 1092

ت- 223

ث- 345

ج- 6854

ح- 124

خ- 11

**اجابات التدريبات العامة :**

1. (أ) وحدة الإدخال ، وحدة الإخراج، وحدة الذاكرة المؤقتة، وحدة الذاكرة الأساسية، وحدة المعالجة المركزية  
 (ب) الحوسبة (ت) المبرمج (ث) لوحة المفاتيح ، الفأرة (ج) تعليمات

.2

- أ- (خطأ) لا يتأثر الحاسوب بالعوامل النفسية ونتائجها دائمًا صحيحة، والخطأ إن حدث يكون بسبب المبرمج الذي قد يعطي تعليمات خطأ أو بطريقة خطأ.  
 ب- (خطأ) لا يمكن الاستغناء عن أي من أنواع الذاكرة الثانوية أو الرئيسية، فكل منها له وظيفة.  
 ت- (صحيحة).  
 ث- (خطأ) تمثل الرقم 18 بالنظام الثنائي يتم بالرقم 10010.  
 ج- (خطأ) الأساس في النظام الثنائي هو 2، أما في العشري هو الرقم 10.

**3. مميزات الحوسبة هي :**

- أ- السرعة الفائقة.  
 ب- الدقة المتناهية.  
 ت- عدم اعتمادها على العوامل النفسية.

.4

- .أ. تمثل العدد 112 بالنظام الثنائي = 1110000

ii. تمثيل العدد 245 بالنظام الثنائي  $= 11110101$

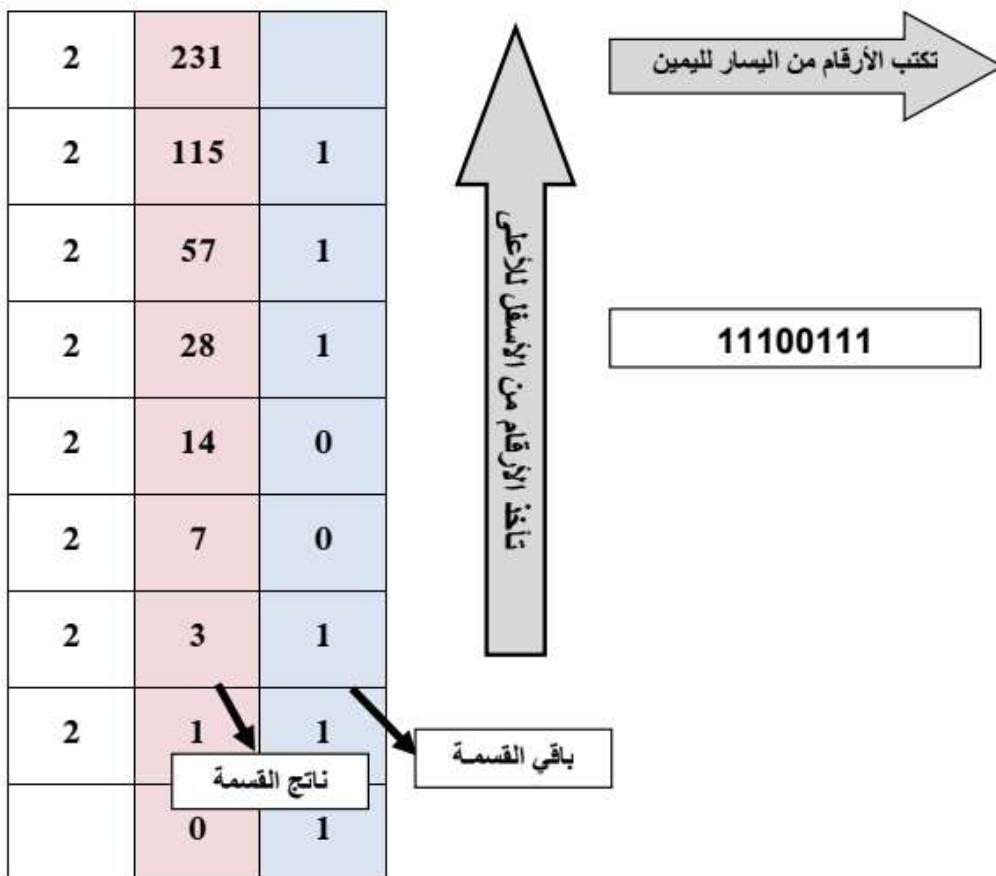
iii. لتمثيل الرقم السالب يتم تخصيص الخانة الأخيرة من البايت للإشارة وتوضع 1 في حال السالب، و0 في حال الموجب، وفي حال الرقم 17 يمكن تمثيله من خلال استفاد بait واحد فقط، وبالتالي يتم وضع القيمة 1 في الخانة الثامنة، كما يلي:

$10010001$

iv. هذا العدد يحتاج تمثيله إلى عدد 2 بايت (أي يحتاج أكثر من 8 بت)، وبالتالي تمثيله  $10000001$

$111$

v. هذه المسألة سنقوم بحلها بطريقة أخرى لعلها تكون أكثر سهولة للبعض، وهي أن يتم تقسيم العدد على الرقم 2، وكذلك نواتجه حتى نصل إلى الرقم 0 وفي كل مرة نحتفظ بباقي القسمة الذي لا يكون إلا 0 أو 1، وفي النهاية يتمأخذ بباقي القسمة من أسفل إلى أعلى وكتابتهم من اليسار إلى اليمين.



.vi. العدد 1287 بالنظام الثنائي يحتاج تمثيله إلى عدد 2 بait (كيف عرفت هذا؟)

تابع معى سلسلة الأرقام التالية ستجد أن مجموع أول سبعة أرقام لا يصل بنا إلى الرقم 1287، بينما مع

إضافة الأعداد الأربع التالية نحصل على العدد وبالتالي فنحن بحاجة إلى 2 بait.

والآن ما هي الأرقام التي مجموعها يعطينا العدد 1287، ستجد أنها الأرقام المحددة أسفل منها خط هي

1024 512 256 128 64 32 16 8 4 2 1

ضع تحتها بالرقم 1 وتحت الباقي الأخرى بالرقم 0 سينتج معك الرقم

.10100000111

.5

.i. 767: حل هذه المسألة يتم تقسيم الرقم بقانون ( $\text{الرقم} = \text{القيمة المطلقة} * \text{القيمة الموقعة}$ ) وبالطريقة ذاتها المسائل التالية.

$$10^2 * 7 + 10^1 * 6 + 10^0 * 7 = 767$$

875 .ii

$$10^2 * 8 + 10^1 * 7 + 10^0 * 5 = 875$$

1207 .iii

$$10^3 * 1 + 10^2 * 2 + 10^1 * 0 + 10^0 * 7 = 1207$$

986 .iv

$$10^2 * 9 + 10^1 * 8 + 10^0 * 6 = 986$$

54 .v

$$10^1 * 5 + 10^0 * 4 = 54$$

انتهى الباب

التفكير في حل المشاكل هي المرحلة الأهم لحلها. قال روبرت فروست (كم مرّة وقعت تفاحة في يد نويتن؟!)  
الفارق هو التفكير !!

## الباب الثاني

### الخوارزميات وطرق حل المشاكل (Algorithms and Solving Problems)

يتوقع من الدارس في نهاية هذا الباب أن:

- يدرك أهمية التفكير في علم البرمجة.
- يكتسب مهارة التفكير في حل المشاكل البرمجية.
- يُجيد قراءة المشاكل بعمق.
- يُجيد فهم المشاكل وتحليلها.
- يُتقن استخدام الشيفرة المزيفة لتمثيل فكرة حل المشاكل.
- يُتقن استخدام مخطط سير العمليات لتمثيل فكرة حل المشاكل.
- يتعامل مع مختلف أنواع المشاكل البرمجية.

يتوفر لهذا الباب شرائح العرض (PowerPoint) وكذلك ملفات مرئية من إعداد مؤلف الكتاب؛ يمكنك الحصول عليها من خلال الموقع الأكاديمي للمؤلف [mfarra.cst.ps](http://mfarra.cst.ps)



## الباب الثاني: الخوارزميات وطرق حل المشاكل

### 2.1 مقدمة

في حياتنا العامة نواجه الكثير من المواقف التي تحتاج فيها إلى التفكير ثم اتخاذ القرار المناسب، وبقدر ما يتوفّر لنا بيانات صحيحة ويكون تفكيرنا صحيحاً يكون القرار الذي نحصل عليه صحيحاً، لذلك تم تخصيص هذا الباب للحديث عن حل المشاكل البرمجية وكيفية التعامل معها حيث إنها مرحلة بالغة الأهمية تسبق تنفيذ الحلول، ففي هذا الباب سنقوم بعرض الخطوات العامة لحل المشكلة في فصل 2.2 حيث نصل من خلاله لفهم كامل لحل المشكلة، وسيتبقى لنا كيفية تمثيل هذا الحل بصورة واضحة من خلال طرق متعارف عليها والتي سنناقشها في الفصلين 2.3 و 2.4 قبل أن نستعرض مجموعة من التمارين المختلفة.

### 2.2 خطوات حل المشاكل

كيفية حل المشاكل التي يواجهها الإنسان كانت ولا زالت محل بحث العديد من الباحثين والمهتمين في هذا المجال، وقد عمل الكثير من الباحثين على تحديد الخطوات العامة التي يمكننا الاعتماد عليها في حل المشاكل، ومن أبرزها ما ذهب إليه العالم الرياضي رين ديكارت، ومما ذهب إليه:

أ- لا شيء يعتبر حقيقة ثابتة قبل أن يثبت بالتجارب والمشاهدات.

ب- كل مشكلة يمكن تبسيطها ليسهل حلها وهي موازية لقاعدة (فرق تسد) المعروفة.

ت- كل مشكلة يبدأ حلها من خلال الأجزاء السهلة ثم التدرج للأجزاء الصعبة شيئاً فشيئاً.

هذه القواعد يتم اعتمادها أيضاً لحل المشاكل من خلال الحاسوب، بالإضافة إلى ضرورة تنظيم وترتيب الخطوات لما لها من أهمية كبيرة على نوع النتائج الصادرة. ولهذا فإن حل المشاكل باستخدام الحاسوب تعتمد على خطوات سنناقشها في النقاط التالية.

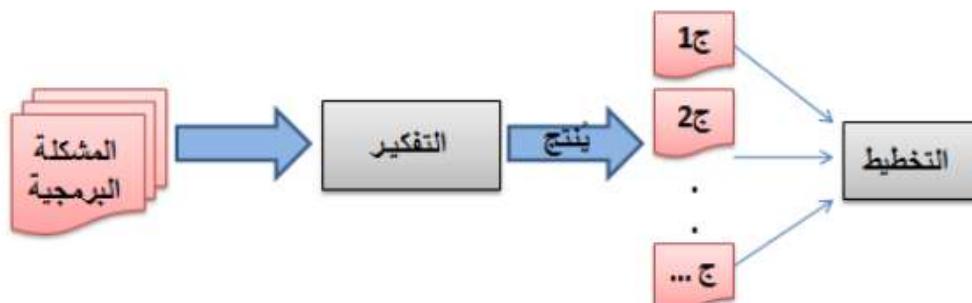
#### 2.2.1 التفكير الم導ي لفهم المشكلة ومن ثم تحليلها:

لعل النظرة الأولى للمشاكل لا تكفي للتعرف على حقيقة سهولتها أو صعوبتها، وقد يعتبر البعض أن لها لا يمكن من النظرة الأولى، وهذا لم يحدث إلا بسبب غياب التفكير الهادئ والمتعمق في المشكلة ولهذا فإننا ننصح حل المشاكل بالتفكير الجيد فيها. والتفكير الجيد له خطوات عامة ننصح باتباعها وهي كما يلي:

أ- تحديد المعطيات في المشكلة، فمثلاً عدد الموظفين، نسبة الضرائب، المرتب الأساسي لكل فئة أو غير ذلك.

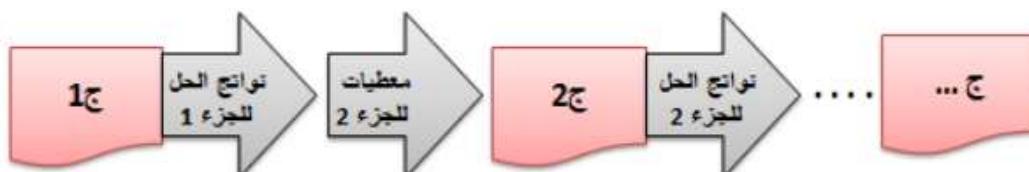
ب- تحديد المطلوب مثل حساب مرتب الموظف النهائي، حساب التقدير التراكمي للطالب أو غير ذلك.

وكما يتضح في الشكل 2.1 فإن تفكيرنا في المشكلة ينتج عنه تقسيمها إلى أجزاء صغيرة بسيطة ولكنها متكاملة، فنحتاج بعد ذلك إلى وضع خطة واضحة في مجموعها تؤدي إلى الحل.

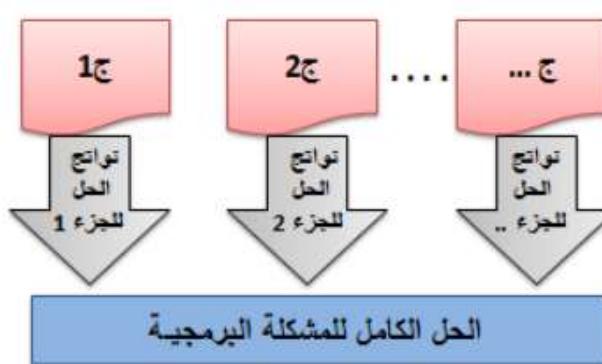


شكل 2.1: دور التفكير في الوصول لحل المشاكل

خلال تقسيم المشكلة إلى أجزاء متتالية أو متوازية - أي نقسم المشكلة الكبيرة إلى مجموعة مشاكل صغيرة يمكن حلها إما على التوالي أو على التوازي -. والمقصود بالأجزاء المتتالية أنَّ نواتج الحل الأول يستخدم في إيجاد الحل الثاني وهكذا كما في الشكل 2.2، أما المتوازية فيها يتم تقسيم المشكلة لمجموعة أجزاء، وكل جزء يتم حله منفرداً عن الآخر، وفي النهاية تجمع الحلول لنحصل على الحل النهائي كما في الشكل 2.3.



شكل 2.2: التقسيم المتماثل للمشكلة المحطة



شكل 2.3: التقسيم المتوازي للمشكلة البرمجية

في المثال التوضيحي التالي، نستعرض كيفية التفكير الذي ينتج عنه تحليل المعطيات والمطلوب وكيف يمكننا تقسيم المشكلة إلى مشاكل متالية للوصول إلى الحل، وهذا يعني أن التفكير له أسلوب واحد أو طريقة واحدة، والهدف من هذا المثال توضيح مفهوم التجزئة فقط مع التأكيد أن كل مشكلة لها تفكير خاص يعتمد على طبيعتها.

**التفكير في حل المشاكل** مهما كانت درجة تعقيدها من أهم الأمور ويظل فقط مسألة توثيق التفكير كاملاً هل ضرورة أم لا هذا نتركه لدرجة تعقيد المشكلة وترابط مطالبها.



### مثال توضيحي 2.1:

أحد المؤسسات التعليمية في غزة يعمل بها 540 موظف، منهم 300 أكاديمي، 100 إداري، 100 رجل أمن و 40 موظف بنظام العقود.

المرتب الأساسي لهم جميعاً هو 300 دينار بينما يحصل الأكاديمي على زيادة قدرها 100% ويحصل الإداري على زيادة قدرها 70% بينما يحصل رجل الأمن على زيادة قدرها 50% ولا يحصل موظفي العقود على زيادات.

كذلك يتم مكافأة كل موظف بمبلغ قدره 100 دينار عند نجاحه في تقديم عمل غير نمطي للمؤسسة ويقدر عدد النجاحات شهرياً بالعدد (س).

إذا كنت المبرمج المختص في هذه المؤسسة وأراد مديرها تطوير تقرير شهري يخرج له الحاسوب لحجم الزيادات التي يدفعها للموظفين بالإضافة لإجمالي المبالغ التي تدفعها المؤسسة شهرياً، اكتب الخطوات اللازمة لإيجاد الحل.

الحل :

**أولاً: إخراج المعطيات وهي /**

1. عدد الموظفين لكل فئة .
2. مقدار الراتب الأساسي لكل فئة.
3. مقدار المكافأة التي تدفع لكل عمل يستحق المكافأة.
4. عدد النجاحات = س

**ثانياً: تحديد المطلوب وهو /**

1. مجموع الزيادات التي تدفعها المؤسسة للموظفين شهرياً.
2. مجموع المبالغ التي تدفعها المؤسسة للموظفين شهرياً.

**ثالثاً: الخطة الممكنة للوصول للمطلوب من المعطيات /**

من خلال التفكير في المعطيات للمشكلة وكذلك بقراءة المشكلة قراءة متأنية وهادئة سنجد، أن هذه المشكلة الكبيرة (المترفرعة) لا بد من تقسيمها إلى قسمين متوازيين هما:

1. حساب مجموع الزيادات لكافة الموظفين حيث نحتاج هنا لحساب زيادات كل فئة على حدا ثم نجمعهم سوياً مع الزيادة الخاصة بالمكافأة وهذا نقسم متوازي أي أن حساب زيادات الأكاديميين لا يؤثر على حساب زيادات الإداريين وهكذا، ولكن التقسيم العام للمشكلة الفرعية يعتبر متالي بمعنى أننا لحساب التقرير النهائي لا بد لنا من حساب الزيادات الخاصة بالفئات المختلفة.
2. حساب المبالغ التي تدفعها المؤسسة للموظفين وهذا المطلب للحصول عليه لا بد من تجزئته إلى جزئين متوازيين هما حساب المرتبات الأصلية لكل فئة بالإضافة إلى مجموع الزيادات.

و بهذا التفكير سيصبح /

$$\text{الزيادة في مرتب الموظفين لأي فئة} = [ (300 \times \text{نسبة الزيادة}) + 300 ] \times \text{عدد موظفي الفئة}$$

$$\text{مجموع الزيادات} = (\text{زيادات الأكاديميين} + \text{زيادات الإداريين} + \text{زيادات الأمن}) + س \times 100$$

$$\text{المبلغ الإجمالي} = \text{مجموع الزيادات} + (\text{مرتبات الأكاديميين} + \text{مرتبات الإداريين} + \text{مرتبات الأمن} + \text{مرتبات العقود}).$$

بالتأكيد قد تشعر أن المسألة لا تحتاج إلى كل هذه الكتابة والاستنتاجات وأنت محق!! ولكن هذا الأمر ستجده ضروريًا عندما تواجه مشاكل برمجية كبيرة ومعقدة، وبالتالي نحن نحتاج إلى مرحلة التفكير.

**التفكير** هو دراسة المشكلة بهدوء وتركيز لوضع اليد على معطياتها ومطالبتها ثم الاتجاه في طرق التخطيط الواضح لطريقة حلها.



## 2.2.2 التخطيط لحل المشكلة:

سنصل إلى هذه المرحلة بعد قضاء وقت في مرحلة التفكير حتى نصل إلى تصور واضح لحل المشكلة يدوياً أو ذهنياً ولكن كيف للحاسوب أن يتدخل الآن وكيف لنا أن نتعاون مع المبرمج لتنفيذ الحل وللتصبح تطبيقاً برمجياً ناجحاً، هذا هو السؤال الذي سنجيب عنه خلال الفصل الحالي وسيزيد وضوحاً في الفصلين 2.3 و 2.4.

مراحل التفكير والتحليل وكذلك التخطيط من مهام محللي النظم (*Systems Analysts*) حيث يقوم أصحاب هذه المهنة بقراءة المشكلة البرمجية وتحليل عناصرها للبحث عن حلها، وبعد أن يصل إلى حل يقوم بتمثيل الحل الذي وصل إليه بطريقة واضحة وسهلة الفهم للآخرين ثم يقدمها للمبرمج الذي يعمل على تحويلها من الورق والكلمات الأدبية إلى لغة يفهمها الحاسوب فتصبح تطبيقاً برمجياً ناجحاً كما أسلفنا. فالخطيط هو مرحلة تتبع مرحلتي التفكير ثم التحليل ويتجلّى الهدف منه في تدوين الحل في خطوات متسللة ومتsequالية يعبر عنها باللغة الأدبية ويحكمها منطق الرياضيات، وهذا ما يعرف بالخوارزميات، فهي تمثل حل أي مشكلة برمجية من خلال تنفيذ مجموعة من الخطوات بترتيب معين.

ومصطلح الخوارزميات تم اشتقاقه من اسم العالم المسلم "محمد بن موسى الخوارزمي" والذي استخدمها في تدوين علم الرياضيات على أنها مجموعة الخطوات (*التعليمات*) المرتبة لتنفيذ عمليات حسابية أو منطقية أو غيرها بشكل تابعي متسلسل ومنظم.

هذه الخوارزميات يتم التعبير عنها بطريقتين هما:

1. الشيفرة الوصفية (*Pseudocode*).

2. مخطط سير العمليات (*Flowchart diagram*).

و سنعرض لهما في الفصول القادمة بالشرح والتفسير الواضح.

**الخوارزميات هي الإجراء اللازم لحل مشكلة برمجية من خلال خطوات محددة للتنفيذ.**



٢. ترتيب معين يتم من خلاله تنفيذ الخطوات.

على أن تكتب هذه الخوارزميات بلغة الأدميين دون أن تستخدم كلمات لها معانٍ خاصة بلغات برمجية تختلف عن الأخرى.

وبناءً على الحاجة لاستخدام الخوارزميات من مجموعة من الفوائد مثل:

- (1) توثيق التفكير من أجل حل المشاكل البرمجية.
  - (2) تحديد الوقت الذي يحتاجه الحاسوب لحل المشكلة.
  - (3) تحديد المساحة التخزينية التي قد تحتاجها الحاسوب لحل المشكلة.
  - (4) المفاضلة بين الطرق والخوارزميات من حيث السرعة والمساحة التخزينية.
  - (5) تساهم في سرعة اكتشاف أخطاء التفكير قبل البدء في مرحلة التطبيق العملي.
  - (6) تساهم في إيصال فكرة حل المسألة بعيداً عن لغات البرمجة المتعددة حتى لا يؤثر ذلك على المبرمج في ضرورة استخدام لغة برمجة محددة.
  - (7) تعطينا الفرصة لحل المشاكل بطرق مختلفة ومتعددة وأخيراً.
  - (8) تساهم في سهولة فهم الأفكار و طرق الحل للمشاكل المختلفة.

وتكون الخوارزميات من ثلاثة أنماط من الترتيب، تم استنباطها من واقع أي عملية في حياتنا، ويمكن استخدامها فرادى أو مجتمعة وهي على النحو التالي:

- نطاق التسلسل:** هو نمط مناسب في حال كانت الخطوات متتابعة دون الحاجة إلى تكرار شيء منها أو الاختيار بين خطوة وأخرى، ومثال عليها أن يطلب منك معلمك أن تجمع الرقم 100 مع رقم سيعطيه لك زميلك في المحاضرة، فهنا أنت ستحصل على الرقم من زميلك ثم تجمعه مع الرقم 100 ثم تمرر المجموع لمعلمك وهي خطوات متتالية لا اختيار ولا تكرار فيها.

**نطاق الاختيار:** هو نمط مناسب في حال كان الانتقال من خطوة إلى خطوة أخرى أثناء الحل معتمد على قرار ما أو اختيار محدد بين خطوتين، ومثال عليها أن يطلب منك معلمك أن تسأل زميلك عن معدله التراكمي ثم تخبره أنه ناجح إذا كان معلمه أكبر من أو يساوي 60 وأنه راسب إن كان دون

ذلك، هنا خطوة تحديد النتيجة (ناجح أو راسب) تعتبر اختيار بين خطوتين ويعتمد ذلك على شرط معين.

- نمط الدوران: هو نمط مناسب للحل الذي يشتمل على مجموعة خطوات تحتاج إلى تكرارها أكثر من مرة حسب شرط معين أو بعد مرات محددة ومثال ذلك أن يطلب منك معلمك أن تجمع الرقم 100 مع رقم سيعطيه لك زميلك في المحاضرة على أن تقوم بهذه العملية مع كافة الطلاب بالمحاضرة، فهنا أنت ستكرر عملية استقبال رقم من زميلك وجمع هذا الرقم مع الرقم 100 بعدد الطلاب في الفصل وهذا ما يعرف بالدوران أو الدوران وكأنك تدور حول خطوات معينة وتتذكرة في كل مرة.

هذه الأنماط الثلاثة للخوارزميات قد تجتمع في خوارزمية واحدة أو قد يجتمع اثنان منهم ويغيب الثالث ولكن الأكيد أن النمط الثالث وهو نمط الدوران لا يأتي مفرداً بل لا بد أن يأتي معه النمط المتسلسلي أو نمط الاختيار كما تلاحظ ذلك في المثال الأخير؛ فتكرار العملية كان بالأساس لخطوات متسلسلة.

وبمجرد الانتهاء من تدوين خطوات حل المشكلة بأحد الطريقتين يتم الانتقال للمرحلة التالية وهي مرحلة الحوسبة أو تطوير التطبيق المحسوب والتي ينفذها المبرمج بعد أن يحصل على هذه الخوارزميات ليقوم بترجمتها بلغة مناسبة للحاسوب، المزيد عن هذه المرحلة سنناقشه فيما يلى.

### 2.2.3 تطوير تطبيق الحل واختباره:

مرحلة تطوير التطبيق الذي يحاكي الحل الذي توصلنا له في الخطوة السابقة لعلها تكون من أسهل المراحل في نظر الكثرين إذ أن المبرمج فيها سيعمل على ترجمة الخوارزميات التي حصل عليها من محل النظم للغة يمكن للحاسوب أن يفهمها من خلال وسيط ما، وهي المرحلة التي نستخدم فيها لغات برمجة مثل لغة جافا والتي سندرس أساسياتها ومفرداتها في هذا الكتاب.

**محل النظم:** هو الشخص الذي يقوم بتحليل عناصر المشكلة والتفكير في كيفية حوسبتها مع تدوين ذلك بوسائل مختلفة منها الخوارزميات.



بعد تفويض إنجاز التطبيق نصبح بصدور مرحلة مهمة وهي اختبار التطبيق لمعرفة هل بالفعل يعطي النتائج المطلوبة أم لا، فمثلاً إذا كنتَ نرغب في تطوير تطبيق لجمع الرقم س مع الرقم ص وبعد أن قمنا بتطوير التطبيق وتنفيذه نبدأ في تجربته من خلال إعطاءه أرقام ونرى هل يخرج عملية الجمع صحيحة كما هي قواعد الرياضيات أم لا ، فإذا كانت لا نعود مرة أخرى على خطوات الحل لنبحث أين الخلل هل كان في التفكير وفهم المشكلة أم في تحليل المسألة أم أن الخوارزميات كتبت بطريقة خاطئة أم أن المبرمج فشل في ترجمة الخوارزميات بصورة صحيحة.

مرحلة الاختبار لا بد أن يقوم بها المستخدم النهائي للنظام (*End user*) وهو الشخص الذي سيستخدم النظام بعد ذلك في عمله، ويُصف بأنه خبير في هذه العملية أي أنه يعرف كيف تتم خطواتها يدوياً بشكل دقيق، ثم بدون ملاحظاته المختلفة على النظام ليعيدها للمبرمج ومحل النظام لتصحيح الأخطاء حتى يصلوا سوياً لنظام برمجي بدقة مميزة، وهذا المفهوم قمت بتوضيحه لك من خلال الشكل التوضيحي 2.4 بالاتجاهين.

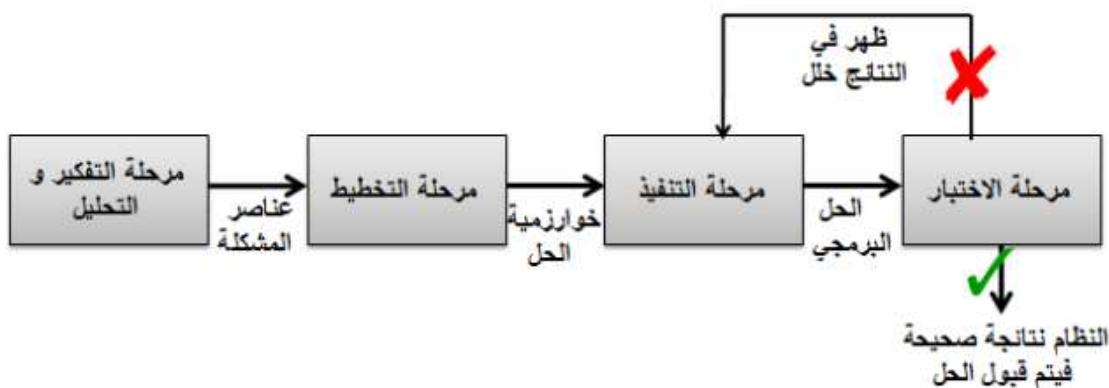
### تسليم الحل البرمجي تباعاً من محل النظم للمبرمج ثم للخبير



إعادة الحل البرمجي عند ظهور خطأ من الخبير للمبرمج و منه للمحل للبحث عن موضع الخطأ

شكل 2.4: العلاقة بين محل النظم والمبرمج والمستخدم الخبير

يبقى أن أخص لك مخرجات كل مرحلة من الأربع التي تحدثنا عنها للوصول إلى حل المشكلة برمجياً وذلك مكتفياً بالشكل التوضيحي 2.5.



شكل 2.5: مخرجات مراحل حل المشكلة

### 2.3 طريقة الشيفرة الوصفية (Pseudocode)

طريقة الشيفرة الوصفية هي إحدى الطرق المستخدمة للتعبير عن الحل الذي وصل له محلل النظم من خلال التفكير والتحليل للمشكلة، ويستخدمه ليوضح كافة عناصر حل المشكلة، بدايةً من المعطيات حتى الوصول للنتائج المطلوبة مع توضيح مفصل وواضح جداً لطريقة الوصول للنتائج.

ومن أبرز ما يميزها أنها:

- طريقة ذكية وواضحة لتمثيل حل المشاكل.
- تكتب بلغة الإنسان سواء كانت العربية، الإنجليزية، الفرنسية، .... .
- لا تعتبر لغة برمجة، بمعنى أنّ الحاسوب لا يفهمها لا مباشرة ولا من خلال وسيط مباشر.
- يمكن تحويلها بسهولة للغات البرمجة المختلفة مثل جافا ، سи شارب، وغيرها بواسطة المطورو.

ولأنها ليست لغة برمجة فهي غير خاضعة لقواعد كتابة صارمة وإنما هناك مجموعة قواعد ينصح بالانتهاء لها عند كتابة استخدام هذه الطريقة وهي:

- أن تكون الكلمات المستخدمة سهلة الفهم قدر الإمكان.
- أن يكون الأسلوب واضحاً بحيث يتمكن المبرمج من التعامل مع الخطوات دون حدوث سوء فهم يؤدي إلى بناء حل برمجي مختلف عن المراد.
- أن يتم كتابتها مع الأخذ بعين الاعتبار الأنماط سابقة الذكر، كما ستتابع في المثال التوضيحي 2.3.
- أن يتم الابتعاد عن استخدام كلمات لها معنى خاص بلغة برمجة معينة ومثال ذلك كلمة (mod) لها معنى خاص في أحد اللغات بأنها عملية باقي القسمة.
- أن يتم ترتيب الخطوات ترتيباً صحيحاً لأن الترتيب من الأمور الهامة لحل المشاكل؛ فجمع رقمين لا بد أن يسبق قراءة الرقمين من المستخدم أو معرفتهم وبعد ذلك يتم جمعهم والعكس بالتأكيد ليس صحيحاً.
- أن يتم توضيح خطوات قراءة المعطيات أو إدخال القيم دون الحاجة إلى تعريف أنواعها، وإن كان البعض يذهب إلى سردها في البداية وتوضيح الهدف منها وهذا ليس بالخطأ ولكنه غير مترافق عليه، وعموماً كلاهما مقبولاً.

طريقة الشيفرة الوصفية لا يتم تنفيذها من خلال الحاسوب وإنما تساعده المبرمج على التفكير جيداً في المشكلة وحلها قبل أن يبدأ في كتابة الكود البرمجي مباشرة.



**مثال توضيحي 2.2:**

أكتب الشیفرة الوصفیة الالزامیة لحساب وطباعة معدل درجات الحرارة لمدینة خان یونس خلال شهور  
ینایر، فبراير ومارس.

**الحل 2.2.1:**

اقرأ قيمة درجات الحرارة  $T_{Jan}$ ,  $T_{Feb}$ ,  $T_{Mar}$   
استخدم المعادلة التالیة لحساب المتوسط =  $(T_{Jan} + T_{Feb} + T_{Mar})$  مقسوماً على 3  
اطبع المتوسط

**الحل 2.2.2:**

اقرأ قيمة درجات الحرارة  
المتوسط = مجموع درجات الحرارة مقسوماً على 3  
اطبع المتوسط

**الحل 2.2.3:**

أدخل قيمة درجات الحرارة حسماً يلي :  
 $T_{Jan}$  لدرجة حرارة ینایر،  $T_{Feb}$  لدرجة حرارة فبراير،  $T_{Mar}$  لدرجة حرارة مارس  
 مجموع الدرجات =  $T_{Jan} + T_{Feb} + T_{Mar}$   
 المتوسط = مجموع الدرجات ÷ 3  
 اطبع المتوسط

الحلول الثلاثة الماضية للمشكلة البرمجية تعتبر صحيحة ومنضبطة بالضوابط التي تحدثنا عنها سابقاً  
 ففي الحل 2.2.1 فمنا بدمج خطوة الجمع لدرجات الحرارة مع خطوة حساب المتوسط ثم طبعنا المتوسط وهذا  
 صحيح، بينما في الحل 2.2.2 تجاهلنا من الأصل أن نذكر أسماء متغيرات وجعلنا التعبيرات عامة بشكل ما  
 دون أن يتأثر المعنى أو أن يكون هناك فرصة للفهم الخاطئ وأوضحتنا كيفية حساب المتوسط بشكل صحيح،  
 بينما الحل الأخير 2.2.3 كان وضوحاً وتفصيلاً ولعله يكون الأكثر طلباً عند المبرمجين خاصة في المشاكل  
 البرمجية الأكثر تعقيداً لأنه يوضح الصغيرة قبل الكبيرة وفي النهاية جميعهم يعتبر صحيح.

لاحظ معي في المثال 2.2 أن الحل اشتمل على نمط واحد فقط وهو النمط المتسلسل وبالتالي جاءت الخطوات متتالية دون دوران معين أو اختيار، وبالتالي تجد أن الخطوات تكتب جميعها على بداية السطر في إشارة إلى أنها متتالية وجميعها تتبع بعض، وذلك بخلاف المثال التالي الذي سنظهر فيه نمط الاختيار.

### مثال توضيحي 2.3:

اكتب الشيفرة الوصفية اللازم لحساب معدل درجات الحرارة لمدينة خان يونس خلال شهور يناير، فبراير ومارس، ثم طباعة "الجو بارد" إن كان المعدل أقل من أو يساوي 15 درجة وطباعة "الجو معتدل" إن كان أكبر من ذلك.

### الحل 2.3:

أدخل قيمة درجات الحرارة حسبما يلى :

$TJan$  لدرجة حرارة يناير،  $TFeb$  لدرجة حرارة فبراير،  $TMar$  لدرجة حرارة مارس

$$\text{مجموع الدرجات} = TJan + TFeb + TMar$$

$$\text{المتوسط} = \frac{\text{مجموع الدرجات}}{3}$$

إذا كان المتوسط أقل من أو يساوي 15 درجة

اطبع "الجو بارد"

و الا

اطبع "الجو معتدل"

في المثال 2.2 طلب منا طباعة المتوسط مباشرة بعد حسابه، بينما في المثال 2.3 كان هناك خطوتين هما:

- اطبع الجو بارد

- اطبع الجو معتدل

وقد تم التحذير بين تنفيذهما، فيطبع واحدة فقط بناء على نتيجة شرط ما وهو (هل المتوسط أقل من أو يساوي 15؟) فإن كان نعم نطبع (الجو بارد) وإلا نطبع (الجو معتدل) وهذا النمط الذي قلنا عنه سلفاً نمط الاختيار. كذلك انتبه معي إلى أن بداية كلًا من الخطوتين اللتين تم التحذير بينهما تم إزاحتهم لليسار ثمانى مسافات لكي يفهم القارئ أن كلًا منهما تابع لنتيجة الشرط فالجملة (الجو بارد) تابع لصحة الشرط والجملة (الجو معتدل) تابع لأن يكون الشرط خاطئ وهو ما يشار له بالكلمة (إلا).

من البساطة والوضوح في كتابة الشيفرة الوصفية أن تكتب الإشارات بمعناها كاملاً وليس بالرموز ، فلا تكتب ( $\text{المتوسط} >= 15$ ) وإنما تكتب ( $\text{المتوسط} \geq 15$ ) أو يساوي ( $= 15$ ) ولا تكتب ( $\text{س}$ ) وإنما نكتب العدد ( $15$ ) .

في المثالين الماضيين كتبنا الشيفرة الوصفية بالنطاق المتسلسل ثم الاختيار ومعه المتسلسل ، فكيف العمل إذا طلب منا تكرار هذا المطلوب لكافة مدن فلسطين؟! هل نكرر الخطوات لكل المدن؟! بالتأكيد الجواب لا، فهناك نمط الدوران والذي يمكننا من إضافة القليل من الكلام على الصيغة التي نكتب لعينة واحدة ثم تصبح الشيفرة الوصفية مناسبة للعمل لآلاف العينات ويزيد، تابع المثال التوضيحي 2.4.

#### مثال توضيحي 2.4:

أكتب الشيفرة الوصفية اللازمة لحساب معدل درجات الحرارة خلال شهور يناير، فبراير و مارس، ثم طباعة الجو بارد إن كان المعدل أقل من أو يساوي 15 درجة و طباعة الجو معتدل إن كان أكبر من 15 و ذلك لكافة مدن فلسطين.

#### الحل 2.4:

أدخل عدد مدن فلسطين.

ضع عدداً عند القيمة واحد.

كرر الخطوات التالية طالما العدد أقل من أو يساوي عدد مدن فلسطين.

أدخل قيمة درجات الحرارة للمدينة الحالية حسبما يلي:

$TJan$  لدرجة حرارة يناير،  $TFeb$  لدرجة حرارة فبراير،  $TMar$  لدرجة حرارة مارس

مجموع الدرجات =  $TJan + TFeb + TMar$

المتوسط = مجموع الدرجات  $\div 3$

إذا كان المتوسط أقل من أو يساوي 15 درجة

اطبع "الجو بارد"

وإلا

اطبع "الجو معتدل"

قيمة العداد = العداد + 1 وانتقل للمدينة التالية

كما تتابع عند الحاجة لتمثيل حل مشكلة بنمط الدوران لا بد من الإشارة إلى حدود معينة يبدأ ويتوقف عندها الدوران سواء من خلال شرط أو قيمة مدخلة أو عدد مرات تكرار (كما في المثال 2.4). كذلك من المهم في هذا النمط أن تكون الخطوات التابعة للتكرار مُزاحة للداخل ثماني مسافات بعد بداية شرط الدوران لكي يفهم القاريء أنها تابعة للتكرار، ففي هذا المثال تم وضع خطوات قراءة الدرجات وحساب المتوسط للداخل، ثم أدخلنا أمري الطباعة للداخل مرة أخرى لأنهما تابعتين أيضاً لخطوة في الأصل هي تابعة، وهذا الأمر نتعامل معه مثل شجرة العائلة أو شجرة المجلدات فكلما كان لك أجاداً أكثر كنت بالعمق أكثر.

لاحظ معي في المثال التوضيحي 2.4 أننا استخدمنا الأنماط الثلاثة وهي التسلسل من خلال الخطوة الأولى والثانية وكذلك الخطوات الأربع الأولى داخل الدوران، كما استخدمنا نمط الاختيار من خلال خطوة الطباعة، واستخدمنا الدوران لتكرار المطلوب لكافية المدن، وهذه الصيغة تكاد تكون هي الأكثر شيوعاً في عالم تطوير النظم الكبيرة، حيث أنك قلما تجد نظاماً لا يحتوي على خطوات تحتاج إلى اتخاذ قرار (الاختيار) وكذلك تكرار خطوات ما، سواء على مستوى طلبة، موظفين، مدن، كتب، جامعات ... وغير ذلك.

### مثال توضيحي 2.5:

أكتب الشيفرة الوصفية الالزمة لحساب وطباعة المعدل التراكمي لطالب من تخصص البرمجيات وقواعد البيانات مسجل لسبعة مساقات، علمًا بأنّ عدد الساعات الدراسية مقدارها 19 ساعة أكademie.

#### الحل :2.5

ضع عدداً عند القيمة واحد.

ضع قيمة المجموع عند القيمة صفر.

كرر الخطوات التالية طالما العدد أقل من أو يساوي العدد سبعة.

أدخل قيمة الدرجة للمساق الحالي وخرنها في mark

$\text{المجموع} = \text{المجموع} + \text{mark}$

قيمة العدد = العدد + 1 وانتقل للمساق التالي

$\text{المعدل التراكمي} = \text{المجموع} \div 19$

اطبع المعدل التراكمي

المثال 2.5 يعرض لك الحالة التي يجتمع فيها نمط الدوران مع التسلسل فقط وهو النمط الذي لم نتعرض له من قبل في هذا الكتاب. وفي هذا المثال احتجنا إلى قراءة درجات لسبعة مسافات وفي كل مرة نجمع الدرجة للمجموع الموجود؛ هذه العمليات ثابتة ونحتاج لتكرارها سبع مرات، في كل مرة من مرات الدوران يوجد عدد من الخطوات المتسلسلة مثل الإدخال، الجمع، زيادة قيمة العداد والانتقال للمساق التالي وهي التي يتم تكرارها.

تجنب استخدام الإشارات المنطقية مثل `<` و `>` و `=` وكذلك العمليات الحسابية أثناء كتابة الشيفرة الوصفية واكتب معناهم ليزداد الأمر وضوحاً وفهمها.



الدوران يتم لمجموعة من الخطوات المتسلسلة أو التي تحتوي على اختيار أو كلاماً، المهم أن تتأكد أنَّ هذه الخطوات يمكن تطبيقها بشكل صحيح لمرة واحدة بعد ذلك نضعها ضمن صيغة الدوران، مثل ذلك: طباعة الأعداد من 1 إلى 100، فهي في الحقيقة طباعة عدد و يمكن أن تتم بسهولة لمرة واحدة، بعد ذلك نضع لها الدوران بداية من 1 حتى الزيادة إلى 100 من خلال بداية و نهاية و زيادة محددة.



## 2.4 طريقة مخطط سير العمليات (Flowchart diagram)

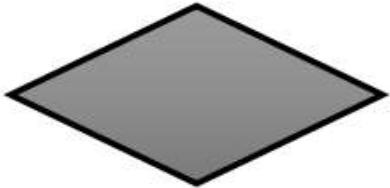
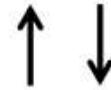
تعتبر طريقة مخطط سير العمليات من أكثر طرق تمثيل حلول المشاكل وضوحاً وسهولة، حيث يتضح فيها ترابط أجزاء الحل بعضه ببعض، وتعُرف على أنها طريقة لبيان خطوات حل المسألة وكيفية ارتباطها ببعض باستخدام رموز اصطلاحية لتوضيح خطوات الحل.

وتتميز بكل ما تتميز به طريقة الشيفرة الوصفية بالإضافة إلى مجموعة أخرى من المميزات هي:

1. سهولة إدخال التعديلات في أي جزء من أجزاء الحل دون الحاجة إلى إعادة دراسته وتتبعه من جديد.
2. سهولة متابعة الخطوات الدقيقة في الحل وتسلسلها وذلك بفضل وضوح الرموز.
3. لا تحتاج إلى التعبير بلغتك كثيراً.

واستخدام هذه الطريقة يخضع لمجموعة من الضوابط وهي:

1. إظهار النمط المستخدم سواء تسلسل أو دوران أو اختيار يعتبر من الضرورات لإيصال الحل واضحًا ودون سوء فهم. (سيتم إفراد هذه النقطة بالشرح المصور فيما يلي).
2. استخدام الرمز المناسب لكل خطوة دون التبديل، فإن استخدام رموز مختلفة يؤدي إلى أداء عمل مختلف وبالتالي نتائج مختلفة وفيما يلي الجدول 2.1 يوضح الرموز ووظائفهم المحددة.

الوظيفة	الرمز
(متوازي الأضلاع) يستخدم للتعبير عن كافة عمليات الإدخال (مثل: قراءة قيمة عدد المدن) وعمليات الإخراج (مثل: طباعة المتوسط). ويتم داخله كتابة عبارات تقييد بالمعنى (مثل: اقرأ عدد الطلبة، أدخل نصف القطر $R$ ، اطبع المتوسط، اطبع الجو بارد).	
(المستطيل) يستخدم للتعبير عن العمليات الحسابية وعمليات التخزين، مثل: (المتوسط = المجموع ÷ العدد) و (العداد = 1)، فهذه المعادلات والعمليات يتم كتابتها داخل شكل المستطيل.	
(المعين) يستخدم لكتابة الشرط الذي بناءً على نتيجته سيتم اتخاذ القرار أو الاختيار بين خطوتين أو أكثر ويتم كتابتها داخل المعين، مع إمكانية كتابة رموز سبقت الإشارة لكتابية معناها في المخطط. مثل ( $6 > X$ ) و (المتوسط $= 15$ )، وغير ذلك من الشروط.	
(بيضاوي) ويستخدم فقط للإشارة إلى بداية ونهاية المخطط وبالتالي فهو أول وأخر رمز يرسم في المخطط، دون كتابة أي لفظ داخله.	
(الأسماء) وتستخدم لتوضيح اتجاه سير العمليات دون أن يرافقها الفاظ سوى في حالة اتخاذ القرار عندما يكتب بجوار السهم (نعم) أو (لا) في إشارة إلى نتيجة الشرط.	
(دائرة) تستخدم لربط أكثر من اتجاه ليتحدد تدفقهم في اتجاه معين وهي تستخدم لتوحيد الاتجاهات للخطوات التي لا تتبع شرط أو دوران.	

مخطط سير العمليات يعتبر التمثيل الرمزي لخطوات حل المشكلة.



مخطط سير العمليات يُعرف أيضًا باسم خرائط التدفق.



وكما كان لطريقة الشيفرة الوصفية أنماطاً ثلاثة فإن مخطط سير العمليات له أنماط مماثلة سنتناقشها في الفصول الفرعية التالية.

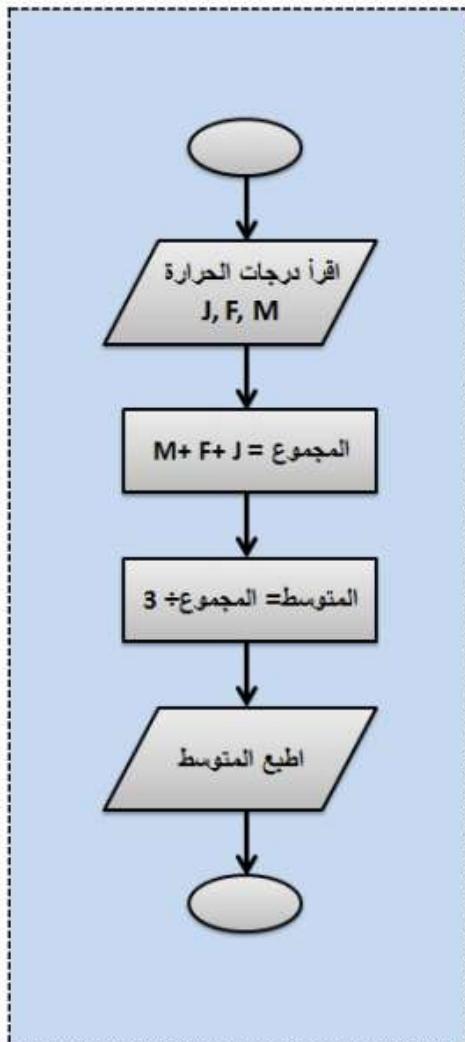
#### 2.4.1 مخطط التتابع البسيط:

هو نوع من المخططات يظهر بشكل سلسلة مستقيمة من بداية البرنامج حتى نهايته بحيث تتعدم فيها القراءات وتخلو من الدوران. وليتضح هذا النوع بشكل أفضل سنعيد حل المثال توضيحي 2.2 بمخطط سير العمليات.

#### مثال توضيحي 2.6:

ارسم مخطط سير العمليات اللازم لحساب وطباعة معدل درجات الحرارة لمدينة خان يونس خلال شهور يناير، فبراير ومارس.

الحل 2.6:



في هذا المثال ومن خلال التفكير والتحليل لعناصر المشكلة كاملة يتضح أن الوصول من المعطيات المطلوب لا يتطلب عملية اتخاذ قرار (اختيار بين أمرين) كما أن الخطوات ذاتها تحتاج لتنفيذها مرة واحدة ، وبالتالي لا يوجد كذلك حاجة إلى الدوران.

من هنا تجد أن الحل للمشكلة يقتصر على التتابع البسيط للخطوات، سواء من القراءة لدرجات الحرارة ثم حساب المجموع والمتوسط ثم طباعة المتوسط.

وعن رسم المخطط، يظهر اختيارنا للشكل البيضاوي لبدء ونهاية الحل، كذلك اختيارنا متوازي الأضلاع في عملية إدخال البيانات اللازمة وهي قراءة الدرجات ثم طباعة المتوسط في ختام الحل، بقى فقط أتنا استخدمنا المستطيل في عمليتين حسابيتين هما حساب المجموع ثم حساب المتوسط.

في بعض الأحيان قد يعتبر بعض محلو النظم أن قراءة درجات الحرارة ثلاثة مرات هي عملية تكرار وبالتالي يمثّلها بتكرار وهو توجّه ليس خاطئاً



#### 2.4.2 مخطط التفرع:

هو نوع من الخرائط يحتوي على تفرع ناتج عن الحاجة لاتخاذ قرار، أو المقارنة بين اختيارين أو أكثر (باستخدام العمليات المنطقية والتي سنتعرض لها بالتفصيل في الباب الرابع)، فيسير كل اختيار في اتجاه مختلف عن الآخر. الشكل التوضيحي 2.6 يُبيّن الشكل العام لهذا النمط وكيفية التعامل مع هذين المسارين وهما ناتج العملية المنطقية (نعم أو لا) أو (صح أو خطأ). الأسئلة المنطقية دائمًا ما تكون إجابتها نعم أو لا، مثل ذلك (هل تقدمت لامتحان مساق مبادئ البرمجة اليوم؟ الإجابة لا تكون إلا نعم أو لا).



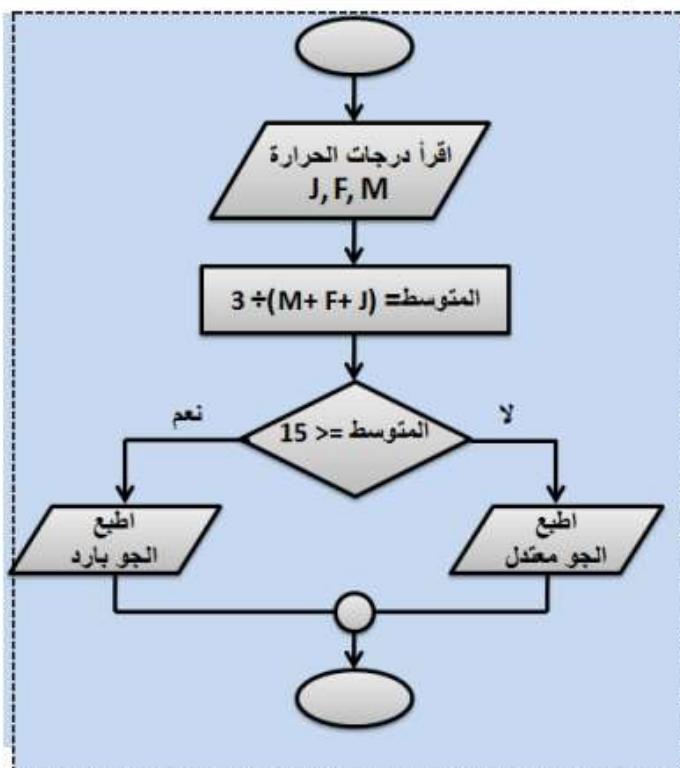
شكل 2.6: مقارنة لها فرعان من الناتج

**ربط اتجاهات سير العمليات** يتم من خلال دائرة صغيرة تمثل دور مفترق الطرق الدائري الذي يتجمع فيه كافة خطوط السير المختلف، ويتم استخدام هذا الرمز عندما تكون هناك مجموعة خطوط سير مختلفة داخل البرنامج وحان الآن موعد اتفاقهم فيتم ذلك من خلال هذه الدائرة الصغيرة.



#### مثال توضيحي 2.7:

ارسم مخطط سير العمليات اللازم لحساب معدل درجات الحرارة لمدينة خان يونس خلال شهور يناير، فبراير ومارس، ثم طباعة الجو بارد إن كان المعدل أقل من أو يساوي 15 درجة وطباعة الجو معتدل إن كان أكبر من ذلك.



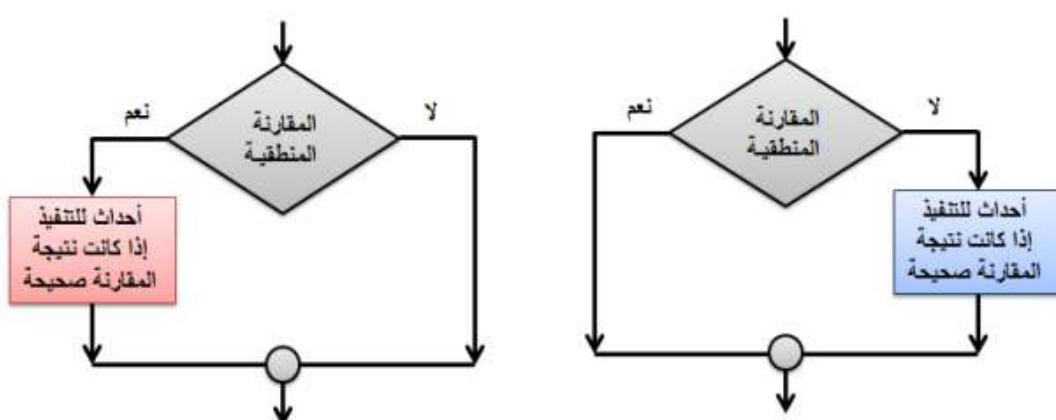
من خلال تحليل هذه المشكلة ستوصى إلى أن حل المشكلة يتطلب حساب المتوسط ثم اتخاذ قرار من خلال مقارنة المتوسط مع القيمة 15 لنطبع "الجو بارد" إن كان أقل أو يساوي وإلا نطبع "الجو معتدل".

وكما تلاحظ، قرأنا درجات الحرارة داخل متوازي الأضلاع لأنها "عملية إدخال" وبعد ذلك قمنا بحساب المتوسط بمعادلة حسابية تم كتابتها داخل مستطيل، أما عملية اتخاذ القرار فكانت من خلال وضع شرط المقارنة داخل معين مع توضيح اتجاه التنفيذ.

يبقى الانتباه إلى أننا في كلا الحالتين سننهي البرنامج فيتم ذلك من خلال نقطة الربط وهي دائرة صغيرة كما ترى.

مخططات التفرع تتقسم لثلاث حالات بناءً على عدد الاتجاهات التي يتم تنفيذها بعد ظهور نتيجة المقارنة وهم:

- **مخطط تفرع بحالة تنفيذية واحدة:** وهو تفرع يوجد به تعليمات تنفذ فقط في حالة واحدة عند نتيجة المقارنة (نعم) أو (لا)، فنتيجة المقارنة إن كانت مثلاً صحيحة يتم تنفيذ خطوات محددة في اتجاه واحد بينما الاتجاه الآخر فلا يتم تنفيذ شيء فيه. ويوضح هذا في الشكل التوضيحي 2.7، مثال هذه الحالة المثال التوضيحي 2.8.

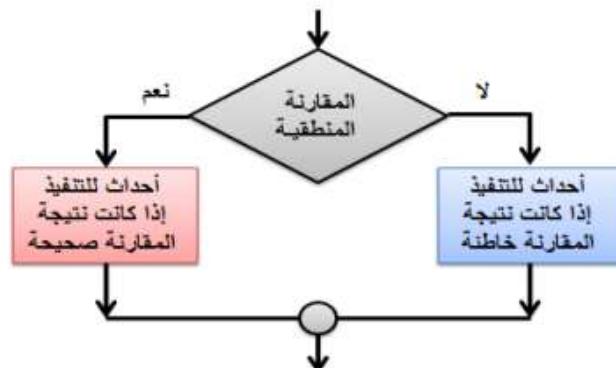


شكل 2.7: مخططات تفرع بحالة تنفيذية واحدة

- مخططات تفرع بحالتي تنفيذ: وهو نقرع يوجد به تعليمات تتضرر التنفيذ في الاتجاهين، أي أنّ نتيجة المقارنة إن كانت صحيحة فهناك تعليمات سيتم تنفيذها، وإن كانت النتيجة خاطئة فهناك تعليمات أخرى سيتم تنفيذها.

هذه الحالة هي الأكثر شيوعاً بين الحالات الثلاث وتوضح هذه الحالة في الشكل التوضيحي 2.8 ومثال على هذه الحالة المثال التوضيحي السابق 2.7.

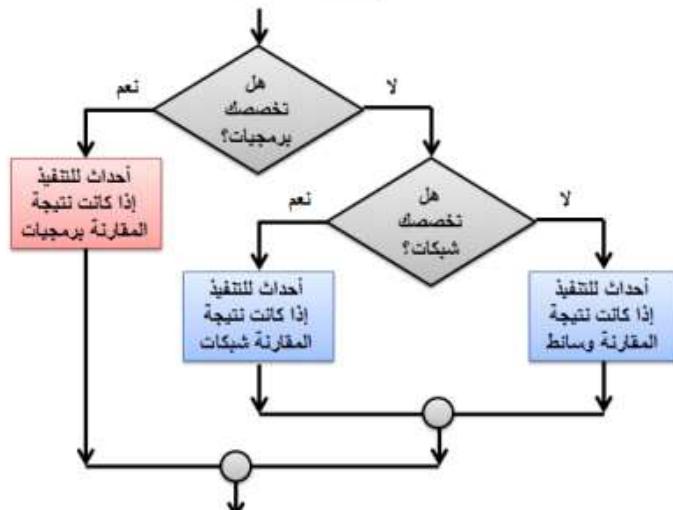
- مخططات تفرع بثلاث حالات تنفيذ أو أكثر: ويستخدم هذا النوع في حالات محددة كالإجابة على سؤال (إن كنت تخصص برمجيات ذهب للشعبة الأولى، وإن كنت تخصص شبكات ذهب للشعبة الثانية، وإن كنت تخصص وسائط متعددة ذهب للشعبة الثالثة). هذه الحالة لا تتنمي مباشرة لحالات اتخاذ القرار المعتمدة على مقارنات منطقية فهي مكونة في الأصل من حالة مقارنة متداخلتين من النوع الأول والثاني، إلا أننا نقوم بتيسيرها من خلال رسمنا بمقارنة واحدة. توضح هذه الخريطة من خلال الشكل التوضيحي 2.9 والمثال 2.9. كما أوضحها لك من خلال استخدام مقارنتين في الشكل التوضيحي 2.10.



شكل 2.8: مخططات تفرع بحالتي تنفيذ



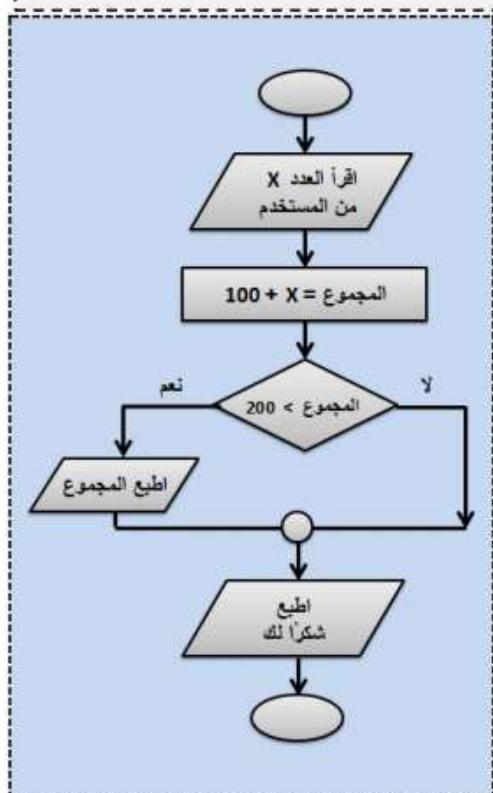
شكل 2.9: مخططات تفرع بثلاث حالات تنفيذ



شكل 2.10: مخططات التفرع بثلاث حالات باستخدام حالتي مقارنة

**مثال توضيحي 2.8:**

ارسم مخطط سير العمليات اللازم لحساب مجموع رقمين الأول يستقبله البرنامج من المستخدم والآخر هو 100 على أن يطبع مجموعهم إن كان أكبر من 200، وفي نهاية البرنامج يطبع للمستخدم عبارة "شكرا لك".



من التفكير والتحليل لهذه المشكلة البرمجية، نجد أن البرنامج المراد سيطلب من المستخدم إدخال عدد من المستخدم ليقوم بجمعه مع العدد 100 وبعد ذلك يتم مقارنة المجموع مع 200 فإن كان المجموع أكبر من 200 يطبع الناتج وفي حال كان المجموع أقل لا يفعل البرنامج شيئاً وهذا هو المقصود بخرائط التفرع الواحد. كذلك البرنامج سيطبع عبارة "شكرا لك" وهي عبارة ستطبع على كل حال أي سواء كان المجموع أكبر من 200 أو غير ذلك. كما تتبع في مخطط سير العمليات، فإن الجزء الخاص بالمقارنة المنطقية تحتوي على خطوات سيتم تنفيذها فقط في حال كانت المقارنة ( $\text{المجموع} > 200$ ) صحيحة بينما إن كانت خاطئة فلا يوجد تنفيذ لخطوات خاصة بهذه الحالة.

بقي أن ننتبه إلى خطوة طباعة "شكرا لك" وهي التي تطبع دون تأثيرها بنتائج المقارنة فهي ستطبع على أي حال وبالتالي تم الوصول لها من خلال توحيد خطوط السير قبلها من خلال نقطة التجميع التي تراها أمامك.

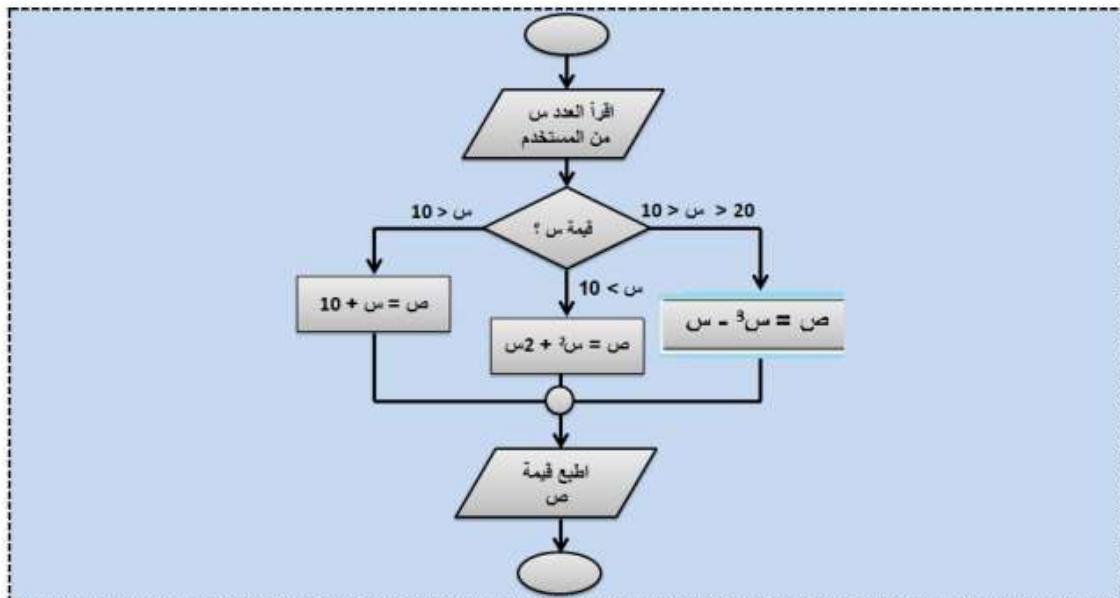
**تدريب مفيد:** في المثال السابق حاول رسم مخطط سير العمليات للمثال ذاته مع إضافة مقارنة أخرى توضع في حال كان المجموع أكبر من 200 بالفعل لتقارن هل هو أصغر من 300 أيضاً فإن كان نعم "يطبع أكبر من 200 و أقل من 300" و إلا يطبع "أكبر 300"



### مثال توضيحي 2.9

ارسم مخطط سير العمليات اللازم لحساب قيمة ص من المعادلة التالية ومن ثم طباعتها:

$$\left. \begin{array}{l} \text{ص}^2 + 2\text{s} , \quad \text{s} < 10 \\ \text{s}^3 - \text{s} , \quad 20 < \text{s} < 10 \\ \text{s} + 10 , \quad \text{s} > 10 \end{array} \right\}$$



وهذا المثال تلاحظ أن حساب قيمة ص لها ثلاثة احتمالات وهذه الاحتمالات مرتبطة بقيمة ص التي لها ثلاثة خيارات، وبالتالي فإننا نحتاج إلى مطالعة قيمة ص ومع كل قيمة (مدى) نحسب قيمة ص بطريقة مختلفة وهكذا، لذلك احتجنا إلى استخدام مخطط تفرع له ثلاثة تفرعاتها أو خيارات. انتبه إلى أنها استخدمنا نقطة التجميع في هذا المثال أيضاً لأن احتمالية سير العملية قد تأتي من ثلاثة طرق، وعلى كل حال فإن العملية ستصل إلى نقطة طباعة قيمة ص وبالتالي يتم تجميع كافة الطرق لتصل إليها.

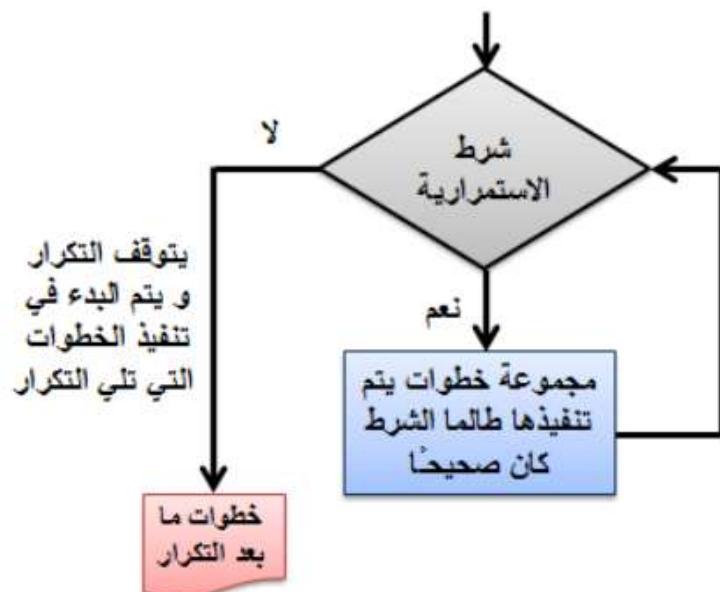
بهذا تكون قد ضربنا المثال لكافة صور خرائط القرع العامة، ويبقى أن تعرف أن هذه الصور هي أنماط تم التعرف عليها من خلال تحليل المشاكل البرمجية، وبالتالي فهذه أشهر الحالات ولا يعني ذلك أنه لا يوجد حالات أخرى أو أنماط أخرى، والفيصل في هذا الأمر هو التفكير الجيد في المشكلة التي تُعرض عليك ثم القيام بتحليلها وبسيطها وتقوم بعد ذلك برسمها شيئاً فشيئاً من الأسهل أو الأصعب إلى الأصعب أو إلى الأكبر، ستجد نفسك بهذه الصورة قد انتهيت من رسم المخطط بصورة صحيحة. التدريبات الخاصة بهذا الباب ستكون فرصة جيدة للتمرين والتدريب.

**تدريب مفید:** أعد رسم مخطط سير العمليات للمثال السابق باستخدام مخطط سير عمليات بحالتي تفرع وحالة تفرع واحدة فقط مع الاستعانة في فهمها بالشكل التوضيحي 2.10



#### 2.4.3 مخططات الدوران البسيط

من أبرز ما يميز الحاسوب قدرته على تكرار العمليات بعدد معين أو حسب شرط محدد دون تعب أو ملل وخرائط الدوران هي خرائط تستعمل للتعبير عن تكرار أو إعادة عملية أو مجموعة من العمليات في البرنامج عدداً محدوداً أو غير محدود من المرات، وقد سُمي بالبسيط لأنّه يحتوي على حلقة دوران واحدة فقط ويسمى أحياناً خرائط الدوران الواحد. ويكون الشكل العام لهذه الخرائط كما في الشكل التوضيحي 2.11.



شكل 2.11: مبدأ عمل مخططات الدوران البسيط

والدوران أو التكرار في الحوسبة يحكمه ثلاثة عوامل أساسية هي:

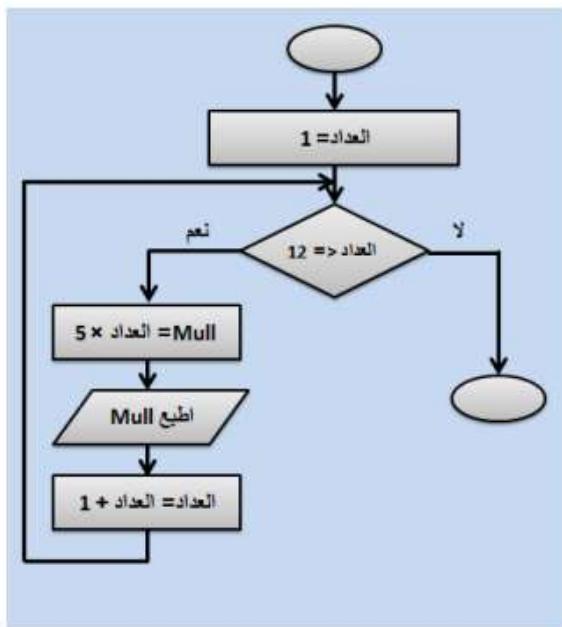
- نتيجة شرط الدوران: وهو الشرط الذي بناء على نتيجته يستمر الدوران أو يتوقف، وليس شرطاً أن يستمر الدوران عندما تكون نتيجة الشرط (نعم) فهناك حالات قد يستمر الدوران عندما تكون نتيجة الدوران خاطئة. [فكّر لضرب أمثلة لهذه الحالات]
- عامل يؤثر على نتيجة شرط الدوران: فقد يعتمد الدوران على قيمة يدخلها المستخدم أو على رقم ما يصل له الدوران أو يتجاوزه بالوجب أو السالب، وهذه القيمة تعتبر هي العامل الذي يؤثر على نتيجة الشرط، فهذا العامل لا بدّ أن يكون حاضراً ضمن الأحداث التي يتم تنفيذها كل دوران جديد مع توفر فرصة لتغيير قيمته بما يسمح بإنها الدوران.
- قيمة بادانية يبدأ عامل الدوران من عندها: لأن شرط استمرارية الدوران مرتبطة بعامل معين سواء كان قيمة رقمية أو قيمة نصية فإنّ هذا العامل لا بدّ أن يحتوي على قيمة بادانية يبدأ الدوران بها ثم يتم التعديل عليها من خلال خطوات الدوران حتى يتوقف.

**القيمة النصية:** يقصد بها عبارات أو كلمات مثل "أحمد"، "بارد"، "الجو معتدل"، بينما  
**القيمة الرقمية:** يقصد بها الأرقام بمختلف أنواعهم مثل "5"، "7.8"، "870.65". بينما  
**القيمة المنطقية:** هي إحدى قيمتين "صحيحة"، "خاطئة".



## مثال توضيحي 2.10:

رسم مخطط سير العمليات اللازم لطباعة جدول الضرب من 1 إلى 12 للرقم 5.



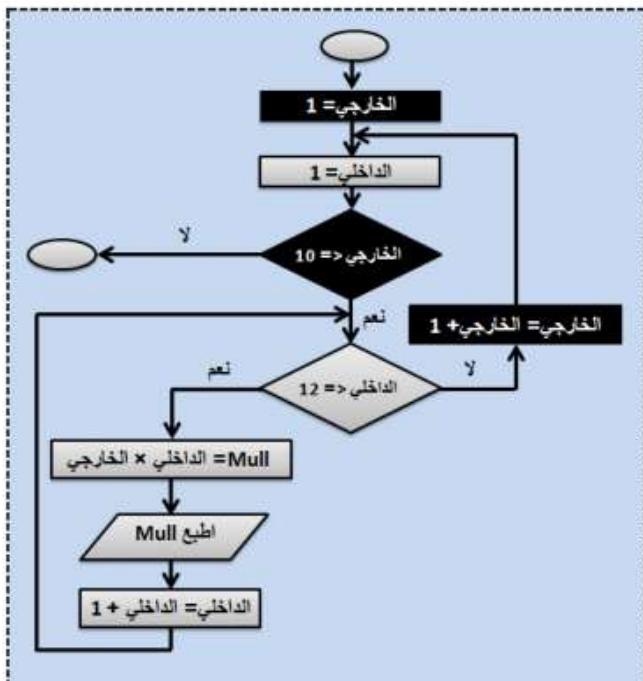
طباعة جدول الضرب لأي رقم من الأرقام عبارة عن عملية حسابية مكونة من رقمين، الرقم الأول وهو ثابت هو رقم الجدول بينما الرقم الثاني هو رقم متغير بمتسلسل من 1 إلى 12، فيبدأ بالرقم 1 ثم يزداد مع كل تكرار واحد وبالتالي مع التفكير البسيط نرى أن العامل الذي يؤثر على نتيجة الدوران هو الرقم المتغير ويبدا عند القيمة 1 ويزداد حتى يصل للرقم 12، في هذه الحالة سيكون شرط استمرار الدوران أن يكون العدد أقل من أو يساوي 12 وهو الحد الأقصى للتكرار، بهذا يبدأ العدد من 1 ويستمر بالدوران حتى يصل إلى 12.

### تبنيات مهمة حول مثال توضيحي 2.10:

- العامل الذي يؤثر على نتيجة شرط الدوران هو (العداد) والذي يتم تغييره بزيادة أو نقصان في كل دورة من دورات الدوران حسب طبيعة المشكلة (في المثال الحالي يتغير من خلال الزيادة بمقدار واحد).
- شرط الدوران هو (هل تجاوز العدد للقيمة 12 أم لا؟) وذلك لأن جدول الضرب للرقم 5 من العدد 1 حتى 12.
- انتبه إلى أننا في كل دورة من دورات الدوران لا بد أن نقوم بزيادة قيمة العدد بعد طباعة قيمة عملية الضرب لا قبله. [فكير ما الناتج إذا تمت الطباعة ثم الزيادة؟]
- لاحظ أن التعبير عن الدوران في مخطط سير العمليات تم من خلال سهم يتم توجيهه بعد الخطوات التي سيتم تكرارها إلى قبل الشرط وهكذا.
- مخطط الدوران البسيط هو مخطط يحتوي على دوران واحد فقط، بينما يمكن أن يتم تضمين دوران داخل دوران ليصبح الدوران معقداً (مركباً) وهذا الذي نضرب المثال 2.11 لتوضيحه.

### مثال توضيحي 2.11:

ارسم مخطط سير العمليات اللازم لطباعة جدول الضرب من 1 إلى 12 للأرقام من 1 إلى 10.



هذا المثال يوضح مفهوم الدوران المركب، وهو دوران يتم تنفيذه داخل دوران آخر، فكما تلاحظ الدوران الخارجي طرفه الموجب يدخلنا إلى الدوران الداخلي.

طباعة جدول الضرب للأرقام من 1 إلى 10، للأرقام من الرقم 1 إلى 12 يعني أننا نريد تطبيق المثال 2.10 بعدد 10 مرات، وبعد انتهاء كل مرة يتم زيادة العدد الخارجي بمقدار 1 ثم نعود مرة أخرى لطباعة جدول الضرب الجديد.

تتبع معى المخطط الخاص بمثال 2.11 مع إعطاء الرقم 1 للعدد الخارجي وكذلك العدد الداخلي. الشرط المنطقى الخارجى يحكم عدد جداول الضرب التى سنقوم بطباعتها، ففي كل تكرار يتم طباعة جدول ضرب لرقم من الأرقام 1 إلى 10 على أن يكون الحاصل هو حاصل ضرب (العدد الخارجى \* العدد الداخلى) والخارجى هنا يثبت لعدد 12 دورة بينما يتم تغيير الداخلى كل دورة بزيادة واحد. عند وصول العدد الداخلى للعدد 13، فلا يُكمل الدوران لأن شرط الاستمرار هو (الداخلى >= 12) وبالتالي يخرج سير العملية خارج الدوران الداخلى ويزيد العدد الخارجى بواحد ثم يتم اختبار الشرط الخارجى (هل تخطينا العدد 10 أم لا؟)، فإن تخطينا العدد 10 يتم انهاء البرنامج وإلا يتم البدء في طباعة جدول الضرب التالى وهكذا.



**مهارة مفيدة:** في بعض الحالات يقوم الدارس برسم مخطط سير عمليات لحل مشكلة برمجية ويحتاج إلى التأكد من صحته في إخراج المطلوب، في كل الحالتين لا بد له من تتبع هذا المخطط من خلال إعطاء قيم بدانية للمتغيرات (مثل الخارجى، الداخلى، العداد) ليظهر له هل هو صحيحاً أم خطأنا.

دعونا نتبع المخطط الناتج من مثال 2.11 مع استبدال الشرط الخارجى بـ (الخارجى >= 3) والداخلى بـ (الداخلى >= 4)، سيكون الناتج كما هو موضح بالجدول التالي:

شرح	الطباعة بهذه القيم الداخلى × الخارجى	نتيجة الشرط الداخلى (الخارجى >= 3)	نتيجة الشرط الخارجى (الداخلى >= 4)	قيمة العدد الداخلى	قيمة العدد الخارجى
قيمة جدول الضرب للرقم 1 من 1 إلى 4	1	3 => 1	4 => 1	1	1
	2	3=>1	4=>2	2	1
	3	3=>1	4=>3	3	1
	4	3=>1	4=>4	4	1
لا يطبع وإنما يخرج للتكرار الخارجى مع زيادة العدد الخارجى بواحد		3=>1	4=>!5	5	1
قيمة جدول الضرب للرقم 2 من 1 إلى 4	2	3=>2	4=>1	1	2
	4	3=>2	4=>2	2	2
	6	3=>2	4=>3	3	2
	8	3=>2	4=>4	4	2
لا يطبع وإنما يخرج للتكرار الخارجى مع زيادة العدد الخارجى بواحد		3=>2	4=>!5	5	2
قيمة جدول الضرب للرقم 3 من 1 إلى 4	3	3=>3	4=>1	1	3
	6	3=>3	4=>2	2	3
	9	3=>3	4=>3	3	3
	12	3=>3	4=>4	4	3
لا يطبع وإنما يخرج للتكرار الخارجى مع زيادة العدد الخارجى بواحد		3=>3	4=>!5	5	3
بهذا ينتهي الدوران الخارجى أيضاً		3=>4!	4=>1	1	4

## 2.5 أمثلة وتدريبات عامة

### 1. استخدم الكلمات المناسبة لتعبئة الفراغات في الجمل التالية:

أ- من أبرز القواعد التي توصل لها العالم الرياضي ديكارت، لا شيء يعتبر حقيقة ثابتة قبل أن يثبت بـ \_\_\_\_\_.

\_\_\_\_\_ و \_\_\_\_\_.

ب- تطبيق قاعدة "فرق تسد" لحل المشكلة يقصد بها \_\_\_\_\_.

ت- تقسيم المشكلة أثناء محاولة حلها قد يتم على أحد نوعين \_\_\_\_\_ و \_\_\_\_\_.

ث- التخطيط لحل المشكلة هي مرحلة تتبع مرحلتي \_\_\_\_\_ ثم \_\_\_\_\_.

ج- يتم تدوين التخطيط من خلال علم \_\_\_\_\_ والذي تم استخدامه من خلال العالم المسلم \_\_\_\_\_.

\_\_\_\_\_.

ح- الخوارزميات تعمل على توثيق التفكير والتخطيط من خلال طريقتين هما \_\_\_\_\_ و \_\_\_\_\_.

خ- الشخص الذي يقوم بتحليل عناصر المشكلة والتفكير في كيفية حوسبيتها مع تدوين ذلك بوسائل مختلفة يعرف باسم \_\_\_\_\_.

د- \_\_\_\_\_ هو دراسة المشكلة بهدوء وتركيز لوضع اليد على معطياتها ومطالبتها.

ذ- عندما يكتشف المستخدم النهائي أن النظام يُخرج نتائج خاطئة يتم إعادة النظام في البداية إلى \_\_\_\_\_ ثم قد يحول إلى \_\_\_\_\_.

ر- الخوارزميات قد تتكون من ثلاثة أنماط هي \_\_\_\_\_ و \_\_\_\_\_ و \_\_\_\_\_.

### 2. حدد صحة أو خطأ كلاماً من العبارات التالية مع التعليل:

- ( ) أ- الشيفرة الوصفية تعتبر إحدى لغات البرمجة التي لا يتم تنفيذها من خلال الحاسوب.
- ( ) ب- من الأمثلة على القيم النصية "564" و "محمود" بينما المنطقية "صحيحة".
- ( ) ت- من مميزات الشيفرة الوصفية أن محل النظم لا يحتاج التعبير بأسلوبه كثيراً.
- ( ) ث- من ضوابط كتابة الشيفرة الوصفية أن تكون الكلمات المستخدمة سهلة الفهم قدر الإمكان.
- ( ) ج- الهدف من مرحلة التفكير أن يصبح محل النظم قادرًا على تحليل عناصر المشكلة.

- ح- تبع الحاجة لمرحلة التخطيط في أنها تساعد محل النظم على تحديد المساحة التخزينية التي قد يحتاجها الحاسوب لحل المشكلة.
- ( )
- خ- يستخدم المستطيل في مخطط سير العمليات للإشارة إلى عمليات الإدخال والإخراج.
- د- لكل مشكلة برمجية يمكن كتابة شیفرة مزيفة صحيحة واحد فقط بينما بمخطط سير العمليات يمكن رسم أكثر من حل صحيح.
- ( )
- ذ- ذكر الحاجة من المتغيرات في بداية الخوارزمية من الضوابط التي يمكن عدم الالتزام بها.
- ( )
- ر- تعتبر مميزات مخطط سير العمليات أكثر شمولية من مميزات الشیفرة الوصفية.

3. مستخدمنا طریقة الشیفرة الوصفیة ومخطط سیر العمليات، عبر عن حل کل ما يأتي:

- أ- إيجاد مساحة الدائرة التي نصف قطرها  $R$ ، علماً بأن مساحة الدائرة تساوي  $\pi R^2$  ، حيث  $\pi$  هي عبارة عن ثابت قيمته دائماً 3.14 تقريباً.
- ب- إيجاد مساحة ملعب كرة القدم مع تحديد إن كان دولي (في حال كانت المساحة أكبر من أو يساوي 600 متر مربع) أم محلي (أقل من 600 متر مربع)، علماً بأن مساحة المستطيل تساوي (الطول × العرض).
- ت- إذا علمت أن قسم علوم الحاسوب بأحد الكليات يضم عدد  $A$  تخصصات وفي كل تخصص يوجد في الفصل الأول عدد  $B$  مساقات وفي كل مساق يوجد عدد  $C$  شعبة، وكل شعبة تحتوي على عدد  $D$  من الطلبة، نريد برنامجاً يحسب عدد الطلبة الذين ينتمون لقسم في هذا الفصل.

ث- بإيجاد قيمة الدالة  $F(x)$  المعرفة كالتالي:

$$F(x) = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$$

- ج- اكتب الخوارزمية اللازمة لحساب قيمة كل من المتغيرات  $A$ ،  $B$ ،  $C$  في المعادلات التالية:

$$A = X^2 + 2Y$$

$$B = 2X - 3A$$

$$C = A^2 + XB$$

- ح- أعد حل الفقرة (ب) مع خصم مساحة دائرة نصف الملعب من المساحة الإجمالية للملعب.
- خ- يمتلك تاجر مجموعة قطع قماش طول الواحدة يزيد عن 5 أمتار ويرغب في تطوير نظام محosب لقطعها إلى قطع طول الواحدة منها 5 أمتار فقط.
- د- أحد شركات تعبئة المياه المقطرة تمتلك خزانًا للماء ارتفاعه 100 متر ويرغب مدير الشركة في أن يطور نظامًا يتيح له أن يملأ الخزان بشكل ذاتي عندما ينخفض ارتفاع الماء عن 80 متر وإلا يتم إغلاق الأنابيب الذي يملأ الخزان.

4. قامت إحدى الجمعيات الخيرية بالإعلان عن مشروع خيري لإقراض الشباب لإنشاء المشاريع الصغيرة على فرض أن السن المناسب للاستفادة من الإقراض هو 20 إلى 27 عامًا، اكتب الخوارزمية اللازمة لبناء نظام محوسب لتسيير هذا المشروع بشكل محوسب.

عندما يطلب كتابة خوارزمية لمشكلة برمجية ما، يمكن الدارس أن يكتبها بطريقة الشيفرة الوصفية أو برسم مخطط سير العمليات لها.

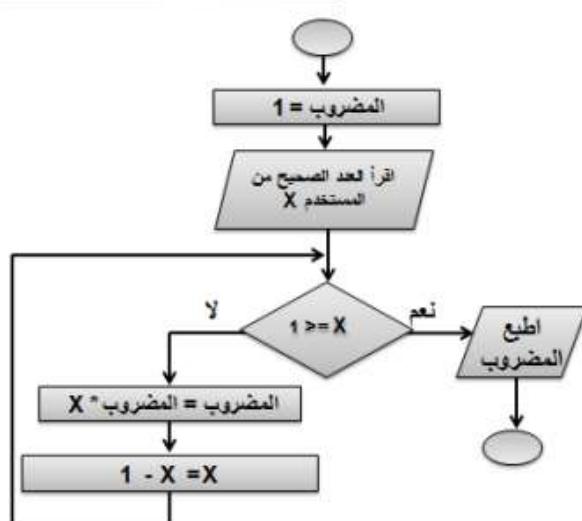


5. ارسم مخطط سير العمليات لنظام يتيح الفرصة للمستخدم لإدخال ثلاثة أرقام ويقوم النظام بطباعة حاصل ضربها على أن يمكن المستخدم من إدخال المزيد من الأرقام إن رغب وإلا يتوقف النظام.

6. تتبع الشيفرة الوصفية التالية، مع شرح الهدف منه، ثم كتابة النتائج المتوقعة:

- |   |                                   |
|---|-----------------------------------|
| 1 | اقرأ الرقم الصحيح $x$ من المستخدم |
| 2 | اجعل المتغير $0 = SUM$            |
| 3 | إذا كان $X$ أكبر من 0             |
| 4 | $X + SUM = SUM$                   |
| 5 | $1 - X = X$                       |
| 6 | اطبع $SUM$                        |

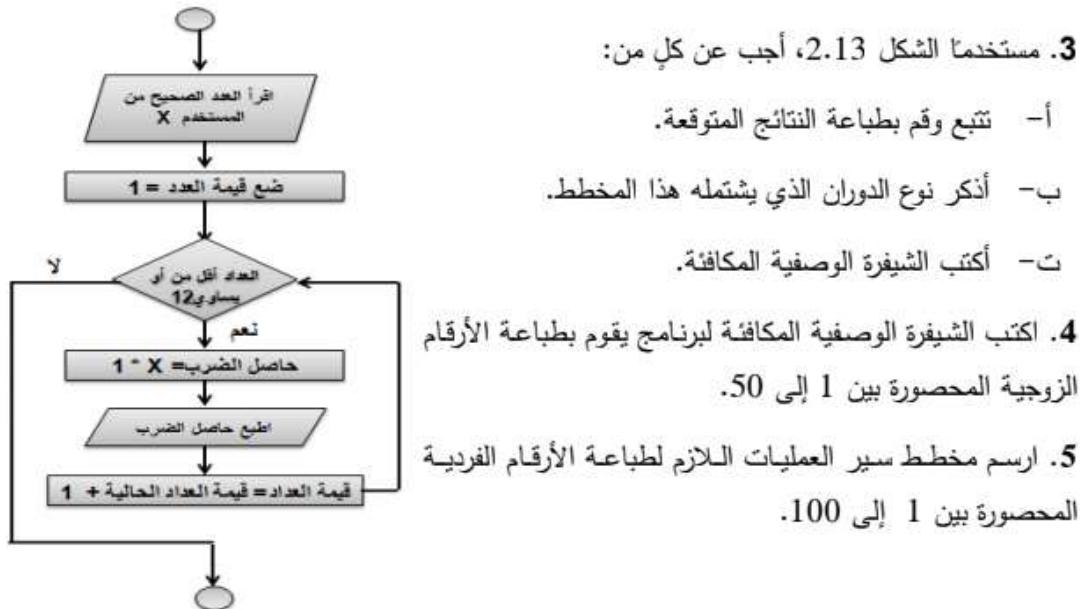
7. تتبع مخطط سير العمليات بالشكل 2.12، مع شرح الهدف منه، ثم كتابة النتائج المتوقعة:



شكل 2.12: مخطط سير العمليات

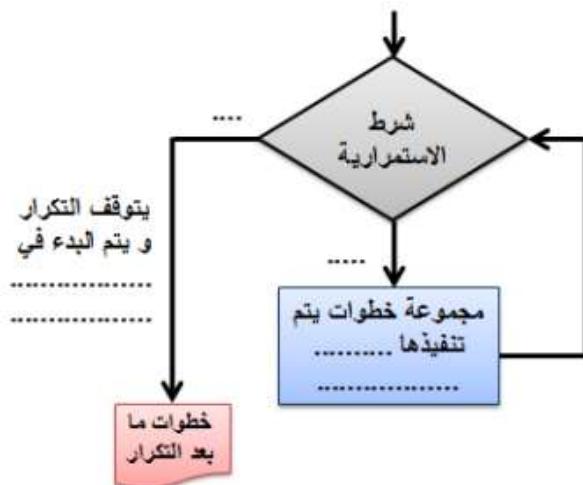
## 2.6 تمارين ذاتية الحل

- الخوارزميات قد تتكون من ثلاثة أنماط من الترتيب ، تم استبانتها من واقع أي عملية في حياتنا ، ويمكن استخدامها فرادى أو مجتمعة، ذكر هذه الأنماط مع ذكر الأمثلة على كل نمط منهم.
- مخططات الدوران البسيط سميت بذلك لأنها تحتوي على دوران واحد فقط بينما المركب يحتوى أكثر من دوران متداخل، اضرب مثلاً على من كل النوعين مدعماً بالرسم والتوضيح.



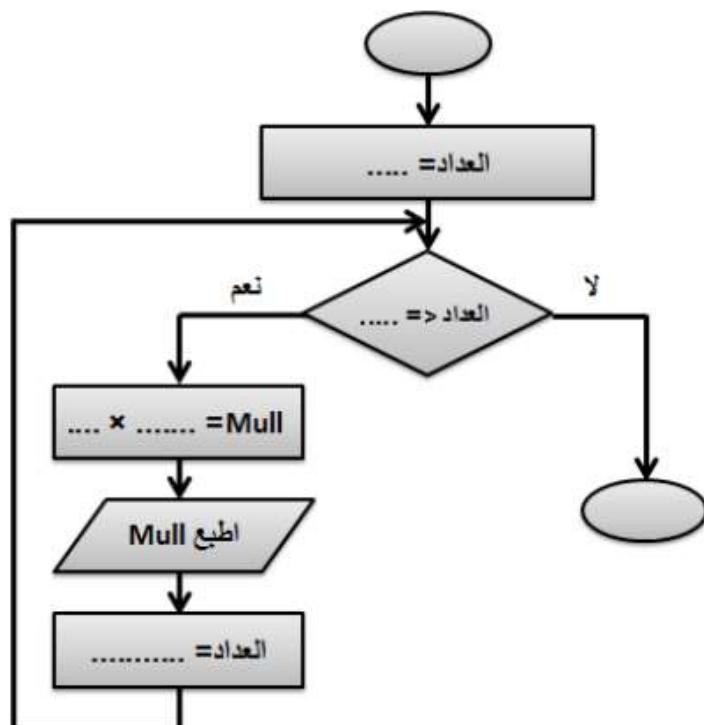
الشكل 2.13 : تدريبات ذاتية، السؤال 3

6. جمعية زراعية تخطط لمشروع دعم الأراضي الزراعية المتضررة من الحرب على غزة بإمدادها بمبلغ مالي قدره 1000 دينار وعدد 350 شتلة زراعية من نوع الزيتون على أن يستفيد من هذا المشروع الأرضي التي تزيد مساحتها عن 10 فدان ويزيد عدد النباتات التي تم تدميرها عن 80، الجمعية تود في عمل نظام لتسيير هذا المشروع.
7. شركة اتصالات تخطط لمشروع تعلن من خلاله أن كل مشترك لديه أكثر من 1000 دقيقة مجانية شهرياً على أن يتم اعتبار كل مكالمة مدتها 100 دقيقة بالضبط ضمن المشروع أما غير ذلك فلا يعتبر، اكتب خوارزمية مناسبة لعمل هذا المشروع.
8. ارسم خريطة سير البرنامج لإيجاد مساحة دائرة نصف قطرها R يتم استقبالها من المستخدم ولكن مع سؤال المستخدم عن رغبته في الاستمرار فإذا أجاب بنعم أعد العملية وإذا أجاب بلا أوقف البرنامج.
9. مطلوب كتابة خوارزمية لبرنامج يحسب عدد الطلبة الناجحين في مادة خوارزميات و مقدمة في البرمجة علما بأن درجة النجاح من 50 وعدد الطلبة 30.
10. اكتب خوارزمية لبرنامج يستقبل من المستخدم رقمين ويطبع مجموعهم و ضربهم وقسمة الثاني على الأول.
11. اكتب خوارزمية لبرنامج يستقبل من المستخدم ثلاثة أرقام ثم يطبع أيهم أكبر.
12. مطلوب منك حساب معدل السكان لمدن قطاع غزة، اكتب الخوارزمية اللازمة.
13. املأ فراغات الشكل 2.14 التالي :



شكل 2.14: مبدأ عمل الدوران البسيط

14. املأ فراغات مخطط سير العمليات التالي ليطبع جدول الضرب للعدد 8 من الأعداد 7 وحتى 20



شكل 2.14: مبدأ عمل الدوران البسيط

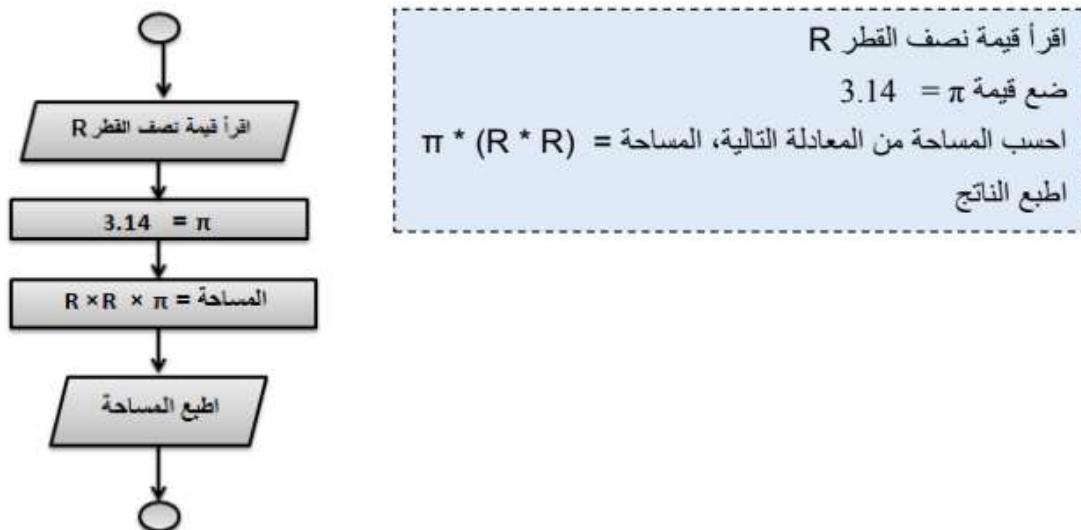
15. ارسم مخطط سير العمليات لتطبيق يقوم بحساب قيمة الرسوم الدراسية الواجب تسديدها من الطالب الذي يحصل على إعفاء الإخوة، علماً بأنَّ قيمة الإعفاء 25% ويشترط للحصول عليه أن يكون للطالب أخ ثانٍ معه وقد مضى على أحدهما فصل دراسي.
16. اكتب الشيفرة الوصفية لتطبيق يقوم بإصدار صوت الأذان عند كل موعد صلاة من الصلوات الخمس.
17. ارسم مخطط سير العمليات لتطبيق يقوم بالبحث عن بيانات موظف ويبطعها.

## اجابات التدريبات العامة ،

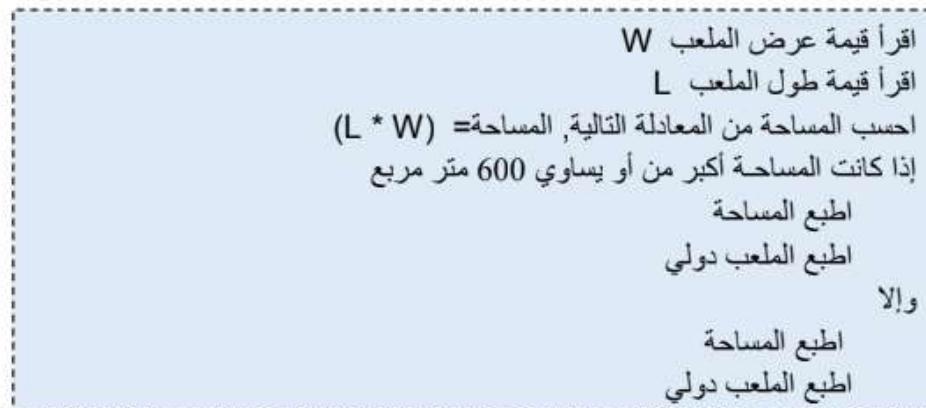
1. (أ) التجارب، المشاهدات (ب) كل مشكلة يمكن تبسيطها ليسهل حلها (ت) المتتالي، المتوازي (ث) التفكير، التحليل (ج) الخوارزميات، محمد بن موسى الخوارزمي (ح) الشيفرة الوصفية، مخطط سير العمليات (خ) محل النظم (د) مرحلة التفكير (ذ) المبرمج، محل النظم (ر) التسلسل، الاختيار، الدوران البسيط

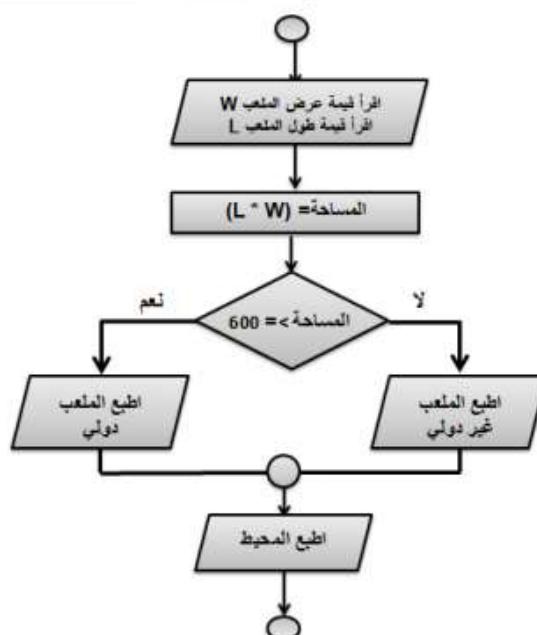
.2

- أ- (خاطئة) الشيفرة الوصفية تعتبر أحد طرق تمثيل الخوارزميات ولا يتم تنفيذها من خلال الحاسوب.
- ب- (خاطئة) من الأمثلة على القيم النصية "محمود" بينما المنطقية "صحيحة".
- ت- (خاطئة) من مميزات مخططات سير العمليات أن محل النظم لا يحتاج التعبير بأسلوبه كثيراً.
- ث- (خاطئة) تمثل الرقم 18 بالنظام الثنائي يتم بالرقم 10010 .
- ج- (خاطئة) الأساس في النظام الثنائي هو 2 أما في العشري هو الرقم 10 .
- ح- (صحيحة)
- خ- (خاطئة) يستخدم المستطيل في مخطط سير العمليات للإشارة إلى العمليات الحسابية والتخزين
- د- (خاطئة) لكل مشكلة برمجية يمكن كتابة أكثر من Pseudo Code صحيح وكذلك رسم أكثر من مخطط سير العمليات صحيح.
- ذ- (صحيحة)
- ر- (صحيحة)
3. في كافة الحلول لهذا السؤال، سيطلب منك رسم مخطط سير العمليات وكذلك كتابة الشيفرة الوصفية.
- (أ)

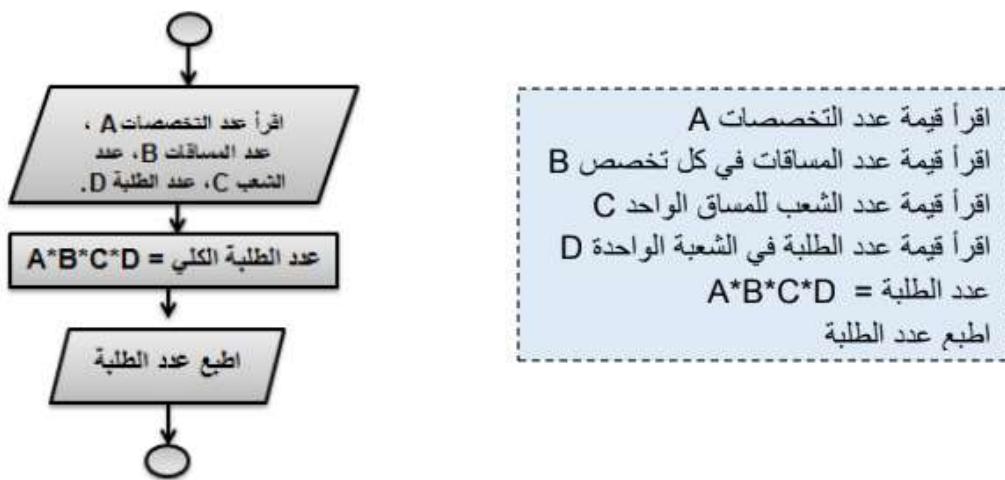


(ب)

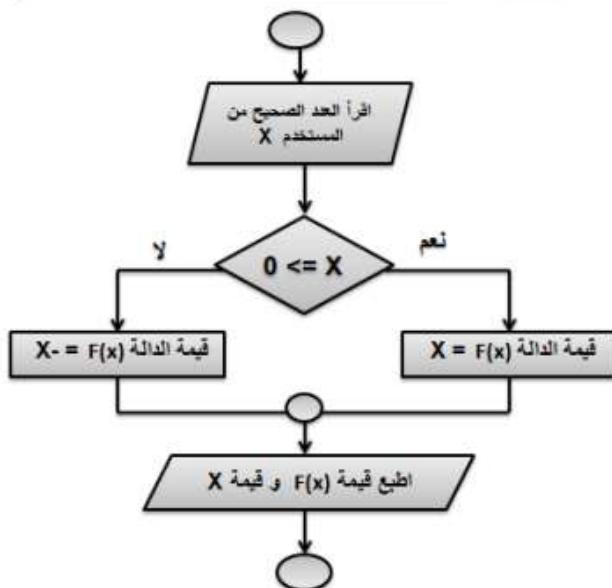




ث

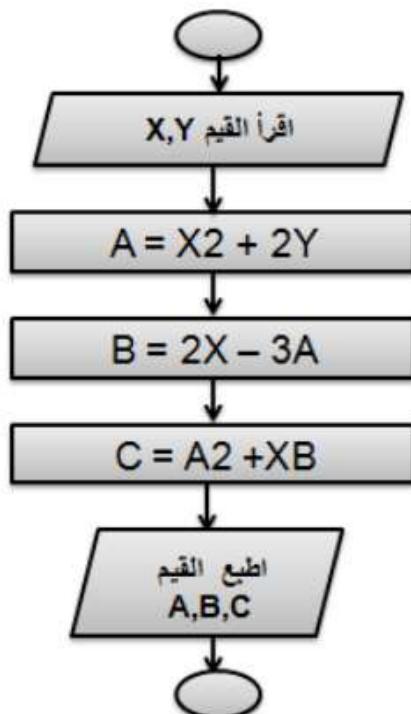


ث



اقرأ قيمة المتغير  $X$   
إذا كانت  $X$  أكبر من أو تساوي صفر  
قيمة الدالة  $F(x)$  تساوي  $X$   
وإلا  
قيمة الدالة  $F(x)$  تساوي  $-X$   
اطبع قيمة كل من  $X$ ،  $F(x)$ .

ج

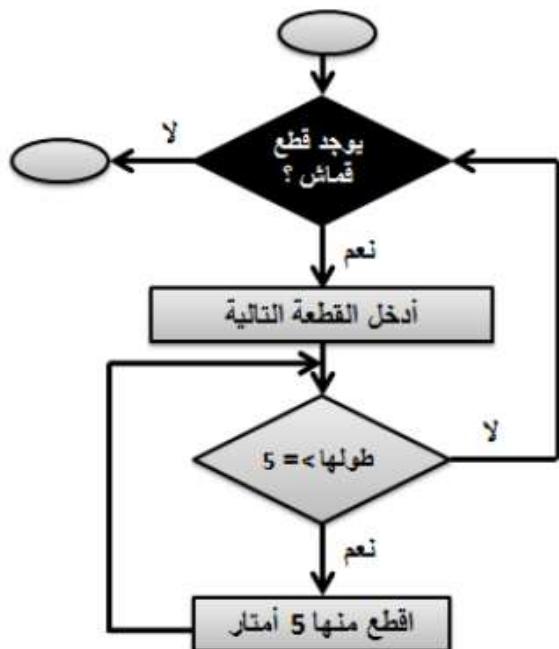


اقرأ قيمة المتغير  $X$   
اقرأ قيمة المتغير  $Y$   
احسب قيمة المتغير  $A$  من المعادلة:  

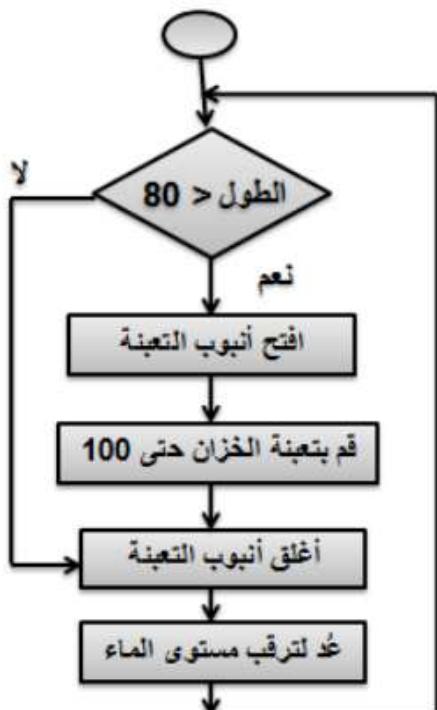
$$A = X^2 + 2Y$$
  
 قم بالتعويض بقيم  $X$ ،  $A$  في المعادلة  

$$B = 2X - 3A$$
  
 لحساب المتغير  $B$   
 قم بالتعويض بقيم  $X, A, B$  في المعادلة  

$$C = A^2 + XB$$
  
 لحساب المتغير  $C$

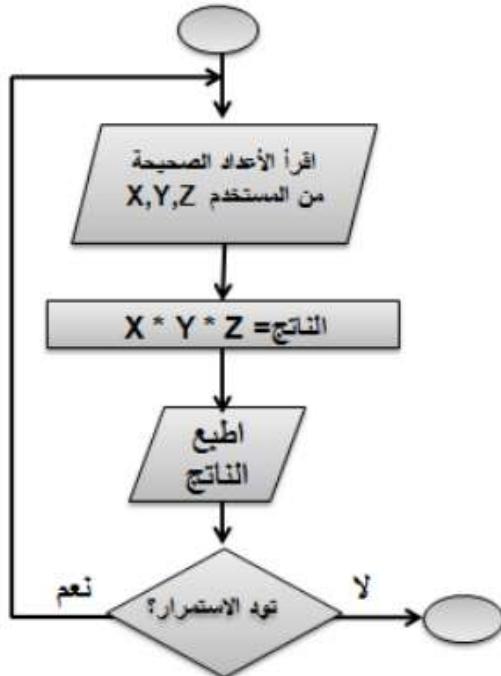


- 1 إن كان يوجد قطع قماش
- 2 خذ واحدة
- 3 هل طولها أكبر من أو يساوي 5 مترا
- 4 اقطع منها 5 مترا
- 5 عد لخطوة 3
- 6 وإن
- 7 عد لخطوة 2
- 8 وإن
- 9 انتهت الكمية من قطع

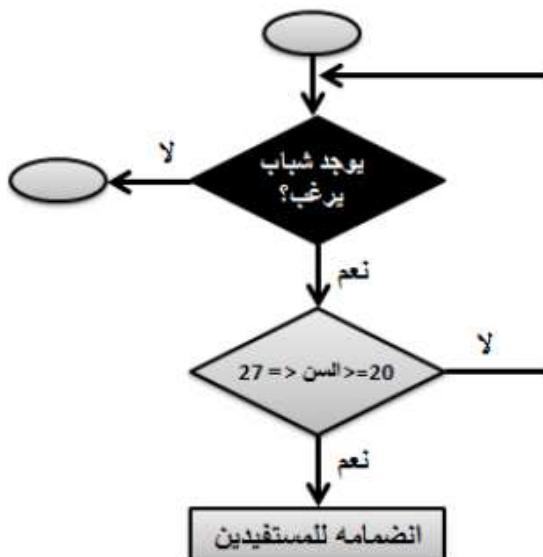


- 1 كرر الخطوات التالية دون توقف
- 2 إن كان ارتفاع الماء في الخزان أقل من 80 مترا
- 3 افتح أنبوب التعبئة
- 4 قم بتعبئنة الخزان حتى ارتفاع 100 مترا
- 5أغلق أنبوب التعبئة
- 6 عد لترقب مستوى الماء
- 7 وإن
- 8 حافظ على أنبوب التعبئة مغلقا

.5



.4



6. الهدف من هذه الشيفرة الوصفية: أن يستقبل النظام من المستخدم رقمًا صحيحًا ثم يقوم النظام بحساب مجموع الأعداد التي تسبق الرقم حتى الصفر، فإن كان الرقم المدخل من المستخدم هو 4 فإن الناتج يكون  $4 = 1+2+3+4$  وهكذا. إذا الناتج المتوقع في حال أدخل المستخدم الرقم 5 يكون طباعة القيمة 15

7. الهدف من مخطط سير العمليات 2.12 هو: حساب وطباعة مضروب الأعداد و هو ما يعرف باسم Factorial ويعامل مع الأعداد الصحيحة. والناتج المتوقع في حال أدخل المستخدم الرقم 5 هو طباعة العدد 120 لأنه يقوم بتنفيذ العملية الحسابية التالية  $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

انتبه إلى أنَّ أمر الطباعة هنا جاء خارج الدوران وبالتالي سُيطبع المضروب فقط مرة واحدة بعد انتهاء الدوران، و كن في حال تم وضع أمر الطباعة داخل الدوران فسيتم طباعة قيمة المضروب في كل مرة من مرات الدوران. (يعنى سيتم طباعة قيمته المعدلة كل دورة من الدورات)



انتهى الباب

الحياة هي مجموعة من المبادئ والقواعد .  
يطبقها الناس بدرجات إجاده مختلفة .

قال كارل ماركس : " لا شيء يمتلك قيمة دون أن يكون مشتق من قاعدة أصلية ."

## الباب الثالث

### أساسيات البرمجة (Principles of Programming)

يتوقع من الدارس في نهاية هذا الباب أن :

- يدرك الحاجة للغات البرمجة.
- يتعرف على مفهوم البرمجة.
- يتعرف على دور الحاسوب في حل المشاكل.
- يفهم كيفية انتقال مرحلة الحل من الإنسان للحاسوب.
- يعرف كيف يفهم الحاسوب كلام الإنسان.
- يتعرف على لغة الجافا.
- يعد ويفهم مكونات لغة جافا.
- يكتسب القدرة على تعريف المتغيرات والثوابت واستخدامها في حل المشاكل.
- يفهم عملية إدخال المعلومات للحاسوب وإخراج النتائج.
- يطور تطبيقات بسيطة بلغة جافا.

يتوفر لهذا الباب شرائح العرض (PowerPoint) وكذلك ملفات مرئية من إعداد مؤلف الكتاب؛ يمكنك الحصول عليها من خلال الموقع الأكاديمي للمؤلف [mfarra.cst.ps](http://mfarra.cst.ps)



## الباب الثالث: أساسيات البرمجة

### 3.1 مقدمة

هذا الباب هو المدخل الحقيقي للبرمجة بلغة جافا من حيث التعرف على مكوناتها الأساسية من متغيرات وثوابت وجمل الإدخال والإخراج وتنفيذ العمليات الحسابية، كذلك كيفية التعامل مع هذه المكونات والاستفادة منها في كتابة أول برنامج بلغة جافا.

وبالطبع لا يمكن لنا الحديث عن لغة جافا ومكوناتها قبل الحديث عن أساس وجودها وأصل انحدارها؛ فالحديث سيكون واضحًا في الفصل 3.3 عن تصنیفات لغات البرمجة وكيف تطورت لغات البرمجة بتطور علمها. وانطلاقاً من كون أنَّ الفهم هو الباب الرئيس للتعامل مع الأشياء من حولنا فقد خصصت الفصل 3.4 لمناقشة وتوضیح دور الحاسوب في حل المشاكل ذلك بعد أن ناقشنا في الباب السابق دور الإنسان في هذه المسألة.

### 3.2 الحاجة إلى البرمجة

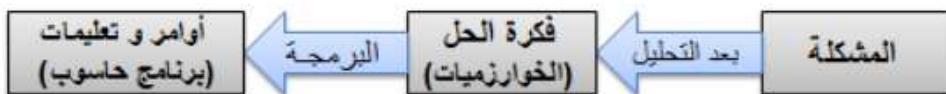
أوضحنا في الباب الأول أنَّ الحاسوب لا يمكنه العمل دون اجتماع طرفيه (الماديّات والبرمجيات)، وقد شرحنا بشيء من التفصيل بعض الأمثلة على الماديّات (*Hardware*) التي يتكون من الحاسوب، بينما على الطرف الآخر البرمجيات التي تستعمل ماديّات الحاسوب لمعالجة البيانات وإخراج نواتج المعالجة تُصنّف إلى:

- **أنظمة التشغيل المختلفة:** والتي تستخدم في إدارة مكونات الحاسوب المادية مثل (ويندوز، يونكس، لينكس).

- **البرامج التطبيقية:** والتي يتم تطويرها ل القيام بوظائف محددة يحتاج لها سوق العمل مثل (أنظمة المحاسبة في البنوك، أنظمة المعاملات الإدارية والأكاديمية في الجامعات والكليات، برامج تحرير النصوص والصور، البرامج الترفيهية من ألعاب وغير ذلك).

- **لغات البرمجة:** وهي اللغات التي يستخدمها المبرمجون للتواصل مع الحاسوب ودفعه لتنفيذ البرامج التي من شأنها القيام بمهام معينة، وقد شهدت هذه اللغات تطوراً واضحًا منذ نشأتها حتى تم تصنیفها إلى مجموعة تصنیفات سيتم مناقشتها في الفصل 3.3.

هذه البرمجيات المختلفة ما ظهرت إلا من أجل العمل على خدمة الإنسان في مجالات مختلفة من حيث السرعة ودقة العمل وزيادة الإنتاج كما أوضحنا ذلك في الفصل 1.5 عند الحديث عن مميزات الحوسبة، هذه الحوسبة التي لا تتم إلا من خلال مفهوم البرمجة هي عبارة عن عملية كتابة البرامج عن طريق كتابة التعليمات والأوامر التي تحكم في البيانات وذلك بهدف حل مشكلة معينة.



شكل 3.1: دور البرمجة في حل المشاكل البرمجية

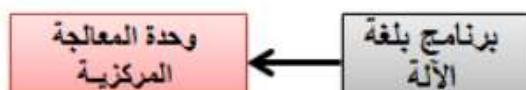
لهذا يمكنني القول أن الحاجة إلى البرمجة تكمن بقوّة في أنها هي الوسيلة التي ستمكننا من إنتاج كافّة البرمجيات المذكورة سابقاً والتي تساعدنَا في زيادة الحوسبة في مجتمعنا وبالتالي يزداد إنتاجنا في زداد مستوى حياتنا على مختلف النواحي. وطالما أصبحنا مدركين إلى حاجتنا للبرمجة، فإنّها لا يمكن الوصول لها دون ابتكار لغات يمكننا من خلالها مخاطبة الحاسوب لإرشاده إلى تنفيذ ما نريد حوصلته. هذه اللغات التي يختلف نوعها ومستوياتها بين لغات عالية المستوى وأخرى متدرّجة، فهي في نهاية الأمر تعتبر لغات يتحدث من خلالها المبرمج مع الحاسوب لتنفيذ الأوامر والتعليمات.

### 3.3 تصنیفات لغات البرمجة

المبرمجون (*Programmers*) - ويعرفون أيضاً بالمطوريين (*Developers*) - يكتبون التعليمات البرمجية بلغات مختلفة، بعضها يفهم مباشرةً من الحاسوب وبعض الآخر قد يحتاج إلى مراحل وسيطة من الترجمة بينها وبين الحاسوب. هذه اللغات تُقسم إلى ثلاثة مستويات من حيث درجة سهولة التعامل معها واستخدامها:

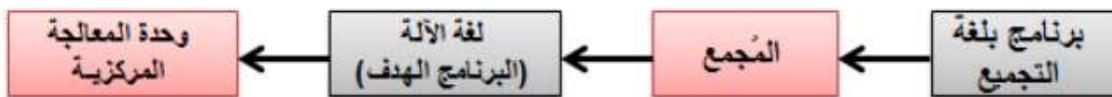
**3.3.1 لغة الآلة (*Machine language*):** هي اللغة الوحيدة التي يفهمها الحاسوب بلا وسيط فهي لغة يتم تعریفها بواسطة مادیات الحاسوب مما يجعلها تختلف من حاسوب آخر. وتكون من سلسلة من الأرقام مكونة من (0,1) لتمثيل التعليمات البرمجية للحاسوب، والتي تنفذ واحدة واحدة. ويعيّنها أنها صعبة التعامل والفهم من طرف المبرمجون مما يقلل من التعامل معها وبالتالي يقلل من الحوسبة، كما يعيّنها البطء في تنفيذ التعليمات. هذه العيوب دفعت المهتمين إلى التفكير في تطوير لغات برمجة مستواها أعلى تستخدم بعض ألفاظ الإنسان لتنفيذ تعليمات محددة وهي ما تعرف باسم لغة التجميع.

لغة الآلة تختلف من حاسوب آخر باختلاف الماديات التي تكون الحاسوب، فهي لا تعتبر لغة موحدة تعمل بها كافة الحواسيب وذلك عكس اللغات عالية المستوى كما سنتابع بعد قليل.



شكل 3.2: برامـج لغـة الآلة يتم تنفيذـها دون الحاجـة إلى وسيـط

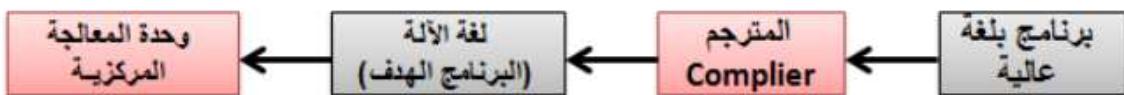
**3.3.2 لغة التجميع (Assembly languages):** هي لغة تستخدم مجموعة من الاختصارات باللغة الإنجليزية لتنفيذ بعض التعليمات الأساسية، وهذه الاختصارات تعتبر هي الأساس لهذه اللغة، وبوجودها أصبح المبرمج يستخدم كلمات مثل (*Store*) لتخزين قيمة داخل متغير وكلمة (*Add*) لجمع رقمين وهكذا. ولكن هذه الاختصارات أصبح يحتاج فهمها من طرف الحاسوب إلى وسيط يترجمها (يحولها) إلى لغة الآلة وهو ما يعرف باسم المُجمّع (*Assembler*)، ولفهم طبيعة عمله انظر الشكل التوضيحي 3.3.



شكل 3.3: رحلة برامج التجميع من إلى كتابة حتى تنفيذ التعليمات

هذه الألفاظ قللـت من صعوبة استخدام لغات البرمجة مما زاد من انتشار لغات البرمجة ومهنتها إلا أن سرعتها لا زالت أقل مما يتـنـاسب مع حاجة الإنسان لسرعة الإنتاج، كما أن المـبرمجـين يـحتاجـون إلى كتابة العـدـيد من التعليمـات البرـمـجيـة إلى كتابـة أبـسـط التطـبـيقـات مما يـزيدـ من الإـرـهـاقـ وإـضـاعـةـ الوقـتـ. هذا دفع المـبرمجـين ومع ازديـادـهـم للـبـحـثـ عن لـغـةـ أـسـرـعـ في إلى كتابـةـ والتـفـيـذـ وأـسـهـلـ في التـعـاملـ.

**3.3.3 لغات عالية المستوى (High Level language):** هي لغات تم تطويرها بحيث يتم استخدام العديد من مفردات اللغة الإنجليزية بمعانيها المـتـعـارـفـ عليها، وكذلك استخدام العمـلـيـاتـ الحـاسـيـةـ والـمـنـطـقـيـةـ بـمـعـانـيـهـاـ دونـ الحاجـةـ إلىـ تـعـرـيفـهاـ مـرـأـخـيـ،ـ مماـ زـادـ منـ سـهـولةـ التعـاـمـلـ معـهاـ وـسـرـعةـ تـفـيـذـهاـ.ـ بهـذـهـ الإـمـكـانـيـاتـ السـهـلـةـ أـصـبـحـ بـمـكـانـ المـبـرـمـجـينـ تـفـيـذـ العـدـيدـ منـ الـعـمـلـيـاتـ بـسـطـرـ بـرـمـجيـ واحدـ مـثـلـ كـافـةـ الـعـمـلـيـاتـ الحـاسـيـةـ.ـ إـلاـ أـنـهاـ تـحـاجـ إلىـ مـتـرـجـمـاتـ تـعـرـفـ باـسـمـ (Compilers)ـ لـتـحـوـيلـ هـذـهـ الـبـرـامـجـ لـلـغـةـ الـآـلـةـ كـمـاـ فيـ الشـكـلـ 3.4ـ.



شكل 3.4: رحلة برامج عالية المستوى من إلى كتابة حتى تنفيذ التعليمات

ومن الأمثلـةـ عـلـىـ لـغـاتـ البرـمـجـةـ عـالـيـةـ المـسـتـوـىـ لـغـةـ جـافـاـ (*Java*)ـ وـالـسـيـ شـارـبـ (*C#*)ـ وـفـيـجوـالـ بـيـزـكـ (*Visual Basic*)ـ وـسـيـ بـلـسـ بـلـسـ (*C++*)ـ وـغـيرـهـاـ.

برامـجـ التـرـجـمـةـ تـقـعـ أـهـمـيـتهاـ فـيـ تـرـجـمـةـ الـبـرـامـجـ مـنـ الـلـغـاتـ المـخـلـفـةـ إـلـىـ لـغـةـ الـآـلـةـ التـيـ يـفـهـمـهـاـ الـمـعـالـجـ بـسـهـولةـ وـمـنـهـاـ المـجـمـعـ،ـ الـمـتـرـجـمـ،ـ وـبـرـامـجـ التـفـسـيرـ الـفـورـيـ التـيـ تـعـملـ عـلـىـ اـكتـشـافـ الـأـخـطـاءـ أـوـلـاـ بـأـوـلـ عـنـ كـاتـبـةـ الـبـرـامـجـ عـالـيـةـ المـسـتـوـىـ



كما تلاحظ، فإن التعبير عن هذه العملية الحسابية البسيطة بلغة الآلة كلفنا كتابة ثلاثة أوامر غير مفهومة للإنسان العادي بينما بلغة التجميع كان الأمر أقل تعقيداً، ولكن في اللغات عالية المستوى سيتم التعبير عن المسألة بمنتهى السهولة كما سنتابع.

بعض التصنيفات تعتبر لغات البرمجة تنقسم إلى قسمين (لغات متقدمة) و(لغات عالية) وأن اللغات المتقدمة تتكون من لغة الآلة و لغة التجميع.



### مثال توضيحي 3.1:

هذا مثال لتوضيح كيفية كتابة برنامج يقوم بإضافة مقابل العمل الجزئي للأجرة الرئيسية باستخدام لغة الآلة، لغة التجميع، واللغات عالية المستوى.

ناتج الحل بلغة الآلة:

+1300042774  
+1400593419  
+1200274027

ناتج الحل بلغة التجميع:

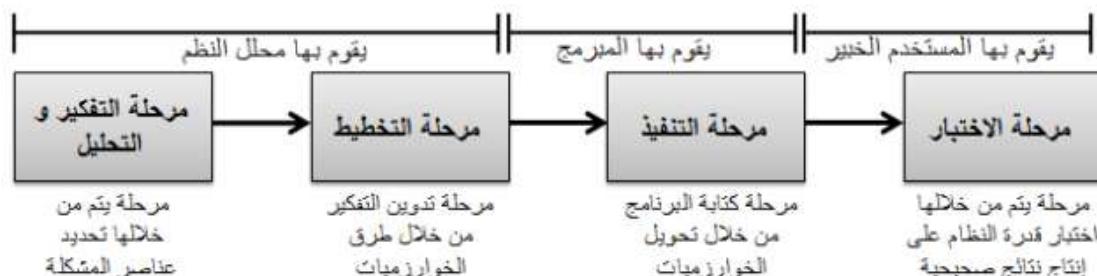
load base  
add overtime  
store totalpay

ناتج الحل بأي لغة من اللغات عالية المستوى:

**totalpay = base + overtime**

### 3.4 مراحل حل المشاكل باستخدام الحاسوب

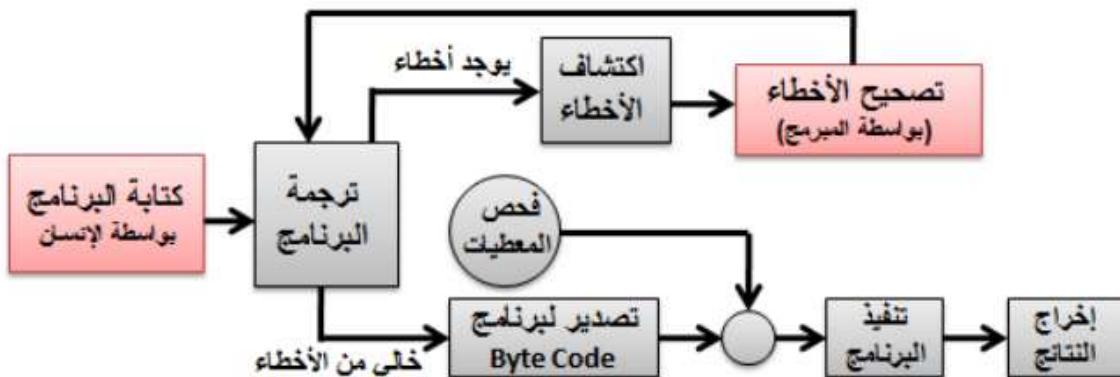
استعرضنا في الباب الثاني إلى الخطوات التي يقوم بها الإنسان ( محلل النظام والمبرمج، المستخدم الخبير ) من أجل حل المشكلة؛ بداية من مرحلة التفكير وحتى مرحلة الاختبار ، ويوضحها الشكل التوضيحي 3.5 وهو شكل مبسط لشكل 2.5 السابق ذكره في الباب الثاني.



شكل 3.5: دور الإنسان في حل المشاكل

بينما يظهر دور الحاسوب في حل المشاكل في مرحلة تظاهر بين مرحلة التنفيذ ومرحلة الاختبار، حيث أن المبرمج أثناء كتابته لتعليمات البرنامج من خلال إحدى لغات البرمجة يحتاج إلى تواصل مع المترجم لاستكشاف مدى صحة المتغيرات والأسماء والتركيب التي يستخدمها، كما يحاول المبرمج تنفيذ بعض خطوات الاختبار للنظام الذي يقوم بتطويره بعد انتهاءه من أجزاء منه ليتأكد من صحة أدائه.

دور الحاسوب في تطوير البرنامج وكيف يتم ترجمته وتحويله إلى ملفات بصيغ أخرى يفهمها الحاسوب موضح في الشكل التوضيحي 3.6.



شكل 3.6: دور الحاسوب في حل المشكلة بلغة جافا

كما يتضح من الشكل 3.5، فإن دور الحاسوب في حل المشكلة يبدأ بمجرد أن ينتهي المبرمج (الإنسان) من كتابة البرنامج بعد تحويل الخوارزمية، بعدها يبدأ المبرمج في عملية ترجمة البرنامج ليرى هل يوجد أخطاء في عمله أم لا، فإن كان هناك أخطاء يكتشفها المترجم ويظهرها في قائمة تعرف باسم قائمة الأخطاء ويقوم المبرمج بتصحيحها واحداً تلو الآخر، وفي حال لم يكتشف المترجم أخطاء يتم تحويل البرنامج إلى ملف من نوع (ByteCode) وهو صيغة ملف تتمكن لغة جافا من تنفيذه عبر وحدة المعالجة المركزية التي تقوم بفحص المعطيات (Data Checking) للتأكد من صحة كافة البيانات، بعدها يتم تنفيذ البرنامج وإخراج البيانات.

هذه العملية كاملة لا يشعر بها المبرمج إلا عندما تظهر الأخطاء أو تظهر النتائج أما العمليات الداخلية فهي تتم كاملة من خلال الحاسوب دون تدخل المبرمج.

**توضيح:** التعبير بكلمة النظام أو البرنامج في هذا الكتاب يؤدي إلى المعنى ذاته، فكلهما يؤدي إلى مفهوم حوسبة عملية ما. بينما يتم التفريق بينهما في أن النظام يضم أكثر من برنامج.



### 3.5 مميزات لغة جافا

تتميز لغة جافا بسهولة القاء معها ومع مكوناتها، كما أنها من اللغات التي تمتلك عدداً من التطبيقات المختلفة مثل تطبيقات الويب وتطبيقات سطح المكتب، كما يزيد من تميزها سهولة ربطها مع الأنظمة وقواعد البيانات بشكل يساعد في بناء أنظمة كبيرة ومعقدة تواكب المؤسسات والوزارات وأنظمة إدارة المعلومات حول العالم. ويعتبر من أهم مميزات هذه اللغة أنها تمتلك الآلة الافتراضية (*Java Virtual Machine "JVM"*) مما يجعل كل أنظمة التشغيل قادرة على تشغيل تطبيقاتها.

البرامـج التي يمكن إنتاجها بلغـة جـافـا تـنقـسـم إـلـى بـرـامـج تـطـبـيقـيـة تـعـمل عـلـى الأـجـهـزـة بـعـد تـنـصـيبـها و توـفـيرـ الـبـيـنـة الـلاـزـمـة لـهـا (مـوـضـوـعـ الـكـتـابـ)، و بـرـامـج تـعـرـفـ باـسـم Applet وـهـي بـرـامـج تـعـمل مـن خـلـال مـسـتـكـشـفـاتـ الإنـتـرـنـتـ وبـالـتـالـي يـمـكـن تـنـفـيـذـها عـلـى أيـ جـهـازـ.



### 3.6 مكونات لغة جافا

لغات البرمجة متـلـ لـغـاتـ البـشـرـ المعـرـوفـةـ لـهـاـ مـجـمـوعـةـ مـوـضـوـعـاتـ الـمـكـونـاتـ الـتـيـ يـتـمـ اـسـتـخـادـهـاـ بـتـرـكـيـبـ معـيـنـ لإـيـصالـ فـكـرـةـ ماـ لـلـغـيـرـ أوـ لـإـنشـاءـ حـوـارـ مـفـيدـ مـعـ الـآخـرـينـ، وـهـكـذـاـ لـغـةـ جـافـاـ لـهـاـ مـكـونـاتـ يـتـمـ اـسـتـخـادـهـاـ لـبـنـاءـ الـبـرـامـجـ.

إنـ أيـ بـرـامـجـ فـيـ لـغـاتـ الـبـرـمـجـةـ وـمـنـهـاـ جـافـاـ يـتـكـونـ مـنـ مـجـمـوعـةـ تـعـلـيمـاتـ (*Instructions*) تـنـقـسـمـ إـلـىـ أـرـبـعـةـ أـنـوـاعـ هـيـ:

- (1) تعليمات لقراءة البيانات من وحدات الإدخال.
- (2) تعليمات لتخزين البيانات في الذاكرة الرئيسية أو الثانوية.
- (3) تعليمات لإجراء العمليات الحسابية والمنطقية على البيانات.
- (4) تعليمات لإخراج البيانات على وحدات الإخراج.

وبـالـتـالـيـ فـيـ كـاتـبـ الـبـرـامـجـ يـتـطـلـبـ مـنـ مـعـرـفـةـ الـمـكـونـاتـ الـخـاصـةـ بـهـاـ سـوـاءـ لـتـخـزـنـ الـبـيـانـاتـ بـتـرـاكـيـبـ مـخـتـلـفـةـ أوـ لـإـدـخـالـ الـقـيـمـ مـنـ خـلـالـ الـمـصـادـرـ الـمـخـتـلـفـةـ وـمـنـهـاـ الـمـسـتـخـدـمـ، كـذـلـكـ كـيـفـيـةـ تـنـفـيـذـ الـعـلـمـيـاتـ الـحـاسـابـيـةـ وـالـمـنـطـقـيـةـ عـلـيـهـاـ وـهـذـاـ مـاـ سـنـتـعـاملـ مـعـهـ خـلـالـ هـذـاـ كـاتـبـ شـيـئـاـ فـشـيـئـاـ، حـيـثـ سـنـتـعـرـفـ فـيـ الـفـصـولـ الـقـادـمـةـ مـنـ هـذـاـ الـبـابـ عـلـىـ كـيـفـيـةـ اـسـتـقـبـالـ الـبـيـانـاتـ مـنـ الـمـسـتـخـدـمـ وـكـيـفـيـةـ تـخـزـنـهـاـ وـإـجـراءـ الـعـلـمـيـاتـ الـحـاسـابـيـةـ عـلـيـهـاـ ثـمـ كـيـفـيـةـ طـبـاعـةـ النـتـائـجـ.

### 3.6.1 أنواع البيانات ومفاهيم الذاكرة:

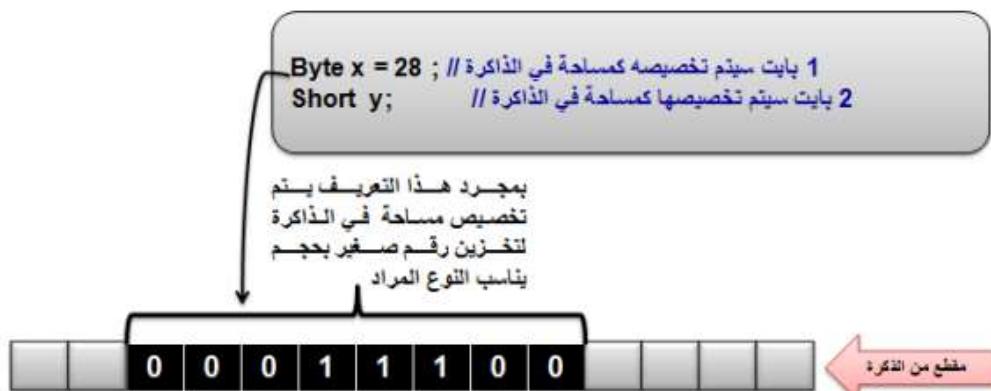
عند مناقشتنا لمفهوم البرمجة في الحاسوب، أوضحنا أنها معالجة لمجموعة بيانات، هذه البيانات لها أنواع وتُخزن في الذاكرة بصورة محددة يقبلها الحاسوب ويستطيع التعامل معها واسترجاعها حسب الحاجة.

أنواع البيانات (**Data Types**) في لغة جافا نوعان هما:

1) الأنواع الأساسية (ستتعرف عليها في الفصل الحالي).

2) الأنواع المشتقة (ستتعرف عليها في الباب التاسع).

الأنواع الأساسية هي التي تتكون من أنواع مفردة متعارف عليها في الحياة مثل الأرقام الصحيحة، العشرية، المنطقية وغير ذلك. كما أنها عند التخزين في الذاكرة لا يقبل موقعها في الذاكرة إلا قيمة واحدة في الوقت الواحد. جدول 3.1 يستعرض الأنواع الأساسية في لغة جافا، كما يستعرض الشكل 3.7 كيفية تخزين البيانات في الذاكرة من حيث حجز مكان في الذاكرة يتساوى مع الحجم المسموح لنوعها سواء بait أو أكثر.



شكل 3.7: مفهوم حجز الذاكرة المكافأة لأنواع البيانات

وبالنظر إلى شكل 3.7، نجد أن المتغير بمجرد أن يتم تعريفه (تحديد النوع الذي ينتمي له) يتم حجز مكان له في الذاكرة حجمه يساوي الحجم الافتراضي لهذا النوع، فعندما يتم تعريف متغير من نوع (**int**) يتم حجز 4 بait في الذاكرة لهذا المتغير وإن لم نستخدم للرقم سوى بait واحد فقط، لهذا نجد أن جافا تحتوي على أكثر من نوع لتمثيل العدد الصحيح أو العشري ولكن أحجامهم تختلف حتى تتم المحافظة على الذاكرة ولا تستخدم دون ترشيد خاصة عند بناء الأنظمة الكبيرة.

هذه البيانات عندما تُخزن في الذاكرة يتم تحديد مكانها من خلال عنوان يستخدم النظام السادس عشر، ولصعوبة التعامل معه وحفظ العناوين به، فإن جملة تعريف المتغيرات تجعل من السهل على المبرمج التعامل مع مواضع المتغيرات وقيمهم.

نوع البيانات	البيانات التي يمكن تمثيلها به	الحجم	مثال عليه
boolean	القيمة المنطقية (false, true) وليس له قيمة أخرى. ويستخدم في فحص قيمة الشرط المنطقية.	(1 بت)	المرتب = 1500 إذا كان (المرتب > 1000) هذا الشرط قيمته تكون true أما إذا المرتب = 700 ف تكون قيمته false.
char	نوع لتمثيل الحروف.	2 بait (16 بت)	مثل L, H, K, \$, #, <, >, ! ...
byte	نوع يمثل الأعداد الصحيحة الصغيرة جدا ذات الإشارة.	1 بait (8 بت)	مثل 3, 28, 87, 23, 34, ...
short	نوع يمثل الأعداد الصحيحة الصغيرة ذات الإشارة.	2 بait (16 بت)	مثل 7, 487, 32, 393, 1, ...
int	نوع يمثل الأعداد الصحيحة الكبيرة ذات الإشارة.	4 بait (32 بت)	مثل 11, 2131, 23, 311, ... , 312, -2131, -23, -311, ...
long	نوع يمثل الأعداد الصحيحة الكبيرة جداً.	8 بait (64 بت)	مثل 4, 4424234, 423, -938, ...
float	نوع يمثل الأعداد العشرية الكبيرة ذات الإشارة.	4 بait (32 بت)	مثل 23.0, 32.32, 29.45
double	نوع يمثل الأعداد العشرية الكبيرة جداً.	8 بait (64 بت)	مثل 123.32, -3234.45345

جدول 3.1: ملخص لأنواع البيانات الأساسية في لغة جافا

### 3.6.2 تعريف المتغيرات (*Variables Declaration*) :

ويتم تعريف المتغيرات من خلال جملة تعريف تحتوي على كلٍ من:

- **اسم المتغير (Identifier):** وهو اسم يغنينا عن التعامل مع عناوين الذاكرة الصعبة ويعيننا عن تنظيم مساحة الذاكرة وسنعرف على شروط تسميتها وما يسمح به في هذا الفصل لاحقاً تحت عنوان الاسم التعريفي.
- **نوع البيانات (Data type):** وهو أحد الأنواع السابقة بجدول 3.1 أو الأنواع المشتقة وذلك ليتم تخصيص المساحة المناسبة للمتغير.
- **الفاصلة المنقطة (Semi colon):** وهي إشارة لا بد أن تنتهي بها كل جملة (تعليمية) برمجة في لغة جافا ونسأتم لتمثيلها الرمز (:) .

وأمثلة ذلك ما يلي:

1. تعريف متغير لتخزين الأرقام المتناسبة للطلبة وعدهم 30 طالب:

**byte no;**

2. انتبه إلى أن الأرقام المتسلسة في هذه الحالة يكفيها النوع (*Byte*) لأنّه يمكن تمثيل الرقم 30 ببایت واحد.

3. تعريف متغير لتخزين ناتج عمليات حساب قيمة الضريبة المخصومة من مرتبات الموظفين وأقصى مرتب هو 5000 دينار، علما بأنّ نسبة الضريبة 17% ستجد غالباً أن الناتج سيحتوي على أعداد عشرية صغيرة وبالتالي أنساب نوع يمكننا استخدامه هو (*float*).

**float tax;**

ويمكن إعطاء قيم للمتغيرات من خلال علامة المساواة (=) كما يلي:

**int size= 5;**

واعطاء القيم للمتغيرات من نوع (*float*) لابد أن يليها الحرف f، بينما القيم لنوع (*char*) لابد أن توضع بين علامات تصيص فردية كما يلي:

**لغة الجافا تعتبر لغة حساسة لحالة الأحرف (Sensitive)** وهذا يعني أن الكلمات بحروف كبيرة لا تكفي الكلمات بحروف صغيرة فمثلا *Float* لا يساوي *float*، وهذه القاعدة تتطبق على كافة أسماء المتغيرات ومكونات لغة جافا.

**float tax = 3.5f;****char truth = 'T';****double = 4.532;****3.6.3 الاسم التعريفي (Identifier) وقواعد:**

أسماء المتغيرات وما يُعرف بالكائنات والأصناف (سيتم التعرف عليها في الباب التاسع) وغيرهم يسمى اسم تعريفي حيث أنه يُعرف كائن ما في البرنامج بوظيفة محددة، وتخضع تسميته لمجموعة قواعد تُعرف بقواعد الاسم التعريفي (*Identifier*) وهذه الضوابط هي:

1. يتكون من الحروف الإنجليزية (A ... Z) و(z ... a) والأرقام (0 ... 9) والرموز \$ و \_ فقط.
2. أن يبدأ الاسم فقط بحرف.

3. لا يحتوي على رموز خاصة غير `$` و `_` ولا يحتوي فراغات.

4. لا يكون من الأسماء الممحوزة (سيتم التعرف عليها لاحقاً في 3.6.4).

وبناءً على ذلك، فإن الجدول 3.2 يحتوي في عموده الأيمن على مجموعة من الأسماء التعريفية الصحيحة بينما العمود الأيسر يحتوي أسماء تعريفية خاطئة.

أمثلة أسماء خاطئة	أمثلة أسماء صحيحة
3seer	Size
length	X_hor
المساحة	Y\$_play
\$4all	salesBy\$
Fname&Lname	S4\$_Taxi
B*	Salary32\$
Farra^lala	A_231_\$

جدول 3.2: أمثلة بعض الأسماء التعريفية الصحيحة والخاطئة

### 3.6.4 الأسماء الممحوزة (*Reserved words*)

لغة البرمجة تحجز مجموعة من الكلمات تُستخدم لأداء وظيفة ومهمة محددة (مثل أسماء أنواع البيانات) وهي وبالتالي لا تعتبر متاحة لاستخدامها من خلال المبرمج لأداء وظائف أخرى، لأنّ يُستخدم المبرمج كلمة `int` كاسم لمتغير أو ما شابه ذلك، الجدول 3.3 يضم كافة الكلمات الممحوزة في لغة جافا.

abstract	Assert	Boolean	Break	Byte
case	Catch	Char	Class	Continue
default	Do	Double	Else	Enum
extends	Final	Finally	Float	For
if	implements	Import	Instanceof	Int
interface	Long	Native	New	Package
private	Protected	Public	Return	Short
static	Strictfp	Super	Switch	Synchronized
this	Throw	Throws	Transient	Try
while	Volatile	Void	Const	Goto

جدول 3.3: الكلمات الممحوزة في لغة جافا

### 3.6.5 علامات الترقيم (Punctuators)

المعروف عند كتابة أي لغة بشرية أو برمجية الحاجة إلى استخدام علامات الترقيم (Punctuators) وذلك لتنظيم فهم الجمل بشكل صحيح من حيث البداية والنهاية للجمل والقطع وفصل الجمل عن بعضها حتى وإن كتبت في سطر واحد. جدول 3.4 يوضح أنواع علامات الترقيم السبعة مع توضيح مفهوم كل واحدة منها. قد لا يوضح هذا الجدول بعض هذه العلامات بصورة كبيرة الآن، إلا أنها ستصبح أكثر وضوحاً من خلال الأبواب والفصول القادمة ومن خلال الأمثلة البرمجية القادمة فلا تشعر بالقلق.

العلامة	رمزها	مفهومها
الفاصلة المنقوطة	;	تستخدم لإنتهاء الجمل (الأوامر) البرمجية.
أقواس لتحديد الجمل المجمعة	{ }	تحتاج خلال كتابة البرامج إلى تحديد الترابط بين مجموعة جمل بحيث تنفذ سوياً أو تترك سوياً، هذه الأقواس تستخدم لتحديد بداية ونهاية هذه المجموعة.
الفراغات		تستخدم للفصل بين الكلمات، فوجود الفراغ يعني انتهاء كلمه وببداية كلمة جديدة.
أقواس القيم الممررة	( )	عند تمرير قيم من دالة لأخرى، يتم تمريرها بوضعها داخل هذه الأقواس. كذلك عند ربط شروط منطقة مع بعضها البعض.
علامات تنصيص زوجية double quotations	" "	تستخدم هذه العلامات لتحديد النصوص داخل البرنامج، مثل "Mahmoud Alfarrar"
علامات تنصيص فردية single quotations	' '	تستخدم لتخزين حروف داخل النوع char مثل ذلك: 'ا', 'H' .

جدول 3.4: ملخص علامات الترقيم المستخدمة في لغة Java

### 3.6.6 بيانات ثابتة (Constants)

هي عبارة عن قيم ثابتة تستخدم في البرامج لأغراض يرتديها المبرمج، والمقصود بثباتها أنها لا تتغير أثناء عمل البرنامج بل لابد من التعديل في بنية البرنامج وتعديل التعليمات البرمجية ثم إعادة بناء البرنامج وعمله. وقد تكون البيانات الثابتة أحد ثلاثة أنواع هي الأعداد الصحيحة، الأعداد العشرية وإما حروف مثل 'b'، '5'. كما يمكن تعريف متغيرات من النوع الثابت من خلال وضع الكلمة المحجوزة (final) قبل نوع المتغير كما يلي وهي متغيرات لا يمكن تغيير قيمتها في البرنامج:

```
final int size = 10;
```

**خطأ برمجي:** محاولة تعديل قيمة متغير بعد تعريفه ثابتًا يعتبر خطأ برمجي يمنع تنفيذ البرنامج.



أنواع وتفصيل الأخطاء التي قد يتعرض لها المبرمج سيتناولها الكتاب بالشرح في الباب الخامس.



### 3.6.7 الثوابت النصية (*Strings*):

باختصار يمكن تعريف الثوابت النصية على أنها النصوص التي نود من المترجم أن يطبعها كما هي أو أن يعطيها المستخدم للمترجم ليخزنها في الذاكرة لمعالجتها بعد ذلك. وتمثل النصوص على أنها مجموعة من الأحرف المتتالية التي تستخدم للتعبير عن جملة معينة ويتم وضعها بين علامتي التصيص حتى لا يخلط المترجم بينها وبين الأمر البرمجية الأخرى. وهذه النص قد يكتب بأي لغة وليس شرطاً اللغة الإنجليزية أو لغة البرمجة، ومثالاً عليها: " المسلم من سلم المسلمين من لسانه ويده " وكذلك "Mahmoud Rafeek Alfarra" و"0597393999" ونحوها.

وهذا لا يشمل استقبال النصوص من المستخدمين خلال عمل البرنامج حيث أن هذا يحتاج إلى استخدام متغيرات من نوع يعرف باسم (*String*) وهو من الأنواع المشتقّة التي سنناقشها لاحقاً.

### 3.6.7 جمل الإخراج:

تحتوي لغة جافا على إمكانية أن يُخرج البرنامج مجموعة من الرسائل المستخدم مثل (النتائج، الرسائل الترحيبية، الرسائل الختامية، الجمل التفاعلية، الإرشادية). هذه الإمكانية تتم من خلال أوامر الإخراج وتكتب في البرامج التطبيقية البسيطة على النحو التالي:

```
System.out.println("Hello World!");
```

هذا الأمر البرمجي يمكننا توضيحه بتفصيل يقدر مناسب لوقت الحالى لتجد أن الكلمة (*println*) هي عبارة عن دالة للطباعة، وتعتبر من الدوال التي توفرها لغة جافا، هذه الدالة تقوم بطباعة قيمة مختلفة على شاشة الإخراج، وأشهر هذه القيم هي الثوابت النصية التي تجدها بين علامات التصيص "Hello World!"، "Hello World!"،

فكم تتبع النص تم وضعه بين علامتي تصصيص، بينما تم استخدام الأقواس () لتمرير هذا النص كاملاً عبر الدالة (print) للمترجم. هناك دالة أخرى للطباعة هي الدالة (println) وتستخدم بالصورة ذاتها كما ترى.

```
System.out.print("Hello World!");
```

والفارق بين الدالتين أن الأولى عندما تطبع القيم الممررة لها، فإن المؤشر على شاشة الإخراج ينتقل للسطر التالي بينما عند استخدام الثانية فإن المؤشر يقف مباشرة بعد القيم الممررة، وليتضح هذا المفهوم وكافة المفاهيم السابقة أطرح لك المثال التوضيحي 3.1 وهو البرنامج الأول بلغة جافا.

من أبرز المشاكل التي تواجه الدارسين الجدد للبرمجة، عدم تركيزهم في كتابة الأوامر بصورة صحيحة سواء ترتيبها أو من حيث الهجاء.



### مثال توضيحي 3.1:

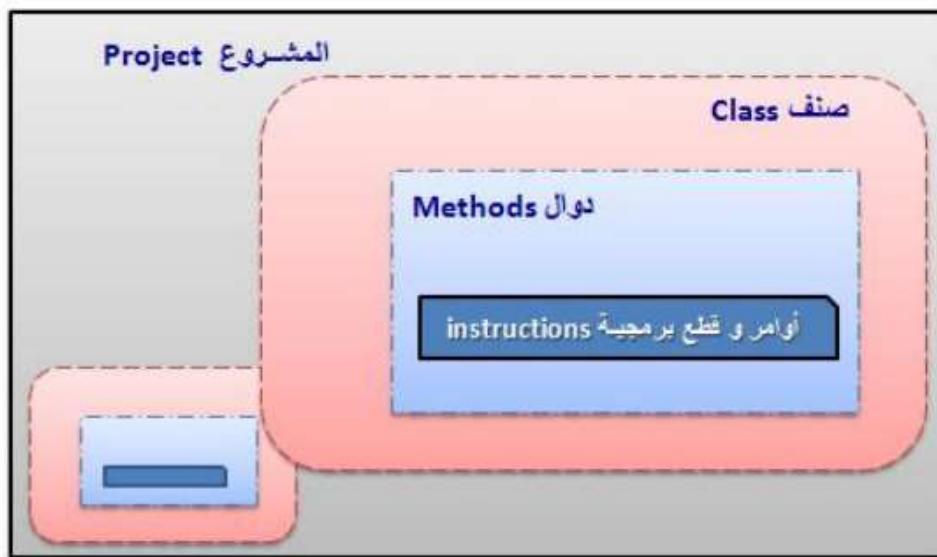
اكتب برنامجاً بلغة جافا ليطبع جملة (Hello World!)

```

1  /** @(#)wel.java
2  * wel application
3  * @author Mahmoud Rafeek Alfarra
4  * @version 1.00 2011/2/11
5  */
6  public class wel {
7  public static void main(String[ ] args) {
8      // TODO: add your application code
9      System.out.print("Hello World!");
10 }
11 }
```

هذا هو برنامجك الأول بلغة جافا، وهو البرنامج الافتراضي الذي يظهر لك بمجرد إتمام الخطوات التحضيرية لإنشاء البرامج في لغة جافا. وقبل أن نبدأ في شرح مكونات هذا البرنامج، دعنا من خلال الشكل

التوضيحي 3.8 نستعرض البناء العام للبرامج في لغة جافا، وكيف يتم تقسيمها وعملها، لأنّ هذا سيساعدك على الفهم بصورة أفضل.



شكل 3.8: تقسيم البرامج في لغة جافا

البرنامج في لغة جافا يُسمى (*Project*) وهو يضم بداخله على الأقل صنف واحد، الذي يضم بداخله مجموعة دوال تُنفذ مهام متخصصة من خلال احتواها على تعليمات برمجية. فعند إنشاء برنامج بلغة جافا، فإنّ التعليمات والأوامر في حقيقة الأمر تُكتب داخل الدوال بشكل كبير مع كتابة بعضها بشكل استثنائي خارجها.

الأسطر من (1 إلى 5) في بداية البرنامج تُسمى التعليق وهي عبارة عن مجموعة أسطر تحمل معلومات تعريفية حول البرنامج وكانتبه، والمقصود بالتعليق أنه كلام يستخدم كوسيلة لإضاح لا يتم ترجمتها من خلال المترجم بل يتم تجاهلها تماماً، ويمكن كتابة التعليق بطريقتين على النحو التالي:

- تعليق لسطر واحد: عندما نريد كتابة تعليق لسطر واحد نستخدم الإشارة // ونضع بعدها الكلام، ومثال على هذا النوع السطر رقم 8 في المثال 3.1.
- تعليق لأكثر من سطر: عندما نريد كتابة تعليق يمتد لأكثر من سطر ونستخدم في هذه الحالة الإشارة /\* في بداية التعليق ، ثم نكتب التعليق كاملاً كما نريد ونختم في النهاية \*/، ومثال ذلك الأسطر من 1 إلى 5.

وستخدم التعليقات دائمًا للتوضيح حول المراد من تعريف المتغيرات والكائنات المختلفة في المشاريع لكي يصبح فهمنا بعد ذلك يسيراً سواء للمبرمج ذاته أو عندما يتم تمرير البرنامج لمبرمج آخر ليكمل العمل أو يراجعه.

بناءً على ما سبق فإن التعليقات تعتبر شيئاً اختيارياً للمبرمج إن أراد استخدامه وإن أراد تركه إلا أننا ننصح دائمًا باستخدامه لشرح الغرض من كافة أجزاء البرنامج المهمة، حتى يسهل فهمه والتعامل معه.

**تنبيه مفید:** اللون الافتراضي الذي يظهر به التعليق في المحرر (*JCreator*) هو اللون الأخضر.



**تنبيه مفید:** الأرقام التي تظهر على يسار البرنامج، ليست من تعليمات البرمجة بل هي أرقام وضعتها فقط لتسهيل الإشارة إلى التعليمات والأوامر أثناء الشرح.



في السطر رقم 5 يبدأ البرنامج بشكل حقيقي من خلال تعريف صنف اسمه (*wel*)، فكل برنامج في لغة Java لا بد أن يحتوي على الأقل على تعريف صنف (*Class*) واحد يكون هو الصنف الرئيسي في البرنامج، يتم اختيار اسمه أثناء إنشاء المشروع، فأنت لا تقوم بكتابته هذا التعريف كاملاً بل يتم كتابته من خلال المحرر ولكن إن أردت كتابته فهو يتم تعريفه كما يلى:

```
public class wel
```

فيبدأ التعريف بكلمة (*public*) وهي أحد محددات الوصول في لغة Java ويقصد بذلك تحديد من وحدات المشروع (البرنامج) يمكنه مشاهدة والتفاعل مع هذا الصنف، وطالما كان (*public*) فإن الجميع يمكنه مشاهدة والتفاعل مع محتويات هذا الصنف (مزيداً حول هذا الموضوع سيتم مناقشته في باب البرمجة شيئاً التوجه). يتبع هذا المحدد كلمة (*class*) وهي أحد الكلمات المحفوظة التي تشير إلى وظيفة محددة من قبل المترجم، ثم يتبع ذلك كلمة (*wel*) وهو الاسم التعريفي للصنف، ومن المتعارف عليه أن الاسم التعريفي للصنف يبدأ بحرف كبير، ولكي تعمدت كتابته بالحرف الصغير لتعلم أن هذا لا يعتبر خطأً. هذا الصنف يحتوي مجموعة من الأوامر والكائنات البرمجية وجميعها تعمل ضمن هذا الصنف وبالتالي تستخدم الأقواس {} لتضم كافة هذه الأوامر والكائنات البرمجية. عليك أن تعرف أن أي تعليمات برمجية تكتب خارج هذه الأقسام يسبب عدم عمل البرنامج وظهور أخطاء في التنفيذ.

**خطأ برمجي:** كتابة تعليمات وأوامر برمجية خارج أقواس {} الأصناف يسبب خطأ برمجي يمنع البرنامج من العمل أو التنفيذ.



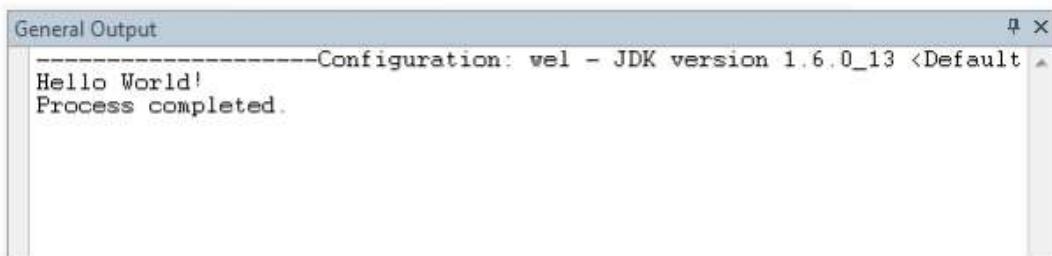
في السطر رقم 7، ستجد تعريف الدالة الرئيسية في الصنف الرئيسي، فكما أوضحتنا قبل قليل كل برنامج بداخله على الأقل صنف واحد دالة واحدة، والدالة الرئيسية للبرنامج هي الدالة (`main`) يتم تعريفها كما يلي:

```
public static void main(String[ ] args) {
```

البداية بـ (`public`) ثم الكلمتين (`void`, `static`) وهما كلمتان ممحوزتان سيتم التطرق لهما في باب البرمجة شيئاً فشيئاً التوجّه، ثم يتم كتابة الاسم التعريفي للدالة وهو (`main`) وهو من الكلمات الممحوزة، بعد ذلك تجد أن الدالة كما شرحنا قبل قليل لها أقواس القيم الممرة ( ) وتحتوي على قيم تمرر لاتجاه ما داخل البرنامج من أجل عمل الدالة، ثم استخدمنا القوس { لأننا بقصد كتابة مجموعة تعليمات وأوامر برمجية جميعها تابعة للدالة (`main`) وسيتم استخدام القوس { لاغلاق المجموعة بعد ذلك إلى هنا، لا يهمني في هذه المرحلة منك أن تعرف أكثر من ذلك.

في السطر رقم 8 يظهر تعليق يرشدك إلى أنك يمكنك كتابة التعليمات التي تريدها داخل هذه الدالة وهو تعليق سيتم تجاهله على كل حال من المترجم، بينما في السطر رقم 9 تجد التعليمية البرمجة الأولى في البرنامج والتي تهدف لطباعة جملة (`Hello World!`) باستخدام دالة الطباعة (`println`) والتي تم وضع الجمل المراد طباعتها بداخل الأقواس () وهي خاصة بالدالة ثم استخدام علامات التصنيص الزوجية لتحديد النص المراد طباعته، ثم انتبه إلى وضع الفاصلة المنقوطة (;) في نهاية الأمر البرمجي للإشارة إلى نهاية الأمر. في السطر رقم 10 يظهر القوس الذي ينهي التعليمات التابعة للدالة (`main`) ثم في السطر 11 يظهر القوس الذي ينهي الدوال التابعة للصنف (`(wel)`).

هذا البرنامج ستنظر نتائجه على شاشة الإخراج التي من النوع (`Console`) لتطبع فقط القيمة الموجودة داخل جملة الإخراج وهي (`Hello World!`) بالطريقة ذاتها التي كتبت بها وبذات حالة الأحرف وعدد الفراغات دون أي تغيير كما في شكل توضيحي 3.9.



شكل 3.9: ناتج برنامج مثال 3.1

دعنا الآن نحاول متابعة الفارق بين استخدام الدالة `(print)` و `(println)` من خلال المثال التوضيحي 3.2.

### مثال توضيحي 3.2:

مستخدماً أربع جمل إخراج (طباعة) اكتب برنامجاً يظهر اسمك الأول والثاني في سطر بينما يظهر صفحتك الإلكترونية بذلك في سطر آخر.

```

1 public class Print_Println {
2
3     public static void main(String[ ] args) {
4
5         System.out.print("Fname: Mahmoud");
6         System.out.println("Sname: Rafeek");
7         System.out.print("Site: Staff.cst.ps/mfarra");
8         System.out.println("Country: Palestine");
9     }
10 }
```

كما تتابع هذا المثال يطلب استخدام أربع جمل طباعة أربع معلومات ولكن على أن تظهر في سطرين فقط وهذا يعني أننا سنستخدم أمر الطباعة `(print)` والأمر `(println)` في مواضع محددة.

الأسطر 1 و 3 تم التطرق لهم سابقاً مع الانتباه إلى اسم البرنامج `Print_Println` وهو اسم يخضع لضوابط تسمية الأسماء وبالتالي فإن استخدام الرمز الخاص `(_)` بين طرفي الاسم لا يسبب خطأ.

أما الأسطر 2 و 4 فهي أسطر فارغة وهي أسطر يعتمد她的 المترجم على أنها أسطر فارغة بلا أوامر، أما استخدامها من طرف المبرمج فيتم بهدف تنظيم مظهر البرنامج.

في المثال تم طلب إظهار الاسم الأول والأخير معاً في سطر واحد ثم إظهار الصفحة الإلكترونية و البلد معاً في سطر آخر، وبالتالي فإننا بحاجة إلى طباعة الاسم الأول بأمر الطباعة `(print)` حتى لا ينتقل المؤشر للسطر التالي، بينما يتم استخدام الأمر `(println)` لطباعة المعلومة الثانية لكي ينتقل المؤشر بعد ذلك للسطر التالي وهذا ما حدث مع طباعة الاسم الثاني، وعلى ذلك يكون مظهر النتائج بعد التنفيذ كما في الصورة التوضيحية 3.10.

```
--Configuration: Print_Println - JDK version 1.6.0_1
Fname: Mahmoud Sname: Rafeek
Site: Staff.cst.ps/mfarra Country: Palestine
Process completed.
```

شكل 3.10: ناتج برنامج مثل 3.2

على شاشة الإخراج، تابع معي أن القيمة الأولى (Fname: Mahmoud) في السطر الأول تتلخص مباشرة بالقيمة الثانية (Sname: Rafeek) وذلك لعدم وجود فراغات داخل علامات التصنيص الزوجية بعد القيمة الأولى، والأمر ذاته في السطر الثاني، فحاول معي أن تترك بعض المسافات بعد القيمة الأولى ستحصل على ناتج قريب من الصورة التوضيحية 3.10.

```
--Configuration: Print_Println - JDK version 1.6.0_1
Fname: Mahmoud Sname: Rafeek
Site: Staff.cst.ps/mfarra Country: Palestine
Process completed.
```

شكل 3.11: ناتج برنامج مثل 3.2 بعد التعديل والتطوير

وبناءً على المثال السابق 3.2، فمن المهم ذكر أن المخرجات يمكن طباعتها بتنسيق يختاره المبرمج من خلال استخدام علامات خاصة للإخراج أو من خلال كتابة المخرجات بطريقة هو يختارها، فكما أوضحنا سابقاً أوامر الطباعة تطبع ما يرسل لها كما هو بالضبط إلا بعض العلامات الخاصة التي يفهمها المترجم بفهم آخر متعارف عليه في اللغة، ونوضح هذه الرموز وتأثيرها من خلال الجدول 3.5.

**تنبيه:** في المثال 3.2 والأمثلة التالية سيتم تجاهل إظهار التعليق التعريفي الافتراضي الذي يكتب في بداية البرنامج.



الرمز	مفهومه وتأثيره	النص داخل أمر الطباعة	النص الظاهر على الشاشة
\n	طباعة العلامة \ ذاتها.	Name\\ Mahmoud Alfarra	Name\\ Mahmoud Alfarra
\\"	طباعة نص يحتوي على علامة تتصيس زوجية	We are all "Muslims"	We are all \"Muslims\"
\r	لإعادة المؤشر لبداية السطر الحالى و بالتالى فإن كل ما يكتب بعدها يتم كتابته فوق الكلام المطبوع من البداية.	Alie	Name\rAli
\t	طباعة ما بعدها في سطر جديد	Mahmoud Rafeek	Mahmoud\tRafeek
\f	تستخدم لطباعة ثمانى مسافات أفقيه	Mahmoud Alfarra	Mahmoud\fAlfarra

جدول 3.5: ملخص الرموز والعلامات الخاصة المستخدمة للطباعة في لغة جافا

وكما تلاحظ فإن علامات الطباعة الخاصة جميعها تحتاج إلى العلامة () لتبنيها وهي تعرف باسم (*escape sequence*)، في المثال التالي 3.3 يتم من خلال توضيح أثر استخدام كلّ من هذه الرموز والعلامات.

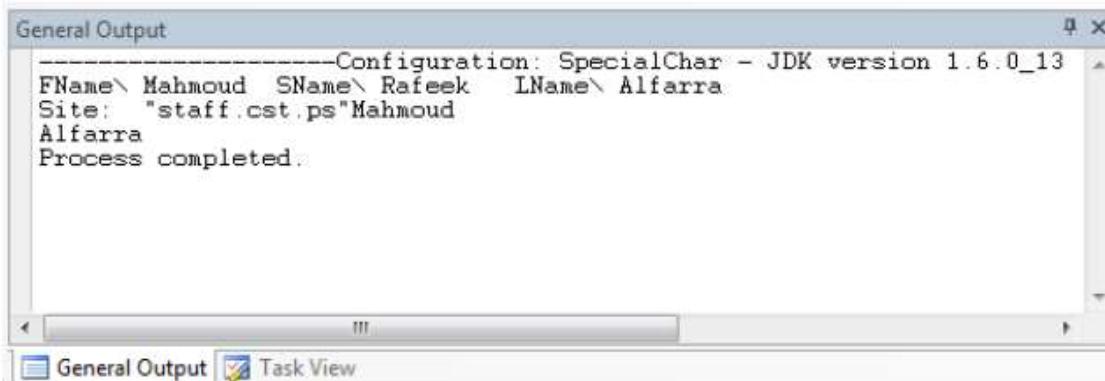
### مثال توضيحي 3.3:

مستخدما جمل طباعة، والرموز الخاصة للإخراج، اكتب برنامجاً يظهر تأثير كلّ من رموز الإخراج الخاصة والموضحة بالجدول 3.5.

```

1 public class SpecialChar {
2
3     public static void main(String[] args){
4         System.out.println("FName\\ Mahmoud\fSName\\ Rafeek\fLName\\ Alfarra");
5         System.out.print("Site: \\\"staff.cst.ps\\\"");
6         System.out.print("Mahmoud\rAlfarra " );
7
8     }
9 }
```

وناتج هذا المثال يظهر كما بالشكل 3.12، وكما تتبع فإن كل رمز من الرموز ترك آثراً يختلف عن الآخر، وكل واحد منهم أصبح له وظيفة مختلفة بمجرد أن التصق بالإشارة () .



شكل 3.12: ناتج برنامج مثل 3.3

من كافة الأمثلة والشرح السابقة وحتى الآن، نهدف إلى أن تتقن بشكل كامل طباعة ما شنت من جمل بالطريقة التي تريدها ومستعينا بالرموز الخاصة، مع الفهم النظري لما مضى، لنكمل المشوار فيما يتبقى بشكل أقوى.



#### مثال توضيحي 3.4:

مستخدمنا جملة طباعة واحدة، أكتب برنامج ليطبع البيانات الخاصة بك (اسمك، عمرك، ديانتك، مدينتك)، على أن يظهر كل واحد منهم في سطر واحد ومع مراعاة المظهر العام والمسافات المناسبة بين العنوان والقيمة كما يلى.

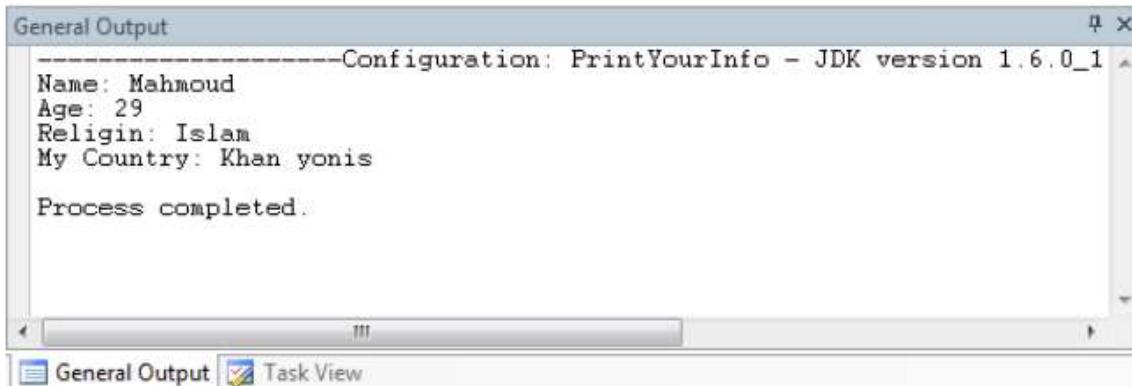
Name: Mahmoud Rafeek Al-farra

```

1. public class PrintYourInfo {
2.
3.     public static void main(String[ ] args) {
4.         System.out.println("Name: Mahmoud\nAge: 29\n"+
5.             "Religion: Islam\nMy City: Khan yonis");
6.     }
7. }
```

اقرأ المثال بشكل جيد لنستخرج منه المطلوب؛ والمطلوب: طباعة أربع معلومات بجملة طباعة واحدة على أن يظهروا على أربعة أسطر متفرقة، وهذا يعني أننا بحاجة لاستخدام الرمز الخاص (\n) بين كل معلومة وأخرى، وهذا ما يظهر في حل المثال في السطرين 4 و5.

أما فيما يخص المسافات فقد استخدمت المسافات اليدوية في هذا المثال لأنك مسافتين وثلاث بين كل معلومة وأخرى. يبقى فقط في هذا المثال أن تتبه جيداً إلى أمر الطباعة أنه ظهر على سطرين وقد تم ربط الجزئين بالإشارة (+) وهي علامة تستخدم في علم الرياضيات لجمع رقمين، بينما في لغة جافا يمكن استخدامه أيضاً لإلصاق نصين مع بعضهما البعض، فهي إن وضعت بين نصين تصبح وظيفتها إلصاق نصوص، وإن وضعت بين رقمين تصبح وظيفتها الجمع، وهذا أحد التطبيقات لمفهوم يعرف باسم التحميل الزائد، وسنتعرف عليه بالتفصيل في باب البرمجة شيئاً فشيئاً التوجه.



شكل 3.13: ناتج برنامج مثل 3.4

### 3.6.8 إظهار نص في صندوق حوار

أظهرنا النتائج فيما سبق من خلال نافذة الأوامر كما في الصورة 3.13، ولكن جافا توفر لنا إمكانية إظهار النتائج من خلال صناديق نص مما يجعلها أكثر سهولة للقراءة و يجعلها تعطي البرنامج الفرصة للتعامل الأسهل مع المستخدم النهائي، وهذه إمكانية من مجموعة إمكانيات ضمن مفهوم ما يعرف بالبرمجة المرئية.

إظهار صندوق النص كما بالصورة 3.14، يتطلب مثلاً استدعاء ما يُعرف بأحد مكاتب جافا الجاهزة غير الافتراضية، فلغة جافا توفر مجموعة من الإمكانيات البرمجية ضمن ما يُعرف بالمكتبات الجاهزة (*Package*) والتي تحتوي على مجموعة من المكتبات الأخرى تحتوي على أصناف تم بناءها مسبقاً في جافا (*Identified classes*) وهذه الأصناف تحتوي على مجموعة من الدوال تقوم بوظائف مختلفة مثل إظهار صندوق الحوار.



شكل 3.14: إظهار النصوص من خلال صندوق الحوار

والمكتبة التي تحتاجها لإظهار الصندوق هي (`javax.swing.JOptionPane`)، ومن داخلها نحتاج استخدام المكتبة (`swing`) ومن داخلها نستخدم الصنف (`JOptionPane`) . هذه المكتبة وما بداخلها لكي يصبح استخدامهم متاحاً أقوم باستخدام الأمر (`import`) كما في الجملة التالية:

```
import javax.swing.JOptionPane;
```

يظل فقط أن تعرف أن جملة استخدام هذه المكتبة لابد أن توضع خارج الصنف، فمثلاً بين التعليق التعريفي وتعريف الصنف الرئيسي للبرنامج كما تتبع في المثال 3.5. ولاظهر الفارق الذي تحدثه هذه المكتبات الجاهزة، دعونا نعيد المثال 3.4. باستخدام صندوق الحوار بمثال جديد هو 3.5 ليظهر الأمر جلياً.

**خطأ برمجي:** عدم كتابة جملة إدخال مكتبة صندوق الحوار (`import`) عند استخدام صندوق الحوار يعتبر خطأ برمجي يمنع تنفيذ البرنامج بصورة صحيحة.

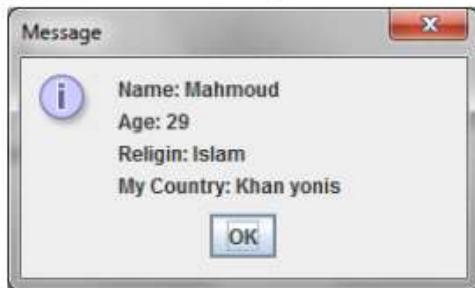


### مثال توضيحي 3.5

مستخدماً صندوق حوار واحد، أكتب برنامج ليطبع البيانات الخاصة بك (اسمك، عمرك، دينك، مدينتك)، على أن يظهر كل واحد منهم في سطر واحد ومع مراعاة المظهر العام والمسافات المناسبة بين العنوان والقيمة كما يلى.

Name: Mahmoud Rafeek Al-farra

```
1. import javax.swing.JOptionPane;
2. public class JOp_first {
3.
4.     public static void main(String[] args) {
5.
6.         JOptionPane.showMessageDialog(null, "Name: Mahmoud\nAge: 29\n"+
7.             "Religion: Islam\nCountry: Khan yonis");
7.     }
}
```

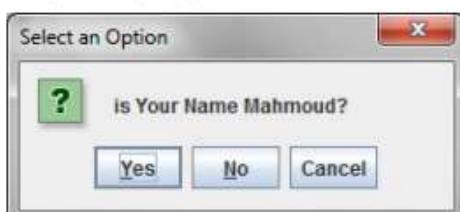


شكل 3.15: ناتج مثال 3.5

وكما تتبع في السطر 6، يظهر فيه أمر استخدام صندوق الحوار وهو

`JOptionPane.showMessageDialog(null, "النص المراد إظهاره");`

وصندوق الحوار هذا له عدة دوال تختلف عن `showMessageDialog` لإنتاج أشكال مختلفة، مثل استخدام `showConfirmDialog`، وهو صندوق الاختيار كما يظهر في شكل 3.16، على كل حال لا يهمنا في هذا المقام الاستفاضة من هذا الامر فهو خاص كما أسلفنا بأوامر البرمجة المرئية والتي لا يدعمها الكتاب الحالي.



شكل 3.16: المظهر العام لصندوق حواري اختياري

### 3.7 العمليات الحسابية

نتيج لنا لغات البرمجة عامة وكذلك لغة جافا، استخدام العمليات الحسابية، وهي متعددة بتنوعها في الحياة الواقعية، من عمليات الجمع إلى القسمة إلى غير ذلك. الجدول 3.6 يوضح العمليات الحسابية المتاحة في لغة جافا مع مفهومها وأمثلة عليها.

الرمز	اسم العملية	مفهومها	مثال عليها
+	الجمع بين الأرقام	تجمع رقمين أو متغيرين من أي نوع و تستخدم في علم الرياضيات بالإشارة ذاتها.	<code>int sum = 3 + 7;</code> <code>float sum = R +3.4f;</code>
-	طرح الأرقام من بعضها	تطرح رقم من رقم آخر من أي نوع و تستخدم في علم الرياضيات بالإشارة ذاتها.	<code>int sub = 10 - 7;</code> <code>float sub = R - 3.2f;</code>
*	ضرب الأرقام	تضرب رقم من رقم آخر من أي نوع و تستخدم في علم الرياضيات بالإشارة ×	<code>int mull = 3 * 7;</code> <code>float mull = R * 3.6f;</code>
/	قسمة الأرقام	تقسم رقم على رقم آخر من أي نوع و تستخدم في علم الرياضيات بالإشارة ÷	<code>int div = 11 / 2;</code> <code>float div = R /3.6;</code>

% باقي القسمة	باقي القسمة
الزيادة بواحد (Increment)	يحسب باقي قسمة رقم صحيح على آخر صحيح وليس لها إشارة في الرياضيات
--	يقوم بزيادة مقدارها 1 على متغير صحيح أو عشري وتحسب في الرياضيات بمعادلة كاملة هي $x = x + 1$
الإنقاص بواحد (Decrement)	يقوم بإنفصال القيمة 1 من متغير صحيح أو عشري وتحسب في الرياضيات بمعادلة كاملة $x = x - 1$

جدول 3.6: ملخص للعمليات الحسابية في لغة جافا

بالإشارة إلى الجدول 3.6، فإن العمليات الحسابية الأربع الأساسية يمكن أن تتم بين أي نوع من الأرقام ولكن يبقى السؤال في ما هو الناتج؟ والإجابة أنه في حال كان المعاملان متماثلين في النوع فالنتيجة تكون من ذات النوع لعمليات الجمع والطرح والضرب، بينما في حال القسمة الناتجة تكون من النوع العشري لأن القسمة غالباً ما ينتج عنها أجزاء عشيرة.

وفي حال كان المعاملان مختلفين في العمليات الأربع ذاتها فالناتج يكون عشري، فمثلاً جمع الرقم 9 مع الرقم 3.2 بكل تأكيد سينتاج عنه رقم عشري قيمته 12.2 وهو عشري. تبقى حالة واحدة وهي عند قسمة عدد صحيح على عدد صحيح وتعرف الناتج من قبل المبرمج أنه عدد صحيح، ففي هذه الحالة يتم تجاهل الكسور وسينتج رقمًا صحيحاً كما في الحالة التالية:

```
int x = 6;
int y = 5;
int div = x/y; // div = 1
```

وقد تجاهل الأجزاء العشرية، بينما إن تم تعريفه (*float*) فالناتج يكون 1.0 وكذلك يتتجاهل الأجزاء العشرية ويوضع فقط الصفر العشري ليضع صبغة عشرية على الناتج دون أن يكون الرقم كاملاً، ذلك لأن جافا تتبع عملية باقي القسمة % للحصول على باقي القسمة.

### مثال توضيحي 3.6:

أكتب برنامج يطبع ناتج العمليتين الحسابيتين الجمع والقسمة، لبيان حالات المعاملات وأنواعهم.

```
import javax.swing.JOptionPane;
public class arithmetics {
    public static void main(String[] args){
```

```

int x = 8;
int y =6;
float a =5.7f;
float b = 7.9f;
int sumInt = x+y;
float sumFlo = x+y;
float sumOfFlo = a+b;
int divint = x/y;
float divfloat = a/b;
int divOfFloat = (a/b); //error
float divIntf = x/y;
// عملية الجمع
 JOptionPane.showMessageDialog(null, "int = int+int "+sumInt+
    "\nfloat = int+int "+sumFlo+
    "\nfloat = float+float "+sumOfFlo);
// عملية القسمة
JOptionPane.showMessageDialog(null, "int = int/int "+divint+
    "\nfloat = float+float "+divfloat+
    "\nfloat = int / int "+divIntf); } }

```



شكل 3.17: ناتج مثل 3.6

عند تنفيذ البرنامج السابق وملحوظة النتائج نقف على عدد من الملاحظات هي:

- عند قسمة عدد صحيح وأخر صحيح يتم تخزين الناتج كرقم صحيح مجرد من الكسور.
- في حال تم قسمة عدد صحيح وأخر صحيح وتخزينهم في رقم عشري، يتجاهل تخزين الكسور ويستبدلها بالرقم صفر مثال ذلك (5.0).

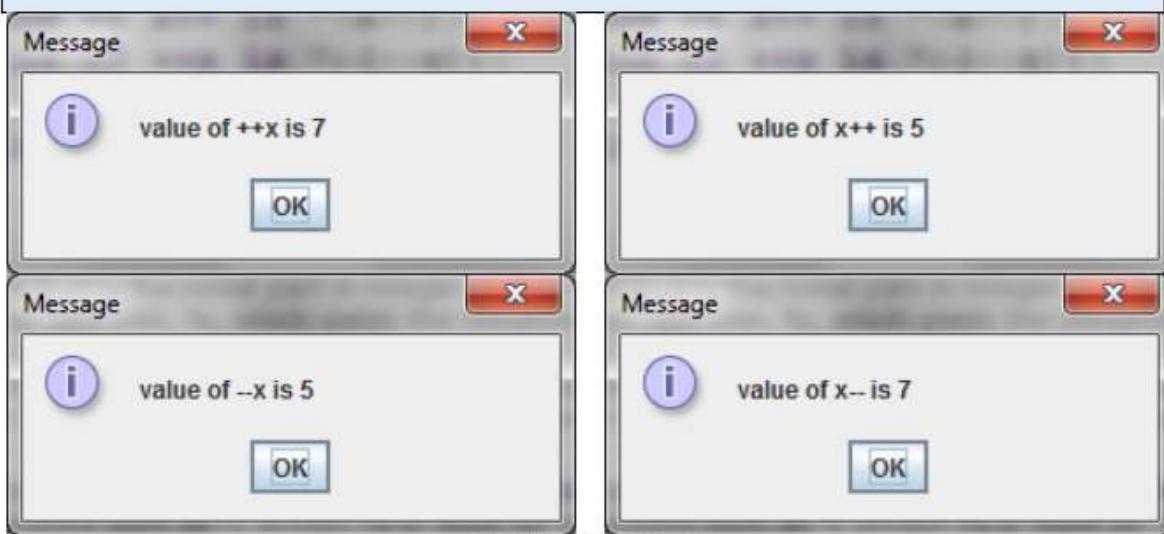
عمليات الزيادة والنقصان بواحد من العمليات التي توفر على المبرمج وقتاً وجهوداً في كتابة بعض المعادلات فهي تختصر المعادلة  $x++ = x + 1$  إلى  $x++$  فقط والأمر ذاته للنقصان.

ويبقى أن تعرف أن هذه العملية يمكن أن يتم وضعها قبل المتغير (*Postfix*) أو بعده (*Prefix*) والفارق بينهم بسيط، فإن وضعت قبل المتغير فتتم الزيادة أو النقصان قبل أن تتم الطباعة أو العملية الحالية والعكس صحيح أي إذا وضعت بعد المتغير فإن الزيادة والنقصان تتم بعد الطباعة أو العملية الحالية، كما يظهر في المثال 3.7.

### مثال توضيحي 3.7:

أكتب برنامج يظهر أظهر العمليات  $++$  ،  $--$  في حالتيهما.

```
import javax.swing.JOptionPane;
public class IncDec {
    public static void main(String[] args){
        int x = 5;
        JOptionPane.showMessageDialog(null,"value of x++ "+x++);
        JOptionPane.showMessageDialog(null,"value of ++x is "+(++x));
        JOptionPane.showMessageDialog(null,"value of x-- is "+x-- );
        JOptionPane.showMessageDialog(null,"value of --x is "+(--x)); } }
```



شكل 3.18: ناتج مثال 3.7

لاحظ في هذا المثال أن عمليات (Post incr.) تتم بعد أن يتم طباعة المتغير قبل الزيادة، بينما تتم عمليات (Pre incr.) قبل الطباعة والحال ذاته لعمليات الإنفاص.

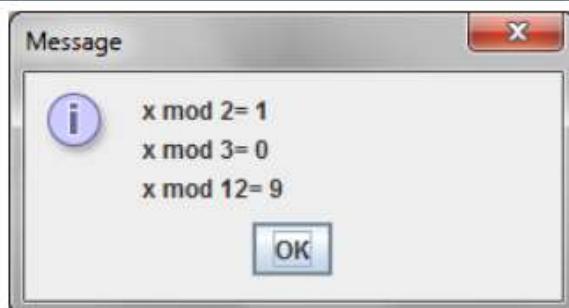
من العمليات الحسابية المهمة في عالم البرمجة هي عملية حساب باقي القسمة لرقمين صحيحين نظراً لكثره تطبيقاتها، فمثلاً باقي قسمة الرقم 7 على الرقم 3 هو 1، ونحسب يدوياً من خلال معرفتنا بأنّ قسمة العدد 7 على 3 هو 2 مع تبقى العدد 1 لأن  $2 \times 3 = 6$  وبالتالي يتبقى 1 من الرقم 7 ، هذا الرقم 1 هو باقي القسمة. وتستخدم هذه العملية في تطبيقات مختلفة وعديدة أبسطها معرفة مضاعفات أرقام معينة، وكذلك في تطبيقات تراكيب البيانات والخوارزميات.

ولمزيد من التعرف على حالات العمليات الحسابية حاول أن تنفذ المزيد من البرامج وتلاحظ ما يحدث معك.

### مثال توضيحي 3.8:

أكتب برنامج يظهر أثر عملية باقي القسمة.

```
import javax.swing.JOptionPane;
public class modOper {
    public static void main(String[] args){
        int x = 9;
        JOptionPane.showMessageDialog(null,"x mod 2= "+x%2+
            "\nx mod 3= "+x%3+
            "\nx mod 12= "+x%12);}}}
```



شكل 3.19: ناتج مثال 3.8

**معلومة مفيدة:** عند حساب باقي قسمة رقم كبير على رقم أصغر منه فالنتائج دائمًا هو رقم الكبير لأنه من الطبيعي لا يقبل القسمة عليه.



**خطأ برمجي:** محاولة حساب باقي قسمة رقم عشري على رقم آخر من أي نوع يعتبر خطأ برمجي حيث أن عملية باقي القسمة لا تتعامل إلا مع معاملات صحيحة فقط.



### الأولويات واستخدام أداة الأقواس:

العمليات الحسابية عندما توضع في معادلات رياضية معقدة قد تخرج لنا نتائج غير التي كنا نتوقعها أو نخطط لها، ذلك لأن بعض العمليات قد تتم على عوامل غير التي كنا نعتقد، لهذا كان هناك مفهوم الأولويات ومفهوم الأقواس لتنظيم عمل العمليات الحسابية وتحديد العمليات التي تتم قبل الأخرى، الجدول 3.7 يوضح الأولويات لكافة العمليات الحسابية و المنطقية، علما بأنها مرتبة ترتيباً تصاعدياً في القوة من أسفل إلى أعلى.

في كافة العمليات السابقة، عندما يوجد في المعادلة ذاتها أكثر من عملية لهم ذات الأولوية يتم التنفيذ من اليسار إلى اليمين.

العملية	ملاحظات
( ) ، ++ ، --	Pre Post
!	عكس قيمة الشرط
* ، / ، %	
+ ، -	
< ، <= ، > ، >=	عمليات منطقية للمقارنة
== ، !=	للمقارنة
&&	أداة تجمع شروط و تشترط جميعهم صحيح
	أداة تجمع شروط وتشترط أحدهم صحيح
=	لإعطاء القيم

جدول 3.7: ملخص الأولويات للعمليات الحسابية في لغة جافا

### مثال توضيحي 3.9

أعد صياغة العمليات الحسابية التالية بلغة جافا، مع توضيح ترتيب تنفيذ العمليات في كل معادلة منهم.

$$\begin{aligned}y &= mx+b \\z &= pr\%q+w/x-y \\y &= x+bx+(c+a)c\end{aligned}$$

المعادلة الأولى بلغة جافا:

$$y = m * x + b;$$

وترتيب العمليات كما يلي: \* ، ، +

المعادلة الثانية بلغة جافا:

$$z = p * r \% q + w / x - y;$$

وترتيب العمليات كما يلي: \* ، \% ، / ، + ، -

المعادلة الثالثة بلغة جافا:

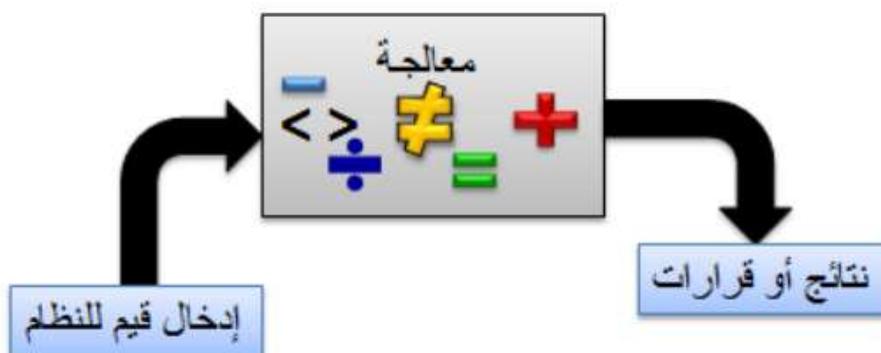
$$y = x + b * x + (c + a) * c$$

وترتيب العمليات كما يلي: + التي بين الأقواس، \* التي في اليسار، \* التي في اليمين، + التي في اليسار، + التي في اليمين.

### 3.8 جمل الإدخال

أغلب إن لم يكن كل الأنظمة البرمجية التي تتعامل معها الشركات والمؤسسات والبنوك تحتاج إلى إدخال قيم من المستخدم وبناءً على هذه القيم تتم عمليات معالجة محددة داخل النظام، ثم يخرج النتائج المستخدمة مرة أخرى، وهذه أنظمة تُعتبر أنظمة تفاعلية حيث يتفاعل المستخدم مع النظام فيها لإنتاج المخرجات المرجوة، كما في شكل 3.20.

لغة جافا تدعم هذه الخاصية من خلال أمر الإخراج (`showInputDialog`) وهو أحد الدوال الجاهزة التي تقدمها جافا ضمن الصنف (`JOptionPane`) ويتم استخدامه بالطريقة الموضحة بالشكل 3.21.



شكل 3.20: الفاندة من إدخال البيانات من المستخدم للنظام

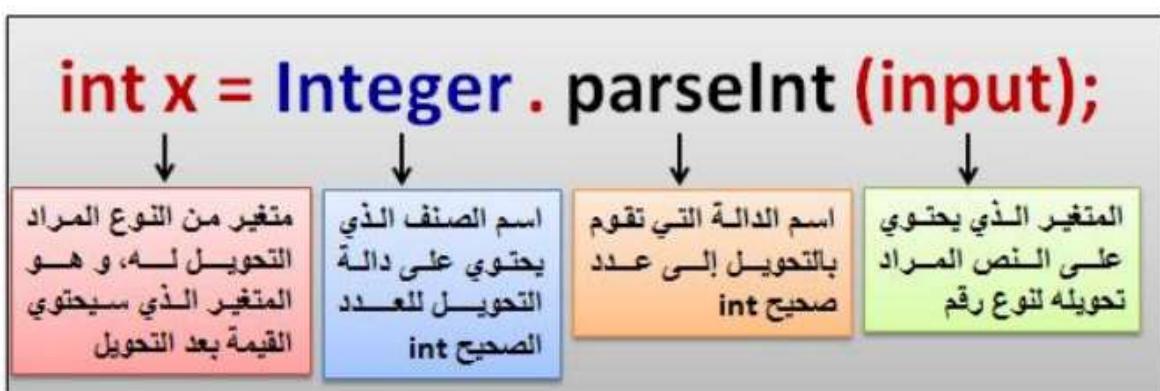
عليك الانتباه عزيزي الدارس إلى أنَّ جملة الإدخال مكونة من شقين أيمن وأيسر، وهو أشبه بالمعادلة الرياضية التي لها طرف أيمن تتم فيه العمليات ثم طرف أيسر تخزن فيه القيم الناتجة، والأمر ذاته في جملة الإدخال حيث يقوم المستخدم من خلال الدالة (`showInputDialog`) الموجودة في الطرف الأيمن بإدخال

البيانات المطلوبة منه، ثم تخزن في الطرف الأيسر والذي هو متغير من النوع (`String`) وهو أحد الأنواع المشتقة من النوع الأساسي (`char`) وتستخدم لتخزين النصوص في الذاكرة.



شكل 3.21: الجملة البرمجية اللازمة لإدخال البيانات من المستخدم

والذي يدعوك للانتباه عزيزي الطالب أن كافة أنواع القيم التي يدخلها المستخدم يتم استقبالها على أنها نص في البداية وبعد ذلك يتم تحويلها لأنواعها الأصلية من خلال دوال للتحويل سنتعرف عليها في هذا الفصل لاحقًا. فالقيم المدخلة للنظام قد تكون أرقام كإدخال قيمة المال لحساب الزكاة الازمة، أو إدخال نصوص كأدخال اسم المستخدم و كلمة المرور في البريد، أو قد تكون أرقام عشرية مثل إدخال نسبة الخدمات بالإضافة إلى الفوائير في أحد المطاعم، وهذه القيم كافة يتم إدخالها كما هي إلا أنها تعتمد على أنها نص ثم نستخدم دالة لتحويلها إلى نوعها المراد، الشكل 3.22 يوضح كيفية التحويل من (`String`) إلى (`int`) وهي الطريقة العامة للت\_conversion مع اعتماد الدوال الازمة، والجدول 3.8 يوضح كافة دوال التحويل من (`String`) إلى الأنواع المتاحة.



شكل 3.22: شرح كيفية التحويل من نوع البيانات `String` إلى النوع `int` كمثال للتحويل

نوع المراد التحويل له	الصنف الذي يستخدم للتحويل	دالة التحويل
Int	Integer	parseint
Byte	Byte	parsebyte
Short	Short	parseshort
Long	Long	parselong
Float	Float	parseFloat
Double	Double	parsedouble

جدول 3.8: دوال التحويل من **String** إلى أنواع الأرقام في لغة جافا

وليس شرطاً أن يتم إرسال متغير للدالة (**parse**) بأحد أنواعها لكي تعمل، بل من الممكن أن يتم إعطاءها نص على هيئة رقم صحيحة مثل إرسال "6" فهذا نص لكنه رقم صحيح، وبهذا يعمل دون مشاكل.

### مثال توضيحي 3.10:

اكتب برنامج لحساب مساحة الدائرة على أن يستقبل نصف قطرها من المستخدم.  
 مساحة الدائرة =  $B = 2 * \pi * R^2$ . حيث قيمة  $\pi = 3.14$ .

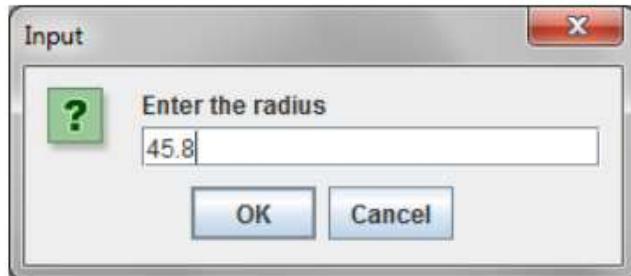
```

1. import javax.swing.JOptionPane;
2. public class Intractive1 {
3.     public static void main(String[ ] args){
4.         استقبال قيمة نصف القطر من المستخدم \|/
5.         String input = JOptionPane.showInputDialog("Enter the radius");
6.         تحويل القيمة من نوع النص إلى نوع الرقم العشري \|/
7.         double R = Double.parseDouble(input);
8.         double area = 3.14 * 3.14 * R;
9.         JOptionPane.showMessageDialog(null,"The area is "+area);
10.    } }
```

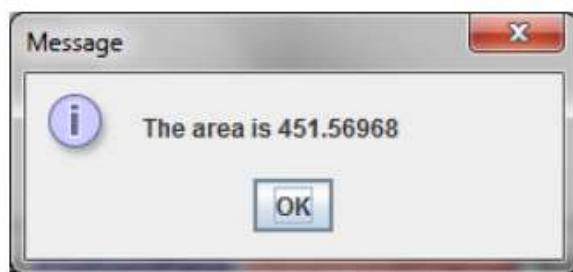
سبق أن قمنا بكتابة الخوارزمية لهذه المشكلة البرمجية في الباب الثاني، وفيها نحتاج إلى إدخال قيمة من المستخدم لنصف القطر ثم نستخدم معادلة مساحة الدائرة لحسابها وطباعتها.

تابع معى في السطر رقم 5 أنشأ طبقنا قيمة من المستخدم من خلال الدالة (**showInputDialog**) وقمنا بتخزينها في متغير من النوع (**String**) ولا يمكننا أن نستقبله في نوع آخر غير هذا النوع (هل تعلم لماذا؟)، بعد ذلك قمنا في السطر رقم 7 بإجراء عملية التحويل إلى النوع العشري وهو **double** وقمنا بتخزين الناتج في متغير من هذا النوع واسمه **R**، هنا عليك الانتباه لماذا حولنا لنوع العشري ولم نحوله لنوع الصحيح؟ فهذه

مسألة لها علاقة بمحل النظم وهي مسألة منطقية أيضاً إذ أن طول أي شيء أو عرضه قد يكون بالكسور وليس شرطًا أن يكون صحيحًا. تابع في الأشكال 3.23 و 3.24 الإطارات الخاصة بالتنفيذ.



شكل 3.23: طلب إدخال قيمة من المستخدم خلال ناتج مثال 3.10



شكل 3.24: مساحة الدائرة بعد إدخال القيمة من المستخدم خلال ناتج مثال 3.10

### مثال توضيحي 3.11

اكتب برنامج يستقبل رقمين صحيحين من المستخدم و يطبع ناتج ضربهم، وجمعهم، وطرح الأول من الثاني، وقسمة الأول على الثاني ثم باقي قسمة الأول على الثاني.

```

1 import javax.swing.JOptionPane;
2 public class OperOnInt {
3     public static void main(String[ ] args){
4         String input = JOptionPane.showInputDialog("Enter first number");
5         int fno = Integer.parseInt(input);
6         input = JOptionPane.showInputDialog("Enter Secound number");
7         int sno = Integer.parseInt(input);
8         int sum = fno + sno;
9         int sub = fno - sno;
10        int mul = fno * sno;
11        double div = fno/sno;
12        int mod = fno%sno;
13        JOptionPane.showMessageDialog(null,"Sum "+fno+" and "+sno +

```

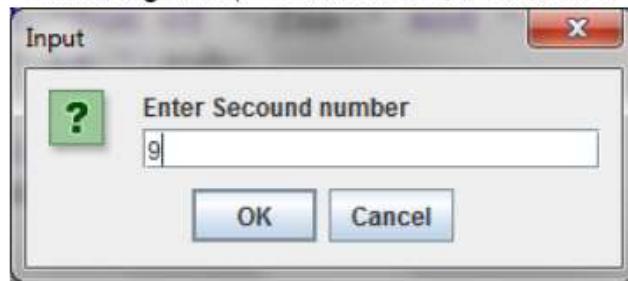
```

14 "is "+sum+" \nSub "+fno+" from "+sno+" is "+sub+" \nMul "+fno+
15 "and "+sno+" is "+mul+" \nDiv"+fno+" By"+sno+" is "+div+
16 "\nMod divide"+fno+" By "+sno+" is "+mod);
17 }

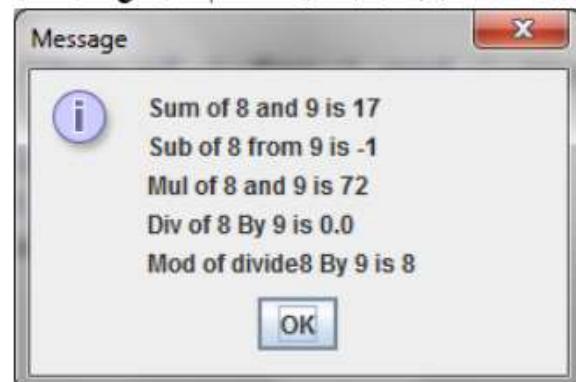
```



شكل 3.25: طلب إدخال قيمة من المستخدم خلال ناتج مثال 3.11



شكل 3.26: طلب إدخال قيمة من المستخدم خلال ناتج مثال 3.11



شكل 3.27: القيم بعد عملية المعالجة لمثال 3.11

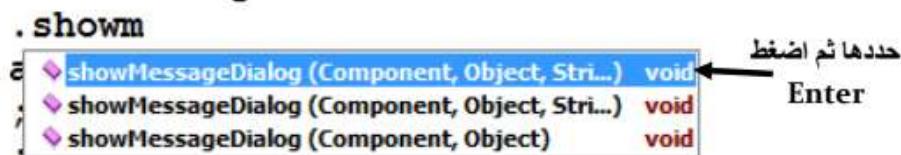
الأشكال 3.25، 3.26 و 3.27 تظهر ناتج تنفيذ البرنامج من حيث إدخال القيم والعمليات الحسابية.

في هذا المثال كما يتضح لك، تم استخدام صندوق الإدخال لاستقبال القيم في الأسطر 4 و 6 ثم تحويلهم إلى قيم صحيحة في الأسطر 5 و 7، بينما تم طباعة القيم بشكل لائق كما في الشكل 3.27 من خلال صندوق حول في الأسطر من 13 حتى 16، ويلاحظ في جملة الطباعة هذه أنها جاءت على أربعة حيث تم

استخدام العلامة (+) لربط النص في السطر السابق وبالتالي وهذا ما تراه في نهاية الأسطر 13، 14، 15، 16؛ وبالتالي يمكنك الاستغناء عن ذلك باستخدام أكثر من جملة طباعة. وبملاحظة النتائج تجد أن باقي قسمة العدد 8 على 9 هو 8 لأن المقسم أصغر من المقسم عليه كما نبهنا على ذلك سابقاً، كذلك قسمة العدد 8 على العدد 9 يساوي 0.0 لأننا أردنا تخزينها في متغير عشري، حاول تخزينها في متغير صحيح ورافق الناتج. بهذا أعتقد أن الصورة لأساسيات لغة البرمجة جافا أصبحت واضحة لك عزيزي الدارس ولتزداد لك وضوحاً، أنصحك بالعودة للباب الثاني، ثم القيام بتحويل كافة الخوارزميات من النوع المتسلسل إلى برامج فهذا سيساعدك كثيراً على فهم كثير من مكونات البرمجة التي قمنا بشرحها حتى الآن.

يبقى أن نقف مع مجموعة تنبّهات هامة حول كتابة البرامج حتى هذه الخطوة:

1. حاول أن تقلل في الفترة الأولى من اعتمادك على نفسك في كتابة الألفاظ الرئيسية للبرامج مثل أسماء المكتبات والأصناف والدوال الظاهرة، واعتمد قدر الإمكان على المساعدة التي يقدمها لها المحرر، وذلك حتى تبتعد عن ظهور الأخطاء المتعلقة بالإملاء، مع الانتباه الجيد حتى تتمكن من تصحيح الأخطاء إن واجهتك دون وجود محرر.



شكل 3.28: الاستعانة بمساعدة المحرر لابتعاد عن الأخطاء.

2. تذكر أن لغة جافا حساسة لحالة الأحرف في كافة الألفاظ المستخدمة ما عدا النصوص التي تكتب داخل علامات التصنيص الزوجية، وأن مجرد كتابتك لحرف يختلف عن حالته المفترضة سيسبب لك خطأ برمجي.

3. انتبه إلى أن التعليمات التي نكتبها حتى الآن تكون دائماً داخل الدالة الأساسية `main`، وأن كتابة تعليمات خارجها سيسبب خطأ برمجي.

4. انتبه إلى ضرورة أن يكون اسم الصنف الرئيسي للبرنامج مماثل لاسم المشروع الذي بدأته، كما تتابع في الشكل 3.28 وإلا لن يعمل البرنامج.

5. اسم الدالة الأساسية `main` هو اسم ثابت ولا يمكن تغييره وإلا يصدر خطأ برمجي.



شكل 3.28: الاستعانة بمساعدة المحرر لابتعاد عن الأخطاء.

6. اعلم أن أي مسافات تتركها داخل أقواس النص تظهر كما هي عند الطباعة.
7. النص الذي تكتبه داخل الأقواس يظهر كما هو بالضبط وينفس حالة الأحرف فيما عدا الرموز والعلامات الخاصة التي قلنا بشرحها.
8. انتبه إلى ضرورة إعطاء قيمة للمتغير تتناسب مع نوعه، لأن إعطاء المتغير قيمة تختلف مع نوعه تسبب خطأ برمجي.
9. الأخطاء التي تظهر معنا خلال البرمجة لها ثلاثة أنواع ستناقشها في الباب الخامس بالتفصيل، وهي باختصار:

أ- الخطأ البرمجي أو القاعدي (*Syntax error*)

ب- الخطأ المنطقي (*Logical error*)

ت- الخطأ الاستثنائي (*Exception error*)

10. تتوفر جملة إدخال أخرى في لغة جافا من الصنف (*Scanner*) تكتب على النحو التالي:

```
Scanner sc = new Scanner(System.in);
int l = sc.nextInt();
```

وهي تابعة للمكتبة *java.util.Scanner*، ويضم هذا الصنف مجموعة كبيرة من الدوال الجاهزة يمكنك تجربتها.

## 3.9 أمثلة و تدريبات عامة

1. استخدم الكلمات المناسبة لتعبئنة الفراغات في الجمل التالية:

- أ- تصنف البرمجيات إلى \_\_\_\_\_ و \_\_\_\_\_ و \_\_\_\_\_.
- ب- يصنف المبرمجون مراحل تطوير البرمجة إلى ثلاثة مراحل وهي على الترتيب \_\_\_\_\_ و \_\_\_\_\_ و \_\_\_\_\_.
- ت- تقع مهمة حل المشاكل برمجياً على عاتق كائنين هما \_\_\_\_\_ و \_\_\_\_\_ و \_\_\_\_\_.
- ث- من مكونات لغة Java \_\_\_\_\_ و \_\_\_\_\_ و \_\_\_\_\_ و \_\_\_\_\_.
- ج- تصنف أنواع البيانات في لغة Java إلى نوعين هما \_\_\_\_\_ و \_\_\_\_\_.
- ح- تعريف أي من المتغيرات لابد أن يحتوي على ثلاثة أجزاء هم \_\_\_\_\_ و \_\_\_\_\_ و \_\_\_\_\_.
- خ- يقصد بأن لغة Java لغة حساسة أنها \_\_\_\_\_.
- د- من ضوابط الاسم التعريفي \_\_\_\_\_ و \_\_\_\_\_ و \_\_\_\_\_.
- ذ- تستخدم علامة الترقيم (;) ل\_\_\_\_\_.
- ر- تغيير اسم الدالة الأساسية **main** يؤدي إلى \_\_\_\_\_.

2. حدد صحة أو خطأ كلاً من العبارات التالية مع التعليل:

- ( ) أ- يلجأ الإنسان إلى البرمجة من باب الرفاهية التي تميز بين طبقات المتعلمين.
- ( ) ب- لا يمكن للحاسوب فهم أي لغة من لغات البرمجة دون وجود وسيط للترجمة .
- ( ) ت- يقع دور الحاسوب في حل المشاكل بين مرحلتي التخطيط والتنفيذ.
- ( ) ث- من أنواع البيانات الأساسية **boolean** وقيمته قد تكون **null, false, true**.
- ( ) ج- التعبر (**float x = 2.3**) يصدر عنه خطأ برمجي لا يمنع تنفيذ البرنامج.

- ح- يمكن لاسم التعريف أن يحتوي على رموز خاصة مثل \_ ، \$ و غيرهم
- خ- يعتبر area2 اسم تعريف صحيح.
- د- الكلمات المhogزة يمكن استخدامهم كأسماء تعريفية مع تغيير حالة أحرفهم لكبيرة.
- ذ- محاولة تعديل قيمة متغير ثابت بعد تعريفه ثابتًا يعتبر خطأ برمجي يمنع تنفيذ البرنامج.
- ر- يستخدم الرمز // لجعل النص التالي له يطبع في السطر بعد التالي.

### 3. عَبَرْ عن الجمل التالية بلغة جافا:

- أ- طباعة رسالة ترحيب تحمل جملة (Salamo alikom) تظهر ضمن نافذة الأوامر.
- ب- طباعة خمس نجوم رئيسية بجملة طباعة واحدة.
- ت- استقبال الاسم من المستخدم ثم طباعته.
- ث- استقبال رقم صحيح من المستخدم وطباعته.
- ج- استقبل رقم من المستخدم وطباعة باقي قسمته على 2.
4. أكتب برنامج بلغة جافا ليطبع البيانات الخاصة بك (اسمك، عمرك، ديانتك، مدينتك)، على أن يكون كل واحد منهم بجملة طباعة ويظهروا جميعاً في سطر واحد يفصل بينهم ثمانى مسافات.
5. أعد السؤال رقم 4 على أن يظہروا في صندوق حوار واحد فقط وكل معلومة في سطر.
6. مستخدماً جملة طباعة واحدة، اكتب برنامج لطباعة الأشكال التالية على أن يظهر كل واحد منها في صندوق حوار.

*	*****	*****
**	***	**
***	**	*****
*****	*	**
(ج)	(ب)	(ا)

7. مستخدماً الرموز الخاصة وأربعة جمل طباعة، اكتب برنامج جافا لطباعة البيانات على النحو التالي:

ID	Name
120090789	Ali
120100768	Hussam
12010543	Mahmoud

8. أكتب برنامج ليقرأ اسمك ثلاثة ثم يظهره كما في الشكل 3.29:



شكل 3.29

9. أكتب برنامج يستقبل من المستخدم ثلاثة أرقام صحيحة يمكن تمثيلها بحد أقصى 16 بت، ثم يطبع كلاً من:

أ- باقي قسمة الأول على الثاني مجموعاً للثالث.

ب- حاصل مضاعف مجموع الأول والثاني مطروحاً منه الثالث.

ت- حاصل مجموع الأول والثاني مرفوع للقوة 2 ثم مقسوماً على الثالث.

10. اكتب برنامج لحساب قيمة كل من المتغيرات A، B، C في المعادلات التالية:

$$A = X^2 + 2Y$$

$$B = 2X - 3A$$

$$C = A^2 + XB$$

### 3.12 تمارين ذاتية الحل

1. اكتب برنامجاً يستقبل من المستخدم رقمين ويطبع مجموعهم وضربيهم وقسمة الثاني على الأول وبباقي قسمة مضاعف الأول على نصف الثاني.

2. اكتب برنامجاً تفاعلياً مع المستخدم لحساب معدل السكان لمدن قطاع غزة الخامس، مع طباعة عدد سكان كل مدينة والمعدل لهم في صندوق حوار واحد.

3. اكتب برنامجاً لحساب معدل الطلبة الزائدين عن 30 في كل فصل لإحدى المدارس والتي تحتوى على خمسة قاعات دراسية، مع طباعة المعدل، وأعداد الزيادة في كل فصل.
4. اكتب برنامجاً لحساب قيمة الزكاة للمال، على أن يقوم المستخدم بإدخال مقدار المبلغ الذي معه ويطبع البرنامج قدر الزكاة المفروضة عليه. علماً بأن مقدار الزكاة يقدر ب (2.5%) من المبلغ.
5. اكتب برنامجاً تفاعلياً مع المستخدم لحساب قيمة الرسوم التي يدفعها طالب الدبلوم خلال فترة دراسته بالكلية، إذا كان مقابل الساعة 10 دنانير، علماً بأن إجمالي الرسوم تحسب بالمعادلة:
- $$\text{اجمالي الرسوم} = \text{عدد الساعات} * \text{عدد المواد للفصل الواحد} * \text{عدد الفصول للعام الواحد} * \text{عدد السنوات}$$
6. اكتب برنامجاً يستقبل من المستخدم ثلاثة أرقام عشرية، ثم يطبع المتوسط الحسابي والمجموع وحاصل الضرب.
7. اكتب برنامجاً تفاعلياً، لحساب مساحة ومحيط إحدى غرف المدرسين بالكلية، علماً بأنها على شكل مستطيل.
8. حول كلاً من المعادلة الجبرية التالية للصيغة تقبلها لغة جافا، مع توضيح ترتيب الأولويات
1.  $y = ax^3 + 7$
  2.  $x = 7 + 3 * 6 / 2 - 1$
  3.  $z = 2Y^2 + RK4$
9. اذكر أربعة من مكونات لغة جافا، مع مناقشة أهميتها وقواعدها مستخدماً الأمثلة قدر الإمكان للتوضيح.
10. اكتب برنامجاً يستقبل من المستخدم رقمين صحيحين ويطبع مضاعف الأول مقسوماً على الثاني.

#### **اجابات التدريبات العامة:**

1. (أ) نظم تشغيل، البرامج التطبيقية، لغات البرمجة (ب) لغة الآلة، لغة التجميع، لغات المستوى العالي (ت) الإنسان، الحاسوب (ث) جمل الإخراج، علامات الترميم، الكلمات المحجوزة (ج) أساسية، مشتقة (ح) نوع البيانات، الاسم التعريفي، فاصلة منقوطة (خ) أن حالة الأحرف لا تعتبر متساوية (د) لا يحتوي فراغات، لا يبدأ برقم (ذ) لتمييز خواتم الجمل البرمجية (ر) خطأ برمجي .2
- أ- (خاطئة) يلجأ الإنسان إلى البرمجة لزيادة الإنتاج وتسريع العمل وتقليل الأخطاء.
- ب- (خاطئة) يمكن له من خلال لغة الآلة.
- ت- (خاطئة) يقع دور الحاسوب في حل المشاكل بين مرحلتي التنفيذ والاختبار.

- ث - (خاطئة) من أنواع البيانات الأساسية *boolean* وقد تكون *false* *true* فقط.
- ج - (خاطئة) التعبير *(float) x = 2.3* يصدر عنه خطأ برمجي يمنع تنفيذ البرنامج.
- ح - (خاطئة) يمكن لاسم التعريف أن يحتوي على رموز خاصة مثل *\_* ، *\$* فقط.
- خ - (صحيحة) يعتبر *area2* اسم تعريف صحيح.
- د - (صحيحة) الكلمات الممحورة يمكن استخدامهم كأسماء تعريفية مع تغيير حالة أحرفهم لكبيرة.
- ذ - (صحيحة) محاولة تعديل قيمة متغير ثابت بعد تعريفه ثابتًا يعتبر خطأ برمجي يمنع تنفيذ البرنامج.
- ر - (خاطئة) يستخدم الرمز *\n* لجعل النص التالي له يطبع في السطر التالي مباشرة.

.3

(إ)

System.out.println("Salamo alikom");

(ب)

System.out.println("\n\*\n\*\n\*\n\*");

(ت)

```
String input = JOptionPane.showInputDialog("Enter number");
int x = Integer.parseInt(input);
JOptionPane.showMessageDialog(null,"number is"+x);
```

(ث)

```
String input = JOptionPane.showInputDialog("Enter number");
int x = Integer.parseInt(input);
JOptionPane.showMessageDialog(null,"number is"+x%2);
```

.4

```
public class Practice4 {
    public static void main(String[] args) {

        System.out.print("Name: Mahmoud\t");
        System.out.print("Age: 29\t");
        System.out.print("Religion: Muslim\t");
        System.out.print("Country: Palestine\t");
    }
}
```

شكل 3.30: حل ترين 4

.5

```
import javax.swing.JOptionPane;
public class Practice5 {

    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null,"Name: Mahmoud"+
            "\nAge: 29\nReligion: Muslim\nCountry: Palestine");
    }
}
```

شكل 3.31: حل تمرين 5

.6

```
import javax.swing.JOptionPane;
public class Practice6A {

    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null,"*****\n**\n*****\n**");
    }
}
```

شكل 3.32: حل تمرين 6 (أ)

```
import javax.swing.JOptionPane;
public class Practice6B {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null,"*****\n"+
            "****\n**\n**");
    }
}
```

شكل 3.33: حل تمرين 6 (ب)

```
import javax.swing.JOptionPane;
public class Practice6C {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null,"*\n"+
            "**\n***\n*****");
    }
}
```

شكل 3.34: حل تمرين 6 (ج)

7. مستخدما الرموز الخاصة وأربعة جمل طباعة ، اكتب برنامج جافا لطباعة البيانات على النحو التالي:

```
public class Practice7 {
    public static void main(String[] args) {
        System.out.println("ID\t\ttName\t");
        System.out.println("120090789\tAli\t");
        System.out.println("120100768\tHussam\t");
        System.out.println("12010543\tMahmoud \t");
    }
}
```

شكل 3.35: حل تمرين 7

.8

```
import javax.swing.JOptionPane;
public class first {
    static String fname;
    static String sname;
    static String lname;
    public static void main(String[] args) {
        fname = JOptionPane.showInputDialog("Please, first name: ");
        sname = JOptionPane.showInputDialog("Please, secound name: ");
        lname = JOptionPane.showInputDialog("Please, last name: ");
        JOptionPane.showMessageDialog(null, "First name: "+ fname +"\n"
            +"Second name: "+sname + "\n Last name: "+lname);
    }
}
```

شكل 3.36: حل تمرين 8

```

import javax.swing.JOptionPane;
public class arithm {
    static String input;
    static short no1, no2, no3;
    static int res1, res2, res3;

    public static void main(String[] args) {

        input = JOptionPane.showInputDialog("input your first number: ");
        no1 = Short.parseShort(input);
        input = JOptionPane.showInputDialog("input your second number: ");
        no2 = Short.parseShort(input);
        input = JOptionPane.showInputDialog("input your third number: ");
        no3 = Short.parseShort(input);
        res1 = (no1+no2)+ no3;
        res2= (2 * (no1+no2) )- no3;
        res3 = ((no1+no2) * (no1+no2))/no3;
        JOptionPane.showMessageDialog(null, "The values are no1: "+ no1+" no2: "
            +no2+" no3: "+no3+"\n(no1+no2)+ no3= "+ res1+"\n"+
            "(2 * (no1+no2) )- no3 = "+res2+ "\n"+
            "((no1+no2) * (no1+no2))/no3= "+ res3);
    }
}

```

شكل 3.37: حل تمرين 9

```

import javax.swing.JOptionPane;
public class equations {
    static String input;
    static int x;
    static int y;
    static int A, B, C;

    public static void main(String[] args) {

        input = JOptionPane.showInputDialog("input The x value: ");
        x = Integer.parseInt(input);
        input = JOptionPane.showInputDialog("input your second number: ");
        y = Integer.parseInt(input);

        A = x*x + 2*y;
        B = 2*x - 3*A;
        C = A*A + x * B;

        JOptionPane.showMessageDialog(null, "A: "+ A+"\n B: "+
            +B+"\n C: "+C);
    }
}

```

شكل 3.38: حل تمرين 10

انتهى الباب

في أي مكان ترى عملاً ناجحاً ستجد  
شخساً ذات مرة اتخذ قراراً شجاعاً.

فلا تتردد في اتخاذ القرار السليم فالخير  
سيعم الجميع ولو بعد حين!!

## الباب الرابع

### جمل الاختيار

(Selection Statements)

يتوقع من الدارس في نهاية هذا الباب أن:

- يدرك مفهوم التحكم في سير العمليات.
- يشرح مفهوم اتخاذ القرار وموضع الحاجة إليه.
- يدرك آلية تمييز الحاسوب بين القيم المختلفة للاختيار بينها.
- يتقن كتابة الشروط المنطقية للمقارنة والفحص.
- يعدد فوائد جمل الاختيار في التطبيقات البرمجية.
- يُفرق بين جمل الاختيار في لغة Java وحالات استخداماتها.
- يُقدر أهمية جمل الاختيار في لغات البرمجة.

يتوفر لهذا الباب شرائح العرض (PowerPoint) وكذلك ملفات مرئية من إعداد مؤلف الكتاب؛ يمكنك الحصول عليها من خلال الموقع الأكاديمي للمؤلف [mfarra.cst.ps](http://mfarra.cst.ps)



## الباب الرابع: جمل الاختيار

### 4.1 مقدمة

تعتمد لغة جافا في تنفيذ التعليمات البرمجية على مفهوم التنفيذ التسلسلي (*Sequential execution*) ويقصد به أن الجمل البرمجية في برامج لغة جافا (وكذا اللغات الأخرى) يتم تنفيذها الواحدة تلو الأخرى حسب الترتيب التي كتبت به وهذا هو النمط الافتراضي، إلا أن بإمكان المبرمج استخدام مجموعة من الجمل ليتحكم من خلالها في سير العمليات وليس بامكانه تحديد المسار الذي يريد بشكل مغاير عن المسار التسلسلي. وبهذا لم يعد ضروريًا أن تكون الجملة التالية في التنفيذ هي الجملة المكتوبة في السطر التالي وهذا ما يعرف باسم نقل مسار التنفيذ (*Transfer of control*), ويُعرف أيضًا بمفهوم التحكم في سير العمليات.

وهذه الجمل تقسم إلى قسمين هما:

- **جمل الاختيار** (*Selection Statements*) وتحرف أيضًا بجمل اتخاذ القرار (*Decision Making*) وسنناقشه في هذا الباب.
- **جمل التكرار** (*Looping Statements*) وتحرف أيضًا بجمل الدوران وسنناقشه في الباب الخامس بالتفصيل والأمثلة.

غير أن كلا النوعين من الجمل يعتمد على بناء شروط منطقية معتمدة على مجموعة من العوامل المنطقية وهذا ما سنعرف عليه في الفصل 4.2 قبل أن نشرح منطق جمل الاختيار وأهم تطبيقاته العملية في الفصل 4.3 لنمارس هذا الفهم من خلال التعرف على أدوات الجمل البرمجية التي تستخدم لبناء تطبيقات برمجية معتمدة على الاختيار في الفصل 4.4.

### 4.2 أساس بناء الشروط المنطقية

الشروط المنطقية هي عبارة عن أسئلة يمكن الإجابة عليها بإحدى إجابتين (نعم أو لا) كأن تسأل أحدهم (هل صليت الفجر اليوم في المسجد؟) فيجيبك بـنعم أو لا.

والشروط المنطقية يتم استخدامها فيما يعرف بالجمل الشرطية ومثالها (إن صلت غدبر صلاة الجمعة سمعطيها جائزة). وتكون الجمل الشرطية من خمسة مكونات؛ وهي على النحو التالي:

أ- الطرف الأيمن.

- بـ- الطرف الأيسر.
- تـ- علاقة منطقية.
- ثـ- أداة شرط منطقية.
- جـ- نتیجة الشرط

فمثلاً قد تسأل (إن كان عمرُ محمود أكبر من عمرُ محمد سيقود محمود السيارة أولاً)، في هذا المثال ستجد التالي:

- أـ العامل الأيمن هو (عمرُ محمود)
- بـ العامل الأيسر هو (عمرُ محمد)
- تـ العلاقة المنطقية هي (أكبر من)
- ثـ أداة الشرط المنطقية هي (إن كان)
- جـ نتیجة صحة الشرط هي (سيقود محمود السيارة أولاً)

فالجمل الشرطية في الواقع لها صورة مقابلة في البرمجة، تصبح فيها المعاملات عبارة عن قيم أو متغيرات، بينما العلاقة المنطقية هي علاقات مقابلة في البرمجة توضحها في الجدول 4.1، بينما أدوات الشرط المنطقية هي موضوع هذا الباب ونناشرها في الفصل 4.4 ويبقى فقط جواب الشرط وهو مجموعة جمل برمجية قد تكون بسيطة مثل التي تعلمنا كتابتها في الباب الثاني أو معقدة بشكل سنشاهده لاحقاً.

رمز العلاقة	مفهوم العلاقة	مثال عليها	رمزها في الحياة
$==$	لفحص تساوي طرفيه		$X == Y$
$!=$	لفحص عدم التساوي		$X != Y$
$>$	فحص أنَّ الطرف الأيسر ( $X$ ) أكبر من الأيمن ( $Y$ )		$X > Y$
$<$	فحص أنَّ الطرف الأيسر ( $X$ ) أصغر من الأيمن ( $Y$ )		$X < Y$
$\leq$	فحص أنَّ الطرف الأيسر أكبر من أو يساوي الأيمن		$Y \leq X$
$\geq$	فحص أنَّ الطرف الأيسر أصغر من أو يساوي الأيمن		$Y \geq X$

جدول 4.1: ملخص العلاقات المنطقية البسيطة في لغة جافا

بناءً على ذلك، فإن الشرط المنطقي في لغة جافا يتكون من قيمتين (أو ما يمثلهم من متغيرات) يربطهما إحدى العلاقات المنطقية الموضحة في جدول 4.1 وأمثلة ذلك ما تراه في عمود (مثال عليها) في الجدول ذاته.

وللمزيد من الأمثلة تابع معي ما يلي من أمثلة مختلفة للشروط المنطقية:

`3 < x, 7 > y, 6 == 7, 6 <= 8, u != 7, k ≤ L`

انتبه إلى الفارق بين علامة المساواة وهي (=) واحدة فقط بينما علامة فحص المساواة عبارة عن (==).



**خطأ منطقي:** استخدام إشارة = بدلاً من == لفحص المساواة يعتبر خطأ منطقي، يؤثر على طبيعة النتائج التي يصدرها البرنامج، حيث يجعل نتيجة الفحص (true) دائمًا.



**خطأ برمجي:** تبديل موضع إشارة = في العلاقة المنطقية = > أو => يؤدي إلى خطأ برمجي، فعليك الانتباه دائمًا إلى أنها على اليمين من الإشارة.



لعل أحد الدارسين يسأل الآن، وكيف يمكن لنا أن نربط بين شرطين أو أكثر، مثل القول: (لن يُسمح لرفيق أن يُسجل الدخول لصفحة إدارة الموقع إلا عندما يدخل (اسم المستخدم وكلمة المرور) بشكل صحيح)، ففي هذه الحالة دخول رفيق لهذه الصفحة مرتبط بصحّة شرطين، بينما عندما نقول إذا كان الطالب من تخصص (الحاسوب أو الهندسة) يمكنه الانضمام للدورة التدريبية فسيتم فحص الشرطين لتأكد من صحة أحدهما. هذه الحالات ولها صور أخرى نستخدم فيها شروط منطقية مركبة أي أنها شروط مكونة من أكثر من شرط بسيط يتم الربط بينهم بعلاقات موضحة بالجدول 4.2.

رمز العلاقة	مفهوم العلاقة	مثال عليها
&&	ترتبط شرطين صحيحين وتُنفَذ And	<code>(x == y)&amp;&amp;(x&gt;z)</code>
	ترتبط شرطين أحدهما على الأقل صحيحًا وتُنفَذ Or	<code>(x != y)   (x&lt;z)</code>
^	ترتبط شرطين على أن يكون أحدهما خاطئًا وتُنفَذ Xor	<code>(x == y)^^(x&gt;z)</code>
!	يتم من خلالها عكس نتيجة الشرط المركب وتُنفَذ Not	<code>!(x == y)</code>

جدول 4.2: ملخص علاقات الربط للشروط المركبة في لغة جافا

ولكل واحدة من هذه العلاقات جدول يعرف باسم جدول الحقيقة (*Truth Table*) يتم فيه توضيح النتائج الصادرة عند استخدام هذه العلاقة بين شرطين، وكلّا من هذين الشرطين له نتيجة منطقية (صحيحة، خاطئة)، كما في الجداول 4.3 حتى 4.6.

الشرط 1 && الشرط 2	الشرط 2	الشرط 1
false	false	false
false	true	false
false	false	true
true	true	true

جدول 4.3: جدول الحقيقة لعلاقة الربط (&&)

الشرط 1    الشرط 2	الشرط 2	الشرط 1
false	false	False
true	true	False
true	false	True
true	true	True

جدول 4.4: جدول الحقيقة لعلاقة الربط (||)

انتبه إلى أنَّ الشرط المركب لا يكون صحيحاً عند ربطه بالعلاقة (**&&**) إلا إذا كان كلا الشرطين صحيحاً فإذا كان الشرط الأول صحيحاً وكذلك الشرط الثاني صحيحاً فإنَّ النتيجة النهائية للشرط المركب تكون صحيحة، بينما الشرط المركب بالعلاقة (**||**) يكون صحيحاً بمجرد أن يكون أحد الشرطين صحيحاً فهي تعني إذا كان الشرط الأول صحيحاً أو كان الشرط الثاني صحيحاً فإنَّ النتيجة النهائية للشرط المركب صحيحة. بينما لا يكون الشرط المركب صحيحاً إذا تم ربط شروطه البسيطة باستخدام العلاقة (**Xor**) إلا إذا كان أحد الشروط خاطئة.

الشرط 1 ^ الشرط 2	الشرط 2	الشرط 1
false	false	False
true	true	False
true	false	True
false	true	True

جدول 4.5: جدول الحقيقة لعلاقة الربط (^)

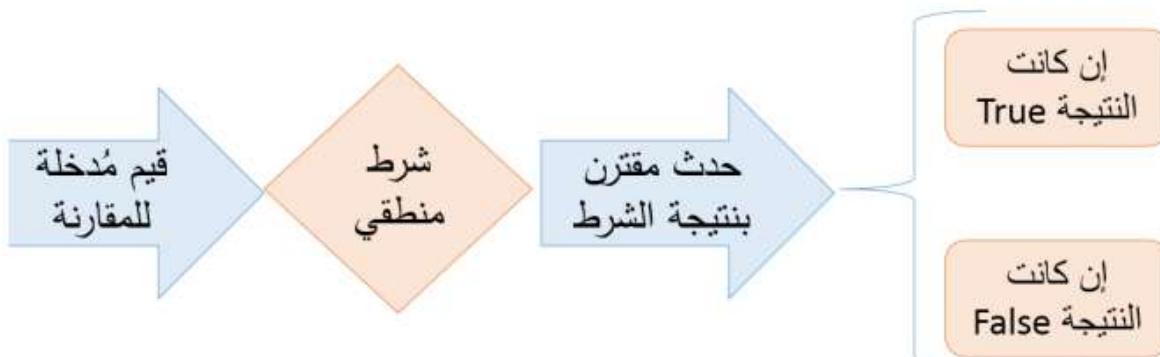
الشرط 1	
true	false
false	True

جدول 4.6: جدول الحقيقة لعلاقة الربط ! (Not)

وهذه الشروط وال العلاقات ستزيد وضوحاً من خلال الأمثلة والتدريبات القادمة.

#### 4.3 فهم منطق اتخاذ القرار وتطبيقاته العملية

التحكم في سير العمليات من الظواهر دائمة الحدوث في حياتنا، فرجل المرور يتحكم في عملية سير السيارات من خلال أخذة لقرار ما ينبع عن هذا القرار إشارات يظهرها للسائقين فيسير البعض ويتوقف الآخرون. وفي بيتك تتحكم في تحركات الأطفال من خلال شروط وضوابط معينة فمن يسير عليها يحصل على أمر ما ومن يخالفها يحرم من هذا الشيء، فأنت تتخذ قراراً بمنع أحدهم جائزة دون الآخر، الشكل 4.1 يوضح المبدأ العام.



شكل 4.1: مبدأ عمل جمل الاختيار

فمفهوم اتخاذ قرار يظهر دائماً عندما يكون الإنسان ممن لديه خيارات مختلفة وعليه أن يختار أحدها، في هذه الحالة يقوم بعض الحسابات الرياضية أو المنطقية وبناءً على نتيجتها يتتخذ قراراً ما، فالمحاضر فمثلاً عندما يسأل عن الطلبة الناجحين يبدأ في البحث في سجل الدرجات ويقول: (من تخطى الخمسين درجة أو ساواها فهو ناجح) ويبداً في البحث بناءً على هذا الشرط المنطقي.

وكذا قد أشرنا إلى هذا المفهوم في الباب الثاني أثناء حديثنا عن خرائط التفرع بأنواعه المختلفة، وناقشت مجموعة من الخوارزميات المتعلقة بهذا المفهوم مثل تعبئة خزان الماء وكذلك تحديد كون الملعب دولي أم لا وغير ذلك

من التطبيقات، هذه التطبيقات جميعها ستحتاج إلى استخدام العلاقات المنطقية التي تعرفنا عليها في الفصل السابق سواء البسيطة أو المركبة.

#### 4.4 الجمل البرمجية للاختيار

توفر لغة جافا مجموعة من الجمل البرمجية التي يتم استخدامها لاتخاذ القرار وهي في مجموعها تتكون من أداة الشرط والشرط وجواب الشرط، وهي متعددة الأنواع بتنوع عدد الخيارات المتاحة لجواب الشرط، ويتوفر منها أربع جمل موضحة بإيجاز في الجدول 4.7، وسيتم التطرق بالشرح لكل واحدة منها في الفصل الحالي.

النوع	استخدامها
if	عند اهتمامنا بخيار واحد (جواب شرط) ولدينا شرط واحد أو أكثر.
if ... else	عند اهتمامنا بخيارات متافقين ولدينا شرط واحد أو أكثر.
if ... else If شرط متافق	عند اهتمامنا بأكثر من خيارات متافقين، يمكن الوصول لأحدhem بفحص أكثر من شرط متافق
Nested true if	عند اهتمامنا بأكثر من خيار متافق، يمكن الوصول لأحدhem بفحص أكثر من شرط متافق
switch	عند وجود شروط متعددة متافقة لها فقط علاقة بأرقام محددة

جدول 4.7: موجز الجمل البرمجية للاختيار

#### 4.4.1 الجملة الشرطية (if) أحادية الاختيار :

هذه أبسط جمل الاختيار حيث أنها أحادية الاختيار بحيث لا يتم اتخاذ قرار أو تنفيذ تعليمات محددة إلا إذا كان الشرط منطقي صحيح وإلا فإن المترجم يتتجاهل جواب الشرط ويقوم بتنفيذ الجملة التي جملة (if). في الباب الثاني أورينا الشكل 2.7 ليوضح مبدأ عمل الجملة، بينما الشكل 4.2 يوضح كيفية كتابة هذه الجملة في لغة جافا.



شكل 4.2: القاعدة الأساسية إلى كتابة جملة if أحادية الاختيار

ومن أمثلة استخدام هذه الحالة في الحياة العامة أن تسأل اثنين من زملائك: أيهما أكبر، ففي هذه الحالة أنت فقط مهتم بالأكبر ولا تهتم بال الخيار الآخر وهو الأصغر، ومثال آخر: أن تقول "إن كان الجو مشمساً سأذهب للبحر" فأنت هنا لم تذكر ما الذي ستفعله إن كان الجو غير ذلك مما يعني اهتمامك فقط بالحالة الصحيحة. وتعد هذه التطبيقات قليلة فنحن غالباً عندما نأخذ قراراً يهمنا شفهياً.

للمزيد عن مفهوم هذه الحالة يمكنك مراجعة الفصل 2.4.2 حول خرائط التفرع.

#### مثال توضيحي 4.1:

اكتب برنامج يستقبل من المستخدم رقمين صحيحين ويطبع أيهما أكبر.

```

1 import javax.swing.JOptionPane;
2 public class BigOfTwo {
3     public static void main(String[ ] args) {
4         String input = JOptionPane.showInputDialog("Enter 1'st");
5         int no1 = Integer.parseInt(input);
6         input = JOptionPane.showInputDialog("Enter 2'nd");
7         int no2 = Integer.parseInt(input);
8         if (no1>no2)
9             JOptionPane.showMessageDialog(null,"no1 is larger than no2");
10        if (no1<no2)
11            JOptionPane.showMessageDialog(null,"no2 is larger than no1");
12        JOptionPane.showMessageDialog(null,"Thanks");
13    }
14 }
```

لاحظ معي في السطر 8 وبعد أن قمنا بإدخال الأرقام من المستخدم، استخدمنا جملة الشرط (if) للمقارنة بين القيمتين (no2,no1) من خلال العلاقة المنطقية (>) لنحدد أيهما أكبر، فإذا كان الشرط (no1>no2) صحيحاً فهذا يعني أن no1 أكبر من no2 فطبع الجملة في السطر 9. فإذا لم تكن صحيحة، يتجاهل المترجم هذه الجملة تماماً وكأنها ليست موجودة ثم ينتقل إلى الجملة التي تليها، وهذا يعني أنه لن ينفذ الجملة في السطر 9 وبالتالي لم يعد التنفيذ متسلسلاً (فماذا أصبح نمط التنفيذ؟)، ينتقل بعد ذلك إلى الجملة الشرطية في السطر 10 ويكرر معها الأمر ذاته. ولكن جملة الإخراج في السطر 12 سيتم طباعتها على كل حال وذلك لأنها غير تابعة (مرتبطة) بأي جملة شرطية.

**جواب الشرط للجملة (if)** قد يكون جملة واحدة أو أكثر من جملة محاطة في بدايتها بـ { وفي نهايتها بـ } ويصبح تنفيذهم مرهوناً بصحة الشرط فينفذ أو يتجاهل سوياً.



شكل 4.3: تنفيذ المثال التوضيحي 4.1

في المثال السابق لاحظت أن صحة الشرط كان يؤدي إلى تنفيذ جملة واحدة مرتبطة به وهي الجملة الأولى التي تليه، بينما يمكننا أن نجعل مجموعة من الجمل مرتبطة بشرط ما من خلال جعلهم جملة برمجية واحدة أو ما يعرف باسم (Block) كما يلي:

```

1 if (no1>no2)
2 {
3     JOptionPane.showMessageDialog(null,"no1 is larger than no2");
4     JOptionPane.showMessageDialog(null,"no2 is smaller than no1");
5 }
6 JOptionPane.showMessageDialog(null,"Thanks");

```

في الحالة التي عرضناها تجد أن الشرط في السطر رقم 1 إذا كان صحيحاً سيتم تنفيذ الجملتين في السطر 3 و 4 لأن كلا الجملتين تم وضعهما ضمن الأقواس {} الظاهرة في السطرين 2 و 5 وهذه الأقواس تجعل المترجم يفهم أن كل ما بداخلها سيتم تنفيذه في حال كان الشرط صحيحاً وسيتم تجاهله بالكامل في حال كان الشرط خاطئاً. بينما الجملة البرمجية في السطر رقم 6 غير مرتبطة إطلاقاً بالشرط وبالتالي يتم تنفيذها على أي حال.

قبل أن ننتقل إلى النوع الآخر من جمل (if)، أعد النظر إلى مثال 4.1 والشكل 4.1 وانتبه إلى أن الشرط في جملة (if) لا يتم وضع فاصلة منقوطة (;) بعده إطلاقاً، في حال تم وضعها فإن المترجم سيعتبر أن الجملة انتهت وسيقوم بتنفيذ كافة الجمل التي تلي جملة (if) بغض النظر عن نتيجة المقارنة المنطقية. كذلك في حال تم عمل مقارنة منطقية بين متغير وأخر ولم يحتوي أحدهما أو كلاهما على قيمة فإن هذا يسبب خطأ برمجياً يمنع البرنامج من التنفيذ.

**خطأ:** وضع فاصلة منقوطة بعد الشرط المنطقي لجملة if يعتبر خطأ منطقى يلغى قيمة الشرط ويعودي إلى تنفيذ كافة الجمل دون أي قيود، بينما وضع الفاصلة بعد الأقواس {} يسبب خطأ برمجي، وهذه الأخطاء تعم على كافة جمل if.



```
if (area > 20); ← فاصلة منقوطة في موضع خاطئ
{
    JOptionPane.showMessageDialog(null, "The area is: " + area);
    JOptionPane.showMessageDialog(null, "Jazak Allah Khayran ");
}
```

هذه الجمل بهذه الطريقة سيتم تنفيذها على أي حال وهذا أمر لم يقصده المبرمج

شكل 4.4: أخطاء يحذر الوقوع فيها

#### 4.4.2 الجملة الشرطية (if ... else) ثانية الاختيار:

في هذا النوع من جمل الاختيار يكون متاحاً أمامنا خيارات ولنا أن نختار أحدهما بناء على الشرط، وقمنا بتوضيح مفهوم هذا النوع عند الحديث عن الخرائط ثنائية التفرع في الفصل 2.4.2 والشكل 2.7. ونكتب هذه الجملة بلغة جافا على حالتين، في الشكل 4.5 يظهر لنا كيفية كتابتها عندما يكون جواب الشرط عبارة عن جملة واحدة فقط، مثلاً نقول إذا كان معدل طالب الدبلوم أكبر من أو يساوي 50، نطبع له ناجح وإلا نطبع له راسب. في هذه الحالة فإن جواب الشرط يعتبر جملة واحدة وهي (نطبع له ناجح) أو (نطبع له راسب) وسيعرفها المترجم ويفندها كونها أول جملة برمجية بعد الشرط أو بعد (else).

بينما الشكل 4.6 يوضح كيفية كتابة الجملة (if ... else) عندما يكون جواب الشرط أكثر من أمر برمجي، ويفهم المترجم هذه الحالة من خلال وجود الأقواس {}, وهي التي تحتوي الجمل.

من دون إحاطة الجمل في جواب الشرط المكون من أكثر من جملة برمجية بالأقواس فإن المترجم يعتبر أن جواب الشرط فقط هي الجملة الأولى بعد if أو else وما بعد ذلك يتم تنفيذه على أي حال دون إظهار أخطاء و هذا ما يعرف بالخطأ المنطقي.



**if (الشرط المنطقى)**

عملية واحدة تنفذ إذا كان الشرط صحيحًا

**else**

عملية واحدة تنفذ إذا كان الشرط خاطئًا

شكل 4.5: صيغة كتابة الجملة if ... else إذا كان جواب الشرط جملة واحدة

**if (الشرط المنطقى)**

{

مجموعة من العمليات تنفذ كجملة واحدة إذا كان الشرط صحيحًا

}

**else**

{

مجموعة من العمليات تنفذ كجملة واحدة إذا كان الشرط خاطئًا

}

شكل 4.6: صيغة كتابة الجملة if ... else إذا كان جواب الشرط أكثر من جملة

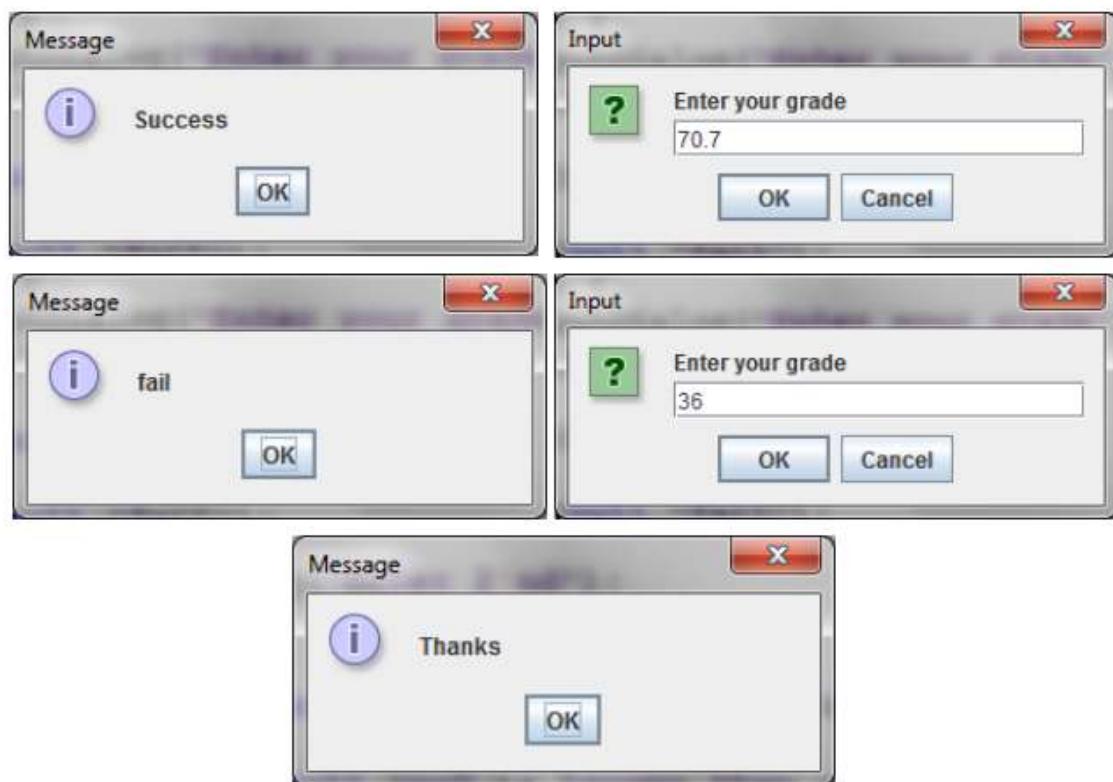
**مثال توضيحي 4.2:**

اكتب برنامج يستقبل من الطالب معدله الدراسي ليطبع "ناجح" إن كان أكبر من أو يساوي 50 وإن كان أقل يطبع "راسب".

```

1 import javax.swing.JOptionPane;
2 public class Grades {
3     public static void main(String[ ] args){
4         String input = JOptionPane.showInputDialog("Enter Your grade");
5         float grade = Integer.parseInt(input);
6         if (grade>=50)
7             JOptionPane.showMessageDialog(null,"Success");
8         else
9             JOptionPane.showMessageDialog(null,"fail");
10    JOptionPane.showMessageDialog(null,"Thanks");
11 }
12 }
```

في هذا المثال، لاحظ معنا احتجنا في السطر رقم 1 لإضافة الصنف (`JOptionPane`) من المكتبة `java.awt` بالأمر (`import`) بينما استخدامنا الدالة (`showInputDialog`) في السطر رقم 4 لاستقبال قيمة من المستخدم وقمنا بتخزينها في المتغير (`input`) الذي هو من نوع (`String`) (هل تعرف لماذا؟)، بعد ذلك استخدمنا دالة التحويل من نص إلى رقم عشرى في السطر رقم 5 وقمنا بتخزين الناتج وهو رقم عشرى في المتغير (`grade`) وأصبح الآن بإمكاننا أن نقوم ببناء الشرط المنطقي (`grade >= 50`) إلى كتابة جملة الاختيار ونضع جواب الشرط، إن كان الشرط صحيحًا طباعة (`Success`) وإن كان خاطئًا طباعة (`Fail`). بينما في السطر رقم 10 سيطبع البرنامج على أي حال الكلمة (`Thanks`) لأنها جملة طباعة غير مرتبطة بالشرط إطلاقاً كما شاهد في الشكل 4.7.



شكل 4.7: تنفيذ المثال التوضيحي 4.2 في الحالتين

### مثال توضيحي 4.3:

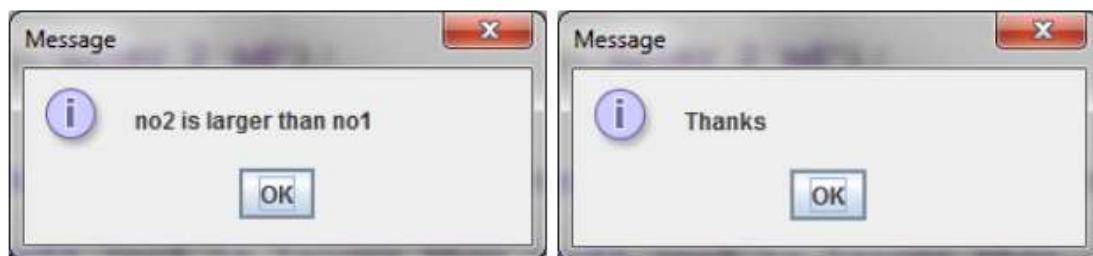
- اكتب برنامج يستقبل من الطالب معدله الدراسي، فإن كان أكبر من أو يساوي 50
- يطبع "Success" ومعدله و "You can be in the next semester"
  - وإن كان أقل من ذلك يطبع "Good Luck"

الآن حاول معي التعديل على هذا المثال لنجعل جواب الشرط أكثر من جملة، علماً بأنه ليس شرطاً أن يكون عدد الجمل البرمجية في جواب الشرط متساوياً في الحالتين (*False*, *True*) ، وبالتالي قد تحتاج إلى وضع أقواس للأولى ولا أضع للثانية، ورغم ذلك إن وضعت أقواس في الحالتين لا يضر .

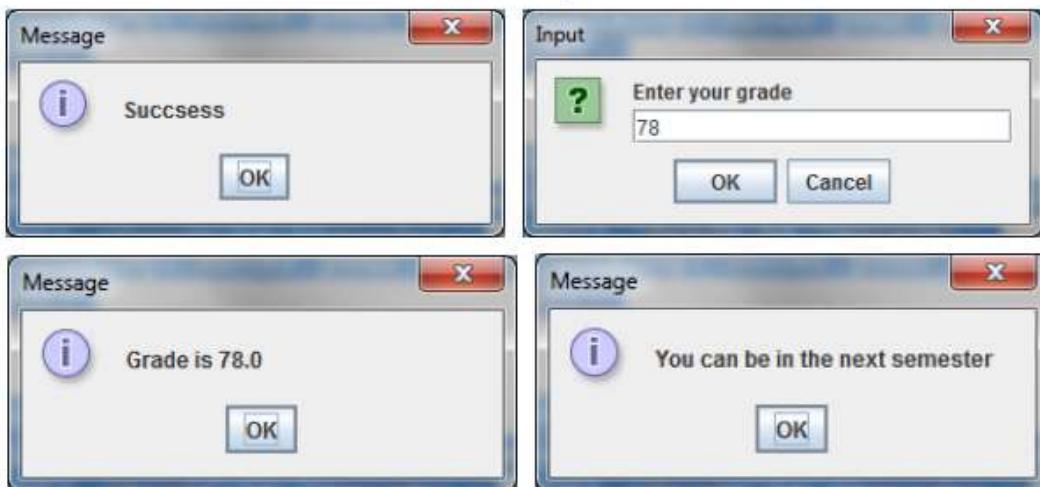
```

1 import javax.swing.JOptionPane;
2 public class grades2 {
3     public static void main(String[ ] args){
4         String input = JOptionPane.showInputDialog("Enter your grade");
5         float grade = Float.parseFloat(input);
6         if (grade>=50){
7             JOptionPane.showMessageDialog(null,"Success");
8             JOptionPane.showMessageDialog(null, "Grade is"+ grade);
9             JOptionPane.showMessageDialog(null," You can be in the next semester ");
10        }
11    else
12        JOptionPane.showMessageDialog(null,"Good Luck");
13    } }
```

لاحظ عندما كان جواب الشرط أكثر من أمر برمجي، كان من الضروري أن نضع الأقواس {}, بينما في الطرف الثاني لم يكن جواب الشرط سوى أمر واحد فلم يكن لزاماً علينا أن نستخدم الأقواس.



شكل 4.8: جزء من تنفيذ المثال التوضيحي 4.3



شكل 4.9: تنفيذ المثال التوضيحي 4.3 عندما يكون الطالب ناجحاً



شكل 4.10: تنفيذ المثال التوضيحي 4.3 عندما يكون الطالب راسبًا

الآن حاول أن تقوم ببعض التعديلات على مثال 4.3 كما يلي:

- احذف أحد طرفي الأقواس {} في جواب الشرط، سكتشف أنه سيعطيك خطأ برمجي يمنع تنفيذ البرنامج.
- بينما إذا قمت بحذف كامل الأقواس {} في جواب الشرط، فإن البرنامج سيعطيك خطأ برمجياً لأنه من غير الصحيح أن تفصل (else) عن (if).
- أدخل قيمة ناجح صحيحة مثل (78) ثم تابع ماذا سيطبع للمعدل؟ - فسر النتيجة.

في بعض التطبيقات، قد تحتاج لاتخاذ قرار بناءً على نتيجة أكثر من شرط، فيتم ربط نتائج الشروط بالعلاقات التي قمنا بشرحها في الفصل 4.2 وفي الجدول 4.2. المثال التالي نستخدم فيه هذا المفهوم.

#### مثال توضيحي 4.4:

اكتب برنامج يستقبل من طالب الثانوية العامة معدله الدراسي، فان كان أكبر من أو يساوي 60 وكان كذلك من القسم العلمي يطبع له (Accepted in IT) وإلا يطبع له (Accepted in Management).

#### 1. import javax.swing.JOptionPane;

```

2. public class CollegeAccepted{
3.     public static void main(String[ ] args) {
4.         String input = JOptionPane.showInputDialog(null,"Enter grade");
5.         float grade = Float.parseFloat(input);
6.         input = JOptionPane.showInputDialog(null,"Enter specific");
7.         if((grade>=60)&&(input.compareTo("علمى")==0))
8.         {
9.             JOptionPane.showMessageDialog(null,"Accepted in IT");
10.        }
11.        else
12.            JOptionPane.showMessageDialog(null,"Accepted in Management");
13.    }
14. }

```

لاحظ معي أن قبول الطالب في إحدى الكليتين يعتمد على شرطين هما قيمة معدله الدراسي ونوع الثانوية العامة، وبالتالي لا بد من بناء شرط مركب باستخدام العلاقة (`&&`) لأن تسجيل الطالب في كلية تكنولوجيا المعلومات يُشترط له صحة الشرطين سوياً. وبالتالي في السطر رقم 7 قمنا بربط الشرطين الأول وهو لفحص المعدل والشرط الثاني هو لفحص نوع الثانوية العامة الذي تخرج منها.

`if((grade>=60)&&(input.compareTo("علمى")==0))`

كما تلاحظ، الشرط الثاني يتم فيه فحص تساوي قيمتين من النوع نص، فنحن نفحص إن كانت القيمة النصية المدخلة تساوي القيمة "علمى"، وهذه المقارنة في لغة جافا لا تتم من خلال إشارة فحص التساوي (`==`) وإنما تتم من خلال الدالة (`compareTo`) وهي إحدى الدوال التي يتم تطبيقها على النصوص والمقارنة بها تعيد ثلاثة قيم صحيحة، لكل قيمة معنى كما في الشكل 4.11.

فإن أعادت الدالة القيمة (0) هذا يعني أن كلاً من القيمتين متساويتين، أما إن أعادت القيمة (-1) فهذا يعني أن القيمة التي استدعت الدالة تعتبر أقل من القيمة الممررة للدالة أما إذا كانت القيمة (1) فهذا يعني أن القيمة التي استدعت الدالة أكبر من القيمة الممررة للدالة.



شكل 4.11: مفهوم الدالة `compareTo`

هذا الشرط المركب لا يكون صحيحاً في إجمالي إلا إذا كان كلا الشرطين صحيحةً كما تابعنا ذلك في الجدول 4.3. والسؤال الذي قد يسأله أحد الدارسين، كيف لي أن أعرف ما هي العلاقة التي يجب استخدامها لربط الشروط؟ فهل أربطها بـ `(&&)` أو `(( ))`، والإجابة أنَّ هذا من العناصر التي يتم تحديدها في مرحلة التحليل، والتي يحدد فيها محل النظم خطوات التنفيذ وشروط كل خطوة إن احتاجت، وتحدد المسألة هل يجب أن يكون كل الشروط صحيحةً أم بعضها وهكذا، ففي هذه المسألة اشترط لدخول قسم (IT) أن يكون الطالب حاصل على معدل أكبر من أو يساوي 60 وكذلك يكون من القسم العلمي، فإن كان أحد الشرطين خاطئاً وبالتالي لن يتمكن الطالب من دخول هذا القسم.

يتوفر في لغة جافا، دوال متعددة لفحص القيم النصية ومنها `compareTo` ، `equals` ،  
`compareToIgnoreCase`، `equalsIgnoreCase`



عدم وضع الشرطين عند كتابة الشرط المركب بين أقواس يعتبر خطأ برمجي، كما يلي:  
`if ((grade>=60)&&(input.compareTo("علمى")==0))`



عدم احتواء المتغير في الشرط على قيمة يجعل المترجم يظهر خطأ برمجي يمنع التنفيذ.



في حالات أخرى، يرتبط التنفيذ بصحبة شرط واحد فقط من مجموعة شروط ومثال ذلك أن نقول أن طالب مرحلة الببلوم يمكنه الانتقال للدراسة في تخصص بكالوريوس الهندسة إذا تجاوز معدله الجامعي 90% أو أنه حصل في الامتحان الشامل على معدل أكبر من أو يساوي 90%， هذه الحالة تحتاج فيها إلى شرط مركب من شروطين يربطهما الإشارة `(( ))` وذلك لأنَّ تنفيذ الجملة البرمجية مرتبط بصحبة أحد الشرطين على الأقل، في المثال التالي سنقوم بحل هذه المشكلة البرمجية.

#### مثال توضيحي 4.5:

اكتب برنامج يطبع كلمة Accepted in Engineering (Accepted in Engineering) لطالب الببلوم الذي معدله الجامعي على الأقل 90% أو حاصل في الامتحان الشامل على تقدير 90% على الأقل.

```
1. public class orChoice {
2. public static void main(String[ ] args) {
3. String input = JOptionPane.showInputDialog(null,"Enter grade");
```

```

4. float GPA = Float.parseFloat(input);
5. input = JOptionPane.showInputDialog(null, "Enter shamel grade");
6. float shamel = Float.parseFloat(input);
7. if((GPA>=90)|| (shamel>=90))
8. JOptionPane.showMessageDialog(null, "Accepted in Engineering");
9. }

```

مطلوب منك في هذا المثال التركيز على استخدامنا للشرط المركب في حال استخدام العلاقة (||) مع الانتباه إلى ضرورة إحاطة الشرطين بالأقواس () حتى لا يظهر خطأ برمجي.

أتراك لك فرصة تجربته بحالاته المختلفة كأن تعطيه أحد الشرطين صحيحاً والآخر خاطئاً ثم تعكس الحالة ثم تعطيه الشرطين صحيحين ثم الشرطين خاطئين وستجد أنك تسير على خطى جدول الحقيقة للعلاقة (||) كما في الجدول 4.4.

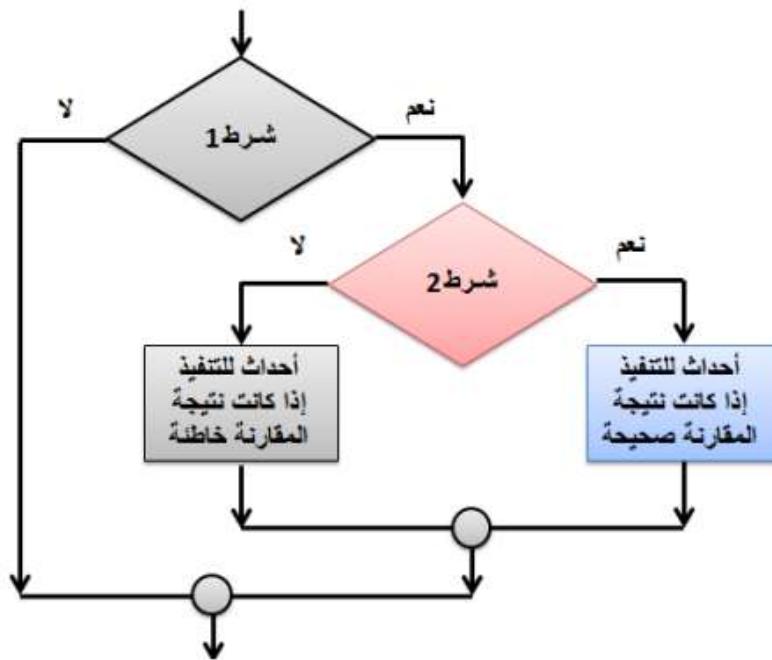
#### 4.4.3 جمل الاختيار المتداخلة

كثيراً من التطبيقات البرمجية في حياتنا تعتمد على فحص شروط متتالية، أي صحة الشرط تأخذ بنا لفحص شرط آخر حتى نصل لاتخاذ القرار النهائي، الشكل 4.11 يوضح مبدأ هذا النوع من جمل الاختيار.

لاحظ في الشكل 4.12 فإنَّ الجمل لا تُنفذ إلا إذا فحصنا أكثر من شرط متتالي، وجمل الاختيار من هذه الحالة على نوعين هما:

1. جمل اختيار متداخلة معتمدة على شروط مترافقه (*if ... else if*): وهي جملة اتخاذ قرار تحتوي على أكثر من شرط يتم اختبارهم من خلال جملة (*if*) أساسية يتفرع منها شروط أخرى تظهر عندما يكون الشرط الأساسي خاطئ، فكلما كان الشرط خاطئاً تم اللجوء لاختبار شرط آخر لعدة حالات وإلا فلا يتم الاستمرار في فحص الشروط.

2. جمل اختيار متداخلة معتمدة على الشروط المترافقه (*if ... if*): وهي جملة اتخاذ قرار تحتوي على أكثر من شرط واحد داخل الآخر، وهذه الشروط يتم اختبارهم من خلال جملة (*if*) أساسية يتفرع منها شروط أخرى تظهر عندما يكون الشرط الأساسي صحيح، فكلما كان الشرط صحيحاً تم اللجوء لاختبار شرط آخر لعدة حالات، الشكل 4.12 يوضح هذا النمط.



شكل 4.12: مفهوم جمل الاختيار المتداخلة

ومثال ذلك أنتا إذا أردنا معرفة تقدير طالب معين هل هو ممتاز أم جيد جداً أو غير ذلك؟، نسألة عن معدله الدراسي شيئاً فشيئاً فإن كان أكبر من 50، تم سؤاله هل هو أكبر من 60؟ وهكذا بالتدليل حتى نصل لنسبة الصحيحة لمعرفة تقديره وهذه الصيغة يمكن الوصول لها من خلال نمط (if ... if ...) وتعرف أحياناً باسم (nested if) أما إذا تم سؤاله من الأعلى للأدنى كأن نقول له هل معدلك أكبر من 90؟ فيقول لا، ثم نسألة هل أكبر من 80؟ فيقول لا وهكذا، فهنا نحن نختبر شروطاً تظهر عندما يكون الشرط السابق خاطئاً فننتقل لاختبار شرط آخر وهذا ما يعرف بالنمط (if ... else if) وننطقه (إلا إذا).

ويقصد بالشروط المترافق أي الشروط التي تكون جميعاً لها إجابة واحدة مترافق (false,true)، بينما المترادفة هي تلك التي تكون متعارضة ولا يمكن أن تتوافق على إجابة واحدة فمثلاً نتيجة الطالب لا يمكن أن تكون "تاجح" و"راسب" في آن واحد بينما في المترافق فيمكن للطالب أن يكون من "دولة فلسطين" ثم "قطاع غزة" ثم "خان يونس" فلا تعارض في قيم الشروط هنا.

الشكل 4.13، 4.14 يوضح كيفية التعبير عنهما برمجيًا في المثال 4.6 ثم في المثال 4.7 نقف برمجيًا مع هذا النمط.

```

if(شرط الأول)
{
    مجموعة من العمليات تنفذ عند تحقق الشرط الأول
}
else if(شرط الثاني)
{
    مجموعة ثانية من العمليات تنفذ عند عدم تتحقق الشرط الأول و تتحقق الثاني
}
و هكذا تكتب المزيد من الجمل حسب الحاجة ....
else
{
    مجموعة ثلاثة من العمليات تنفذ عند عدم تتحقق الشرط السابقه وليس ايجاريا
}

```

شكل 4.13: مبدأ كتابة جمل الاختيار المتداخلة بالشروط المتافقنة

```

if(شرط الأول)
if(شرط الثاني)
if(شرط الثالث)
مجموعة من العمليات (عملية واحدة) تنفذ عند تتحقق
الشروط الأول و الثاني و الثالث و قد يكون شرطًا متدخلاً آخر
else
جمل تنفذ إن كان الشرط الأول و الثاني صحيحين لكن الثالث خاطئاً
else
جمل تنفذ إن كان الشرط الأول صحيحًا لكن الثاني خاطيء
else
جمل تنفذ إن كان الشرط الأول خاطئاً

```

شكل 4.14: مبدأ كتابة جمل الاختيار المتداخلة بالشروط المتفقة

جمل اتخاذ القرار المتداخلة بكل النوعين هي ببساطة أحد جمل اتخاذ القرار السابقة إلا أن جواب الشرط فيها عبارة عن شرط جديد، فان كان الشرط الجديد جاء كجواب شرط للحالة (نعم) كان النمط `if ... if` ... if)، وإن كان الشرط الجديد جاء كجواب شرط للحالة (لا) كان النمط `if ... else if` ... if)، فكر بهذه الصورة سيكون الأمر أكثر سهولة.



**مثال توضيحي 4.6:**

اكتب برنامج ليستقبل من الطالب معدله الجامعي ثم يطبع له البرنامج تقديره حسب ما يلي:

- طباعة " Excellent " – إذا كانت العلامة أكبر من أو تساوي 85 .
- طباعة " Very Good " – إذا كانت العلامة أكبر أو تساوي 75 وأقل من 85 .
- طباعة " Good " – إذا كانت العلامة أكبر من أو تساوي 65 أقل من 75 .
- طباعة " Passed " إذا كانت أكبر من أو تساوي 50 وأقل من 65 .
- وطباعة " Failed " إذا كانت أقل من 50 .

```

1 import javax.swing.JOptionPane;
2 public class Grades {
3     public static void main(String[ ] args) {
4         String input = JOptionPane.showInputDialog("Enter your gpa");
5         float gpa = Float.parseFloat(input);
6         if (gpa>=50)
7             if(gpa>=65)
8                 if (gpa >=75)
9                     if (gpa>= 85)
10                         JOptionPane.showMessageDialog(null,"Excellent");
11                     else
12                         JOptionPane.showMessageDialog(null,"V Good");
13                 else
14                     JOptionPane.showMessageDialog(null,"Good");
15             else
16                 JOptionPane.showMessageDialog(null,"Pass");
17         else
18             JOptionPane.showMessageDialog(null,"fail");
19     }
20 }
```

**عدم وضع else لأحد الشروط، قد يظهر خطأ منطقي في النتائج الظاهرة (تجربة الان)**



لاحظ عزيزي الدارس في هذا المثال، قمنا باستخدام جملة الاختيار المتداخلة بشرط متوافق، الواحدة داخل الأخرى، فإن كان الطالب معدله أكبر من أو يساوي 50 فمما بالسؤال هل معدله أكبر من أو يساوي 65 وهذا

فإن كانت الإجابة نعم، زدنا في السؤال حتى أكبر من أو يساوي 85. فإن كانت الإجابة نعم طبعنا له تقديره الممتاز، وإلا استخدمنا الأمر (else) فطبعنا ما يقابلها، فمثلاً (إن كان المعدل أكبر من أو يساوي 65 ولكن ليس أكبر من أو يساوي 75) فهذا يعني معلده بين 65 و 75 وبالتالي أطبع له في الموضع (else) تقديره جيد وهذا.

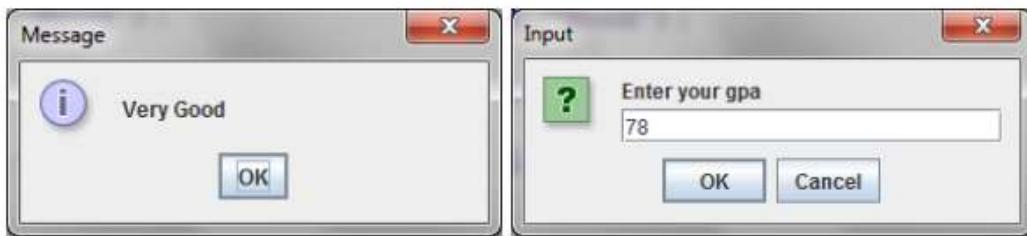
#### مثال توضيحي 4.7:

اكتب برنامج مكافئ لمثال 4.6 مستخدماً جمل الاختيار المتداخلة بشروط متناقضة.

```

1 import javax.swing.JOptionPane;
2 public class GradesByElseif {
3     public static void main(String[ ] args) {
4         String input = JOptionPane.showInputDialog("Enter your gpa");
5         float gpa = Float.parseFloat(input);
6         if (gpa>=85)
7             JOptionPane.showMessageDialog(null,"Excellent");
8         else if(gpa>=75)
9             JOptionPane.showMessageDialog(null,"Very Good");
10        else if (gpa >=65)
11            JOptionPane.showMessageDialog(null,"Good");
12        else if (gpa >=50)
13            JOptionPane.showMessageDialog(null,"Pass");
14        else
15            JOptionPane.showMessageDialog(null,"fail");
16    }
17 }
```

في هذه الطريقة نبدأ في اختبار الدرجات من أعلى للأسفل، أو من الخارج للداخل، فنسأل في كل مرة بالجملة *if* فإن كانت الإجابة لا، نختبر شرطًا جديداً بجملة (*if*) جديدة وهذا. والسؤال في هذه الطريقة يُسأل هكذا، هل المعدل أكبر من أو يساوي 85؟ إذا كانت الإجابة نعم اطبع كذا، وإلا هل معدلك أكبر من أو يساوي 75؟ إذا كانت الإجابة نعم اطبع كذا، وإلا هل... وهكذا فإن كل حالة تحتاج متنًا إلى شرط إلا أنه شرط متداخل متناقض.



شكل 4.15: ناتج مثل 4.6 و كذلك 4.7

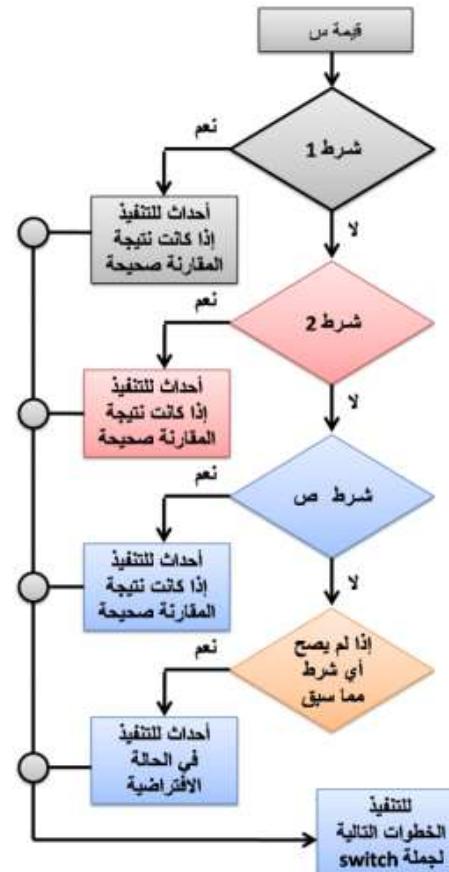
عكس الإشارات المنطقية مثل => يصبح <= يعتبر خطأ برمجي، كذلك عند فصلهما.



#### 4.4.4 جملة switch متعددة الاختيارات

تعتبر من الصور الأخرى لجمل الاختيار، وهي تعتمد على وجود عدة خيارات متقاضة وعليها أن نختار أحدهم بناءً على قيمة محددة يدخلها المستخدم أو تنتج لنا من عملية أخرى. ويظهر مبدأ عملها من الشكل 4.16 كما تظهر طريقة كتابتها بلغة جافا بالشكل 4.17.

```
switch (x)
{
    case 1 :
        جمل برمجية تنفذ في حال كانت x = الحالة ١
        break;
    case 2 :
        جمل برمجية تنفذ في حال كانت x = الحالة ٢
        break;
    case 3 :
        جمل برمجية تنفذ في حال كانت x = الحالة ٣
        break;
    default:
        جمل برمجية تنفذ في حال كانت x لا تساوي
        أي حالة من الحالات السابقة
        break;
}
```



شكل 4.17: طريقة كتابة الجملة switch بلغة الجافا

شكل 4.16: مبدأ عمل الجملة switch

كما تلاحظ في الشكلين السابقين، فإن جملة **(switch)** تحتوي على مجموعة حالات (شروط) وب مجرد أن تقابل القيمة المدخلة مع حالة من هذه الحالات، يتم تنفيذ الجمل البرمجية المدرجة ضمنها، ثم مباشرةً ينتقل التنفيذ إلى خارج جملة **(switch)** من خلال جملة برمجية تسمى **(break)** وهي جملة برمجية تجبر المترجم على الخروج إلى خارج القطعة التي هي بداخلها. كذلك فإن جملة **(switch)** تحتوي على حالة افتراضية تعرف باسم **(default)** وهي حالة يتم تنفيذها في حال لم تقابل القيمة المدخلة مع أي حالة من الحالات السابقة.

أعتقد أن أحد الدارسين الآن يود أن يقول أن جملة **(switch)** في منطقها تشبه تماماً جملة **(if ... else if)** وأنا أحبيه على تفكيره بهذا صحيح إلا أن هناك فرق مهم جداً بينهما وهو أن الشرط في جملة **(if ... else if)** يمكنها التعامل مع الفترات (المدى) باستخدام العلاقات المنطقية مثل **(<, >)** وغير ذلك فتشمل القيم العشرية أيضاً بينما جملة **(switch)** تتعامل فقط مع قيم محددة وكأن العملية الوحيدة التي يمكن استخدامها للمقارنة هي المساواة، فإنه عند كل جملة **(if (I == x))** كأنك تقول **(case 1:)** وهذا يتكرر لكافة الحالات.

#### مثال توضيحي 4.8:

اكتب برنامج يستقبل من المستخدم أحد الأرقام من 1 إلى 7 ويطبع له اليوم المكافئ

مثل 1 : السبت، 2: الأحد، ...

```

1 import javax.swing.JOptionPane;
2 public class PrintDaysUsingSwitch{
3     public static void main(String[ ] args) {
4         String input = JOptionPane.showInputDialog("Enter num");
5         int num = Integer.parseInt(input);
6         switch(num){
7             case 1:
8                 JOptionPane.showMessageDialog(null,"Saturday");
9                 break;
10            case 2:
11                JOptionPane.showMessageDialog(null,"Sunday");
12                break;
13            case 3:
14                JOptionPane.showMessageDialog(null,"Monday");
15                break;
16            case 4:
17                JOptionPane.showMessageDialog(null,"Tuesday");

```

```

18     break;
19     case 5:
20         JOptionPane.showMessageDialog(null,"Wednesday");
21         break;
22     case 6:
23         JOptionPane.showMessageDialog(null,"Thursday");
24         break;
25     case 7:
26         JOptionPane.showMessageDialog(null,"Friday");
27         break;
28     default:
29         JOptionPane.showMessageDialog(null,"number between 1 To 7");
30         break;    } }

```

هذا المثال يحتاج من المستخدم إلى إدخال قيمة صحيحة، في الأسطر 4 و 5 قمنا باستقبال القيمة وتحويلها إلى النوع الصحيح (قد تحولها إلى نوع صحيح آخر لا يضر)، بعد ذلك نبدأ في تطبيق جملة **(switch)** والتي سيكون شرطها متمركزاً حول القيمة التي أدخلها المستخدم وهي **(num)**، وأن كافة الخيارات التالية تعتبر جزء من الجملة **(switch)** فقد استخدمنا الأقواس **{}** ثم بدأنا في وضع الحالات وهي سبع حالات (1 إلى 7) بالإضافة للحالة الافتراضية، لاحظ معنـى أنـنا في كل حالة من الحالـات نضع الحـدث الذي يـقابلـها وهو جـملـة طـبـاعـة، بالإضافة إلى جـملـة **(break)**، وهـيـ الـتيـ ستـجـعـلـ المـتـرـجـمـ يـتـجـاهـلـ بـقـيـةـ الجـمـلـ وـيـنـقـلـ لـخـارـجـ جـملـةـ **(switch)**. في النـهاـيـةـ وـضـعـهـاـ الـحـالـةـ الـافـتـراـضـيـةـ وهـيـ الـحـالـةـ الـتـيـ سـيـتـتـقـيـذـهـاـ إـذـاـ أـدـخـلـ الـمـسـتـخـدـمـ قـيـمـةـ خـارـجـ الأـقـارـامـ (7ــ1ــ).

**الأنواع التي يمكن لجملة switch التعامل معها هي char, int, short, byte فقط.**



يبقى أن نضع مقارنة موجزة بين جملة **if else ... if** وجملة **switch** في الجدول 4.8.

If else ... if	switch	وجه المقارنة
كافـةـ أنـوـاعـ الـبـيـانـاتـ	تـتـعـالـمـ معـ charـ وـintـ وـshortـ وـbyteـ وـلاـ تـتـعـالـمـ معـ longـ أوـ Stringـ	أنـوـاعـ الـبـيـانـاتـ لـلـشـرـطـ
كافـةـ الـعـلـمـيـاتـ الـمـنـطـقـيـةـ	الـمـساـواـةـ فـقـطـ (==)ـ وـهـيـ تـسـتـخـدـمـ ضـمـنـيـاـ	الـعـلـاقـاتـ الـمـنـطـقـيـةـ
الـأشـمـلـ	-	الـشـمـولـيـةـ

جدول 4.8: مقارنة بين صيغ if و switch



كافحة المشاكل البرمجية التي يمكن حلها بجملة **switch** يمكن أيضًا حلها بجملة **if else** لأنّ الثانية هي الأشمل.

بالإضافة لكافة الجمل السابقة التي يمكننا من خلالها اتخاذ قرار، فإنّ لغة جافا تتمتع بتوفير معامل الشرط أو المعامل الثالثي (**Conditional operator**) ويرمز له بالرمز **(?:)**، حيث يمكننا استخدامه لاستبدال الجملة **(if .. else)** في القطعة البرمجية التالية على نحو بسيط كما يلي:

```
if(a>b)
    max= a;
else
    max =b;
```

باستخدام معامل الشرط، يمكن التعبير عنها كما يلي:

```
Max= (a>b) ? a:b;
```

ومفهومها كالتالي: إذا كان الشرط **(a>b)** صحيحاً يتم اختيار القيمة الأولى بعد المعامل وهي **a** وإلا يتم اختيار القيمة الثانية وهي **b**.

#### 4.5 أمثلة وتدريبات عامة

##### 1. استخدم الكلمات المناسبة لتعبئة الفراغات في الجمل التالية:

- أ- تقوم لغة جافا بتنفيذ التعليمات في الوضع الافتراضي بتنفيذ التعليمات الواحدة تلو الأخرى بسلسل متتالي وهذا ما يعرف باسم \_\_\_\_\_.
- ب- مفهوم استخدام جمل برمجية للتحكم في سير العمليات يعرف باسم \_\_\_\_\_.
- ت- عند احتياج التطبيق لتنفيذ أحداث معتمدة على صحة شرطين أو أكثر نستخدم علاقة الربط \_\_\_\_\_ بينما إن اعتمد على صحة أحد الشروط على الأقل نستخدم \_\_\_\_\_.
- ث- تبديل موضع إشارة = في العلاقة المنطقية => أو => يؤدي إلى \_\_\_\_\_ بينما الفصل بينهما يردي إلى \_\_\_\_\_.
- ج- من التطبيقات التي تحتاج إلى اتخاذ قرار في حياتنا \_\_\_\_\_ و\_\_\_\_\_.

ح- عندما يكون جواب الشرط أكثر من جملة برمجية نستخدم ————— لكي يفهم المترجم هذا الأمر.

## 2. حدد صحة أو خطأ كلاً من العبارات التالية مع التعليل:

- ( ) أ- لفحص المساواة بين رقمين صحيحين نستخدم العلاقة (=)
- ( ) ب- عند بناء شرط مركب بين شرطين خاطئين بالعلاقة <sup>٨</sup> يكون الناتج العام للشرط الخطأ.
- ( ) ت- عند اهتمامنا بخيارات متلاقيتين في وجود شرط واحد يمكن استخدام الجملة *if.. else*.
- ( ) ث- وضع فاصلة منقوطة بعد أقواس الشرط في جمل اتخاذ القرار يؤدي لخطأ منطقى.
- ( ) ج- تستخدم الدالة *compareTo* للتحويل من *String* إلى *boolean*.
- ( ) ح- عدم احتواء المتغير في الشرط على قيمة يجعل المترجم يظهر خطأ برمجي يمنع التنفيذ.

## 3. عُبَّر عن الجمل التالية بلغة جافا:

- أ- إن كان معدله أكبر من أو يساوي 50 سنطبع له "ناجح".
- ب- إن كان المواطن عمره 60 فما فوق ولديه طفلين على الأقل نطبع "سيصرف لك 200 دينار".
- ت- تحويل النص choice إلى نوع صحيح يمكن تخزينه في متغير Num حجمه 1 بايت فقط.

## 4. حدد الأخطاء ونوعها في الجمل البرمجية التالية مع التصحيح:

- أ-
 

```
if (c < 7);
JoptionPane.showMessageDialog (null, “c is less than 7”);
```
- ب-
 

```
if (c < 7)|| (x>8)
JoptionPane.showMessageDialog (null, “Open”);
```
- ت-
 

```
long num = 23;
switch(num){
    case 1:
        JOptionPane.showMessageDialog(null,“Satarday”);
```

```

        Break;
case 2:
    JOptionPane.showMessageDialog(null,"Sunday");
    Break;
default:
    JOptionPane.showMessageDialog(null," should be 1 To 7");
    Break;
}

```

5. اكتب برنامج يحسب مساحة المربع على أن يطبعها إذا كانت أكبر من 20 متراً مربع. (علمًا بأن مساحة المربع تحسب بضرب طول الضلع في نفسه).

6. اكتب برنامج يستقبل ثلاثة درجات حرارة لمدينة خان يونس ويحسب المتوسط الحسابي لهم، ثم يطبع

- الجو حار إن كان المتوسط أكبر من أو يساوي 24

- الجو معتدل إن كان المتوسط أكبر من أو يساوي 18 وأقل من 24

- الجو بارد إن كان المتوسط أقل من 18

7. اكتب برنامج يقرأ من المستخدم رقم صحيح ويطبعه إذا كان من مضاعفات الرقم 3 وإلا يطبع عبارة (Try again).

8. اكتب برنامج يقرأ من المستخدم ثلاثة أرقام صحيحة فيطبع أكبرهم.

9. اكتب برنامج يقرأ من المستخدم اسم مستخدم وكلمة مرور ثم يخزنهم، ثم يطلب من المستخدم إدخال اسم المستخدم وكلمة المرور فإذا كانت مطابقة للأولى يطبع له (welcome) وإلا يطبع له (Try again).

10. اكتب برنامجاً بلغة Java يستقبل رقمًا صحيحاً من المستخدم ثم يطبع هل هو فردياً (odd) أم زوجياً (even).

#### 4.6 تمارين ذاتية الحل

1. حدد الأخطاء ونوعها في الجمل البرمجية التالية مع التصحيح:

أ-

```

if (age >= 65);
    System.out.println("Age greater than or equal to 65");
else
    System.out.println("Age is less than 65");

```

ب-

```

if (gender == 1)
    System.out.println("Woman");
else;

```

تأليف: أ. محمود وهيق الفرا

باب الرابع: جمل الاختيار

```
System.out.println("Man");
```

ثـ

```
if (day=="Friday") || (hour >= 14)
System.out.println("Shop is Closed");
else
System.out.println("Welcome");
```

2. اكتب برنامجاً يستقبل رقمين من المستخدم ثم يطبع هل الثاني من مضاعفات الأول أم لا.

3. اكتب برنامجاً يستقبل من المستقبل رقمين ثم يطبع نتائج كافة المقارنات الممكنة بينهم.

4. اكتب برنامجاً يستقبل من المستخدم ثلاثة أرقام ثم يطبع أكبرهم وأصغرهم وأوسطهم.

5. اكتب برنامجاً يستقبل من المستخدم رقماً طوله 5 خانات ثم يطبع الخانات بين كل واحد والآخر مسافة.

مثال: 87453 تصبح 3 4 5 8 (مساعدة: استخدم العمليات  $\% 1$  إلى حل المسألة)

6. مستخدماً الجملة البرمجية `switch`, اكتب برنامجاً يحاكي إشارات المرور بحيث يطبع "قف" إذا كانت الإشارة حمراء، ويطبع "استعد للسير" إن كانت الإشارة صفراء، ويطبع "يمكنك السير" إن كانت الإشارة خضراء، بينما إن لم يكن هناك لو يطبع له "إشارات المرور معطلة أعط حق الأولوية للغير".

7. إحدى المكتبات العامة أعلنت عن مشروع لدعم الطالب الجامعي بخصم على قيمة المشتريات على النحو التالي:

- 50% إن كان تقديره الجامعي "امتياز".
- 30% إن كان تقديره الجامعي "جيد جداً".
- 20% إن كان تقديره الجامعي "جيد".
- 10% إن كان تقديره الجامعي "مقبول".
- 5% إن كان فقط ناجحاً.

اكتب برنامجاً لتطبيق هذا المشروع مستخدماً ما تعلمنه في هذا الباب.

8. إحدى المؤسسات التعاونية، قررت البدء في مشروع لدعم التعلم الإلكتروني في الجامعات الفلسطينية، على أن يتم منح كل مؤسسة مبلغاً قدره 50000 دينار بشرط أن تتتوفر في المؤسسة الشروط التالية سوياً:

- لديها 150 محاضراً لهم صفحات إلكترونية فعالة.
- لديها نظام تعليمي إلكتروني تفاعلي (EISL) عدد المشتركين فيه 2000 طالب.
- عدد الطلبة والمحاضرين الذين يستخدمون البريد الإلكتروني للراسلات الجامعية 3000 طالب.

اكتب برنامجاً لتسيير هذا المشروع مع ذكر خطوات التفكير والتخطيط ثم البرنامج.

9. اكتب جملة if لزيادة نسبة الدفعات (pay) لمؤسسة تجارية بنسبة 3% إذا كان حجم البيع (Score) أكثر من 10.000 دينار ولا تزيد بنسبة 1% فقط.

10. في القطعة البرمجية التالية، افترض أن كلاً من  $x$  و  $y$  لهم القيم التالية:

قيمة $y$	قيمة $x$	الحالة
2	3	الأولى
4	3	الثانية
2	2	الثالثة

ما هي مخرجات القطعة في كل حالة؟

```
if (x > 2) {
    if (y > 2) {
        z = x + y;
        System.out.println("z is " + z); }
    else
        System.out.println("x is " + x); }
```

11. اكتشف الخطأ في القطعة البرمجية التالية:

```
if (score >= 60.0)
    System.out.println("D");
else if (score >= 70.0)
    System.out.println("C");
else if (score >= 80.0)
    System.out.println("B");
else if (score >= 90.0)
    System.out.println("A");
else
    System.out.println("F");
```

اجابات التدريبات العامة:

1. (أ) التنفيذ المتسلسل (ب) نقل مسار التحكم (ت)  $\&&$ ،  $\|$  (ث) خطأ برمجي، خطأ برمجي (ج) تحديد النجاح من الرسوب، تحديد حاجة الخزان الآوتوماتيكي للتعبئة من عدمه. (ث) الأقواس { } (ث)

2. حدد صحة أو خطأ كلاً من العبارات التالية مع التعليل:

أ- (خاطئه) لفحص المساواة بين رقمين صحيحين نستخدم العلاقة  $(==)$

- بـ - (صحيحة) عند بناء شرط مركب بين شرطين خاطئين بالعلاقة <sup>٨</sup> يكون الناتج العام للشرط الخطأ.
- تـ - (صحيحة) عند اهتمامنا بخيارين متناقضين في وجود شرط واحد يمكن استخدام الجملة if.. else
- ثـ - (صحيحة) وضع فاصلة منقوطة بعد أقواس الشرط في جمل اتخاذ القرار يؤدي لخطأ منطقى
- جـ - (خاطئة) تستخدم الدالة compareTo للمقارنة بين النصوص
- حـ - (خاطئة) عدم احتواء المتغير في الشرط على قيمة يجعل المترجم يظهر خطأ برمجي يمنع التنفيذ.

### 3. عَبَرْ عنِ الْجَمْلِ التَّالِيَّةِ بِلُغَةِ جَافَا:

```
أـ إن كان معدله أكبر من أو يساوي 50 ستطيع له "ناجح"
if (GPA >= 50)
    JOptionPane.showMessageDialog (null, "Scuccess");

بـ إن كان المواطن عمره 60 فما فوق ولديه طفلين على الأقل نطبع "سيصرف لك 200 دينار".
if ((Age >= 60) && (Child >=2))
    JOptionPane.showMessageDialog (null, "سيصرف لك 200 دينار");

تـ تحويل النص choice إلى نوع صحيح يمكن تخزينه في متغير Num سعته 1 بait فقط.
byte num = Byte.parseByte(choice);
```

### 4

أـ الخطأ: وضع فاصلة منقوطة بعد الشرط  
نوع الخطأ: خطأ منطقى

```
if (c < 7)
    JOptionPane.showMessageDialog (null, "c is less than 7");

بـ الخطأ: عدم وضع الشرط المركب بين قوسين  
نوع الخطأ: خطأ برمجي
```

if ((c < 7)|| (x>8))
 JOptionPane.showMessageDialog (null, "Open");

تـ الخطأ:

- استخدام النوع long للمتغير الشرطي في switch وهو غير مسموح
  - الحرف الأول في الكلمة break حالة كبيرة والأصل أنها صغيرة
- نوع الخطأ: في الحالتين خطأ برمجي

```

int num = 23;
switch(num){
    case 1:
        JOptionPane.showMessageDialog(null,"Satarday");
        break;
    case 2:
        JOptionPane.showMessageDialog(null,"Sunday");
        break;
    default:
        JOptionPane.showMessageDialog(null," should be 1 To 7");
        break;
}

import javax.swing.JOptionPane;
public class squarea {
    static String input;
    static int L, area;

    public static void main(String[] args) {

        input = JOptionPane.showInputDialog("input The Length: ");
        L = Integer.parseInt(input);
        area = L * L;
        if (area > 20)
            JOptionPane.showMessageDialog(null, "The area is: "+ area);

        JOptionPane.showMessageDialog(null, "Jazak Allah Khayran ");

    }
}

```

شكل 4.18: حل تمرين 5

```

import javax.swing.JOptionPane;
public class Practice4_6 {
    public static void main(String[] args) {
        String input = JOptionPane.showInputDialog("enter Temp1");
        float temp1 = Float.parseFloat(input);
        input = JOptionPane.showInputDialog("enter Temp2");
        float temp2 = Float.parseFloat(input);
        input = JOptionPane.showInputDialog("enter Temp3");
        float temp3 = Float.parseFloat(input);
        float avgTemp = (temp1+temp2+temp3)/3;
        if(avgTemp>=24)
            JOptionPane.showMessageDialog(null,"حار");
        else if (avgTemp>=18)
            JOptionPane.showMessageDialog(null,"معتدل");
        else
            JOptionPane.showMessageDialog(null,"بارد");
    }
}

```

شكل 4.19: حل تمرين 6

```

import javax.swing.JOptionPane;
public class Practice4_7 {
    public static void main(String[] args) {
        String input = JOptionPane.showInputDialog("enter num");
        int num = Integer.parseInt(input);
        if(num%3==0)
            JOptionPane.showMessageDialog(null,num+"is dividable by 3");
        else
            JOptionPane.showMessageDialog(null,"Try again");
    }
}

```

شكل 4.20: حل تمرين 7

```

import javax.swing.JOptionPane;
public class Practice4_8 {
public static void main(String[] args) {
String input = JOptionPane.showInputDialog("enter 1'st");
int num1 = Integer.parseInt(input);
input = JOptionPane.showInputDialog("enter 2'nd");
int num2 = Integer.parseInt(input);
input = JOptionPane.showInputDialog("enter 3'rd");
int num3 = Integer.parseInt(input);
سيتم الآن مقارنة المتغيرات الثلاث لمعرفة الأكبر //
int max = num1;
if (num2 > max)
{
    if (num2 > num3)
        max = num2;
    else
        max = num3;
} else if (num3 > max)
    max = num3;
JOptionPane.showMessageDialog(null, "The max is "+max);
}
}

```

شكل 4.21: حل تمرين 8

في التمرين نحدد أحد المتغيرات على أنه هو العنصر الأكبر، ثم نبدأ في مقارنته بالعناصر الأخرى، فإن تم افتراض فمثلاً أن الرقم الأول هو الأكبر، نقوم بعد ذلك بمقارنته وبالتالي، فإن لم يكن أكبر من الثاني، نسأل هل أكبر من الثالث، فإن كان نعم فيكون الرقم الأول هو الأكبر وإن كان لا فيكون الرقم الثالث هو الأكبر وهكذا، المهم أن نعلم أن الثلاثة أرقام لا بد أن يكون بينهم مقارنة صريحة أو ضمنية لتحديد أيهم الأكبر كما تتبع في إلى حل.

```

import javax.swing.JOptionPane;
public class password_Prog {
    public static void main(String[] args) {
        String name, password, tname, tpassword;
        name = JOptionPane.showInputDialog("Enter your name");
        password = JOptionPane.showInputDialog("Enter your Pass word");
        tname = JOptionPane.showInputDialog("Enter your name");
        tpassword = JOptionPane.showInputDialog("Enter your Pass word");
        if ((name.equals(tname)) && (password.equals(tpassword)))
        {
            JOptionPane.showMessageDialog(null, "Welcome!!!!");
        } else if (name.equals(tname))
            JOptionPane.showMessageDialog(null, "retype your pass word");
        else if (password.equals(tpassword))
            JOptionPane.showMessageDialog(null, "retype your user name");
        else
            JOptionPane.showMessageDialog(null, "your data is incorrect");
    }
}

```

شكل 4.22: حل تمرین 9

في التمرین 9 السابق، يمكن الاقتصار فقط على فحص إن كانت إحدى المعلومات خاطئة فيتم إخراج رسالة بالخطأ، بينما الحل الموجود أكثر عمقاً حيث يظهر للمستخدم أين الخطأ بالضبط إن حصل. لاحظ أن التمرین تم حله من خلال الشرط المركب بواسطة العلاقة `&&` لأنه لا يمكن أن يفتح له الحساب دون أن يكون كلا المعلوماتين صحيحتين. التطبيق نفسه يمكنك محاولة حله باستخدام جمل الاختيار المتداخلة بالشروط المتفقة.

```

import javax.swing.JOptionPane;
public class Practice4_10 {
    public static void main(String[] args) {
        String input = JOptionPane.showInputDialog("enter num");
        int num = Integer.parseInt(input);
        if (num%2==0)
            JOptionPane.showMessageDialog(null, "Even");
        else
            JOptionPane.showMessageDialog(null, "Odd");
    }
}

```

شكل 4.23: حل تمرین 10

انتهى الباب

كثير من العمليات يتكرر تنفيذها من قبل الإنسان في وقت أصبح بإمكانه ترك تلك المهام للحاسوب لينجزها أسرع وبشكل أدق.

## الباب الخامس

### جمل التكرار

#### (Repetition Statements)

يتوقع من الدارس في نهاية هذا الباب أن:

- يدرك الحاجة إلى استخدام جمل التكرار في البرمجة.
- يفهم كيفية تكرار الجملة البرمجية لعدد من المرات.
- يُتقن كيفية التحكم في سير العمليات باستخدام جمل التكرار.
- يُعدّ الجمل البرمجية الخاصة بتكرار الأوامر.
- يُتقن استخدام جمل التكرار *for*, *do.. while*, *while*.
- يُفرق بين استخدامات جمل التكرار.
- يُميز عملياً بين استخدام الأمر *continue* و*break*.
- يتقن كتابة برامج أكثر تفاعلاً مع المستخدم باستخدام جمل التكرار المتداخلة.
- يُميز بين أنواع الأخطاء التي قد يواجهها المبرمج خلال تطوير تطبيقاته.

يتوفر لهذا الباب شرائح العرض (PowerPoint) وكذلك ملفات مرنية من إعداد مؤلف الكتاب؛ يمكنك الحصول عليها من خلال الموقع الأكاديمي للمؤلف [mfarra.cst.ps](http://mfarra.cst.ps)



## الباب الخامس: جمل التكرار

### 5.1 مقدمة

في الباب السابق، استعرضنا مفهوم التحكم في سير العمليات، حيث ناقشنا بعمق جمل الاختيار كونها من أهم وأشهر جمل التحكم، نظراً إلى حاجتنا لها في معظم - إن لم يكن - جميع تطبيقاتنا الحياتية، بينما في هذا الباب سنناقش نوعاً آخر، ألا وهو جمل التكرار (*Repetition Statements*) أو الدوران، فإن كانت جمل اتخاذ القرار تعمل على تجاهل تنفيذ أوامر وتتنفيذ أخرى؛ فإن جمل التكرار تعمل على تنفيذ مجموعة من الأوامر البرمجية بعدد من المرات يزيد عن عدد ظهورها في البرنامج حسب شروط معينة دون تجاهل الغير.

في الفصول القادمة سنوضح منطق التكرار وتطبيقاته العملية ثم الجمل البرمجية التي تسمح لنا بتطبيق هذا المفهوم وفي النهاية سنجمل الأخطاء التي قد تظهر علينا أثناء كتابتنا للبرامج مع تفصيل أنواعها وتوضيحها عند التعامل مع جمل التحكم في سير العمليات.

### 5.2 فهم منطق التكرار والتطبيقات العملية له

يقصد بالتكرار في البرمجة، تنفيذ جملة أو مجموعة من الجمل البرمجية ككتلة واحدة لعدد من المرات أو بناءً على شرط منطقي معين، وذلك بناءً على حاجة التطبيق أو المشكلة البرمجية التي نحن بصدد البحث عن حل برمجي لها، على أن يساهم الحل في تحقيق كافة مميزات البرمجة أو الحوسبة التي تم مناقشتها سابقاً في الباب الأول والثالث.

إذا نظرنا حولنا، سنجد أنَّ كثيراً من العمليات سواء الاقتصادية أو التنظيمية عند استخدام التكرار البرمجي فيها سيساهم في زيادة الانتاج والدقة لها، مثل ذلك إجراء العمليات الحسابية المعقدة ذات التركيبة الواضحة مثل حساب مضروب العدد (*factorial*) أو طباعة جدول الضرب لمجموعة أرقام، كذلك البحث عن الأرقام الزوجية أو الفردية أو مضاعفات رقم ما في نطاق كبير من الأرقام المتسلسلة، وهذه من التطبيقات البسيطة حولنا التي تحتاج فيها إلى التكرار زيادة سرعتها ودقتها ونتاجها، كما يمكن أن تحتاجها في تطبيق مشاريع مثل تعبئة خزانات المياه والبترول وغيرها بشكل تلقائي.

لعلنا ناقشنا منطق التكرار في الباب الثاني عندما تحدثنا عن خرائط التكرار بأنواعها، وقد تسمى دوراناً لأن تصميمها بالخوارزميات يوحى بوجود دوران كما تلاحظ في الشكل 2.11 من الباب الثاني.

كما أظهرنا أن التكرار يحكمه ثلاثة عوامل مهمة هي:

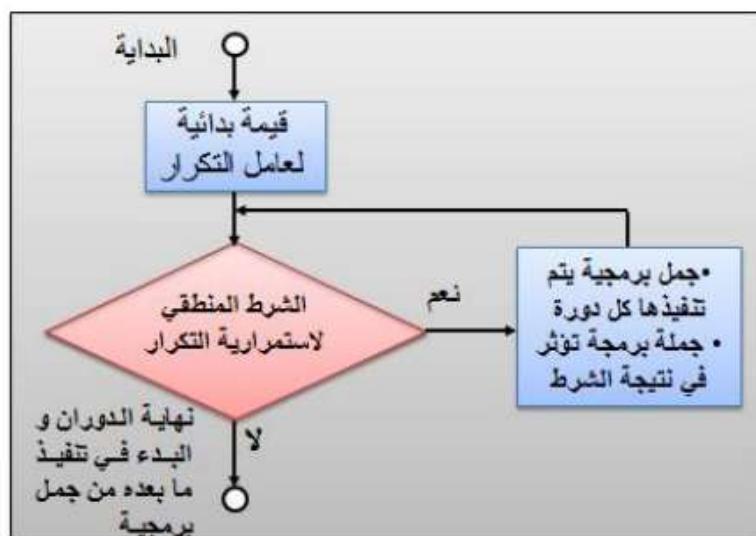
1. شرط الاستمرارية.
2. عامل يؤثر على شرط الاستمرارية.
3. قيمة بداعية لهذا العامل يبدأ من عندها.

عليه فإن تطبيق التكرار بلغة جافا يحتاج إلى جمل تعالج هذه النقاط بطريقتها الخاصة، وهذا ما سنوضحه مع كل نوع من جمل التكرار الثلاث القادمة.

### 5.3 استخدام جملة التكرار while

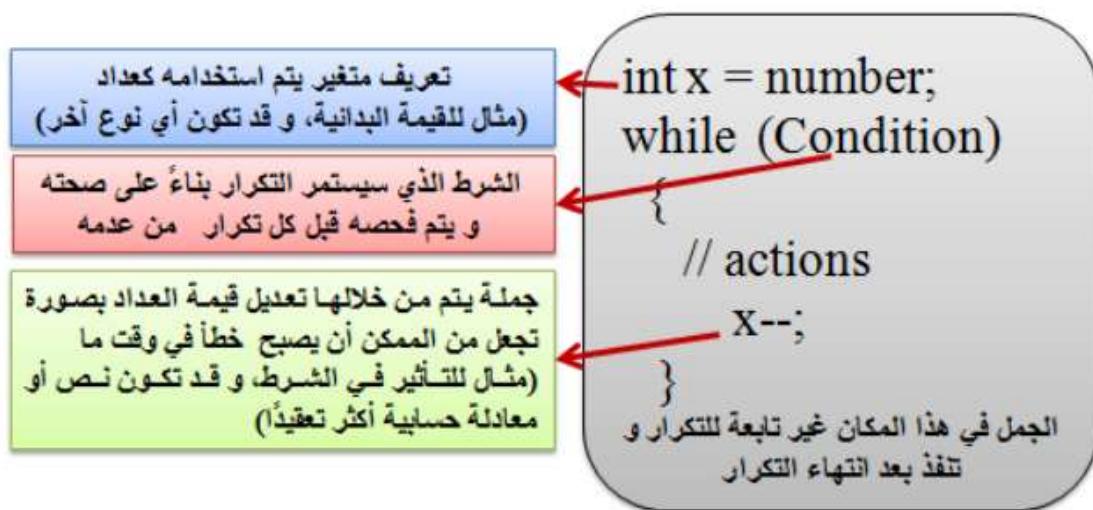
تعتبر جملة (while) من جمل التكرار التي تعمل على فحص شرط الاستمرارية قبل البدء في تنفيذ أي تعليمية برمجية تابعة للتكرار، ويوضح الشكل 2.11 (من الباب الثاني) المنطق العام لها كما يوضح الشكل 5.1 طريقة عملها.

وكما تلاحظ في الشكل 5.1، فإن القيمة البدائية للعامل الذي يتأثر به شرط التكرار لا بد أن يكون قبل بدء التكرار، ولا يتم تنفيذه إلا مرة واحدة حتى لا يحدث خطأ منطقى يؤدي إلى إعادة الحاله لبدايتها في كل دورة (تصفير العداد)، ثم بعد ذلك فإن العملية تتوجه للدخول على شرط التكرار الذي يتأثر بالعامل السابق، فإن كانت نتيجة الشرط (نعم) تم الانتقال لتنفيذ الجمل التابعه للتكرار ومنها جملة تؤثر على شرط التكرار، وإن كانت (لا) يخرج سير العملية من التكرار وتنقل لتنفيذ الجمل التالية للتكرار.



شكل 5.1: منطق عمل التكرار باستخدام while

هذا المنطق من التكرار يمكن تحويله إلى البرمجة كما بالشكل 5.2 والذي يُظهر استخدام كلمة (while) وهي إحدى الكلمات المhogزة التي تُستخدم لتطبيق هذا النمط من التكرار.



شكل 5.2: مبدأ عمل جملة التكرار while في لغة جافا

تعرف جملة (while) باسم "جملة التكرار مسبقة الاختبار" (*Pre-test loop*), لأن الشرط يتم فحصه قبل تنفيذ أي جملة تابعة للتكرار، فإن كانت نتيجة الشرط (نعم) يتم تنفيذ مجموعة الجمل، وإن كانت (لا) يتم تجاهل كافة الجمل الموجودة ضمن نطاق التكرار والانتقال لما بعد التكرار.

في المثال التالي سنستخدم جملة (while) في طباعة الأرقام من 1 إلى 100 كمثال توضيحي بسيط على استخدام هذه الجملة، وعند التفكير في هذه المشكلة البرمجية، نجد أننا بحاجة إلى البحث عن عامل يتأثر به التكرار، وطالما كان الأمر متعلق بأرقام متسلسلة، فإن هذا العامل يكون عداد (*Counter*) يبدأ من حيث نريد الطباعة وينتهي حيث نريد انتهاء الطباعة، وطالما نحن نريد الطباعة من 1 إلى 100، فإن بداية العدد تكون 1 ونهايته عند 100. ثم نبحث عن شرط استمرارية التكرار وهو الشرط الذي يضمن لي أن أكرر العملية 100 مرة وبالتالي يكون (العداد=> 100) حيث سنكرر العملية 100 مرة لأننا بدأنا من 1 وسنصل للعدد 100.

### مثال توضيحي 5.1:

اكتب برنامج يطبع الأرقام الصحيحة من 1 إلى 100 مستخدماً التكرار.

```
1 public class Print1_100 {
2     public static void main(String[ ] args) {
```

```

3 int x = 1;
4 while (x<=100)
5 {
6     System.out.println("x= "+x);
7     x++;
8 } // جمل التكرار
9 } // نهاية الدالة الأساسية
10 } // نهاية البرنامج

```

في هذا المثال، قمنا بتعريف المتغير (x) من النوع الصحيح وأعطيته قيمة بدأية (يبدأ بها) مقدارها 1 لأن البرنامج يريد منا طباعة الأرقام من 1 إلى 100 وبالتالي جعلنا الشرط ( $x \leq 100$ )، وانتبه معنـى إلى أنـنا استخدـامـنا أصـغرـ من أو يـساـويـ وـلـمـ نـسـتـخـدمـ أـصـغـرـ منـ فـقـطـ (ـلـمـاـذاـ؟ـ) لـأـنـاـ نـحـاجـ طـبـاعـةـ الرـقـمـ 100ـ مـعـ الـأـرـقـامـ،ـ بـيـنـماـ إـذـاـ كـانـ المـطـلـوبـ فـقـطـ لـرـقـمـ 99ـ،ـ فـكـانـ يـكـفـيـ وـقـتـهاـ أـقـلـ (100 > x)ـ أـوـ (99 <= x).

الآن، جملة التكرار سيتم من خلالها تنفيذ أكثر من جملة ولهذا تم استخدام الأقواس {}, لنتتبع سوياً الجملة: في السطر رقم 3 تم تعريف متغير يعمل كعداد اسمه (x) وأعطيته قيمة 1، ثم انتقلنا إلى جملة التكرار بداية بشرط الاستمرارية، هل قيمة العدد أصغر من أو يساوي 100؟ الإجابة نعم لأن ( $x \leq 100$ ) فيتم الانتقال للجمل ضمن التكرار في سطر 6 و 7 لا بد أن يتم تنفيذها سوياً أو تجاهلها سوياً.

في السطر 6 نطبع الرقم الذي وصلنا له بداية من 1 ثم ننقل للسطر رقم 7 الذي يقوم بعمل زيادة مقدارها 1 للمتغير x ليصبح 2 ثم نعود للشرط في السطر 4 فنجد الشرط معناه هل (قيمة المتغير x أصغر من أو يساوي 100)؟ فتكون الإجابة نعم، فنكمـلـ تنـفيـذـ الخطـواتـ دـاخـلـ التـكـرـارـ فـنـطـبـعـ الرـقـمـ 2ـ ثـمـ نـزـيدـ قـيمـةـ xـ وـهـكـذـاـ حـتـىـ يـصـلـ الرـقـمـ فيـ العـدـادـ إـلـىـ 100ـ وـبـالـتـأـكـيدـ (100 >= 100)،ـ فـنـطـبـعـ الـقـيمـةـ 100ـ وـنـزـيدـ عـلـىـ العـدـادـ الـقـيمـةـ 1ـ لـيـصـبـحـ 101ـ وـنـعـودـ إـلـىـ الـشـرـطـ فـهـلـ (101 >= 101)،ـ الإـجـابـةـ بـالـتـأـكـيدـ لـاـ فـيـتـوقـفـ التـكـرـارـ وـيـخـرـجـ التـحـكـمـ لـخـارـجـ التـكـرـارـ فـإـنـ كـانـ هـنـاكـ جـمـلـ لـلـتـنـفـيـذـ يـتـمـ تـنـفـيـذـهـ إـنـ لمـ يـكـنـ (ـكـمـ هـوـ الـحـالـ إـلـاـنـ)،ـ يـنـتـهـيـ الـبـرـنـامـجـ.

**خطأ منطقـيـ:** عند عدم كتابة أي جملة برمجية تجعل شـرـطـ استـمـرـارـيـةـ التـكـرـارـ يـصـبـحـ خـاطـئـ فيـ وقتـ ماـ،ـ فـهـذـاـ يـجـعـلـ الـبـرـنـامـجـ يـسـتـمـرـ فيـ التـنـفـيـذـ دونـ نـهـاـيـةـ (*infinite loop*).



وضع فاصلة منقوطة بعد أقواس التكرار { } لا يعطي خطأ منطقياً فالمنترجم يعتبر أنها جملة برمجية فارغة، ولا ينفذ الأوامر سوى مرة واحدة، ورغم ذلك عليك الانتباه إلى أنَّ جملة `while` لا تشتمل على فاصلة منقوطة في نهايتها.



**خطأ منطقى:** عند كتابة شرط منطقى وقيمة بدانية لا ينتهي على الإطلاق فهذا يجعل التكرار يستمر دون توقف وهو ما يعرف بـ *infinite loop* وهو خطأ منطقى لا يظهر أثناء التنفيذ. **مثال:** القيمة البانية `x = 100`، الشرط (`x < 1`)، التأثير في القيمة من خلال (`x++`) بهذه الصورة ستظل قيمة `x` تكبر وسيبقى الشرط صحيحاً ولن يتوقف التكرار.



### مثال توضيحي 5.2 :

اكتب برنامج يطبع الأرقام الصحيحة الزوجية من 1 إلى 100 مستخدماً التكرار.

```

1 public class even_100 {
2     public static void main(String[ ] args) {
3         int x = 2;
4         while (x<=100)
5         {
6             System.out.println("x= "+x);
7             x = x+2;
8         } // نهاية التكرار
9     } // نهاية الدالة الرئيسية main
10 } // نهاية الصنف الرئيسي
    
```

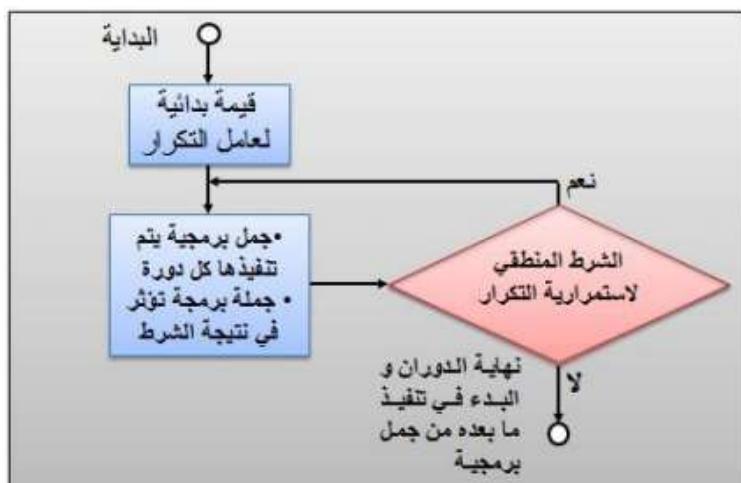
في هذا المثال ليس أمامنا الكثير لنشرحه، فكُّر معي قليلاً ستجد أنه طلب مثنا طباعة الأرقام الزوجية من 1 إلى 100 بدلاً من طباعة كافة الأرقام، وكما تعلم فالأرقام الزوجية هي 0، 2، 4، ... وهكذا نستمر بزيادة القيمة 2 على ما سبق مع البداية من رقم زوجي، هذا يأخذنا إلى أنَّ البداية لا بد أن تكون زوجية وهي 2 لأنَّ طلب

طباعة الأرقام الزوجية من 1 إلى 100، والرقم 1 كما تعلم ليس زوجياً؛ لذلك نختار أول زوجي بعده وهو الرقم 2، ثم تكون الزيادة بمقدار 2 كما في السطر رقم 7، ولاحتاج التغيير على الشرط الوارد في المثال 5.1.2، الآن يمكنك بفهم هذا المنطق أن تكتب العديد من البرامج مثل طباعة مضاعفات العدد 3، والعدد 5 وغيرهم، كذلك يمكنك طباعة الأرقام الفردية [ما هي قيمة البداية؟ وما هي قيمة الزيادة؟].

#### 5.4 استخدام جملة التكرار (do ... while)

جملة (do ... while) تعمل بالمنطق ذاته الذي تعلم به جملة (while) في سير عملية التكرار، باستثناء أنها تقوم بفحص شرط الاستمرارية بعد أن يتم تنفيذ الجمل التي تحتاج إلى تكرارها وليس قبل ولهذا تعرف بأنها جملة تكرار بعديّة الاختبار (Post-test loop) فالاختبار يأتي بعد التنفيذ لا قبله.

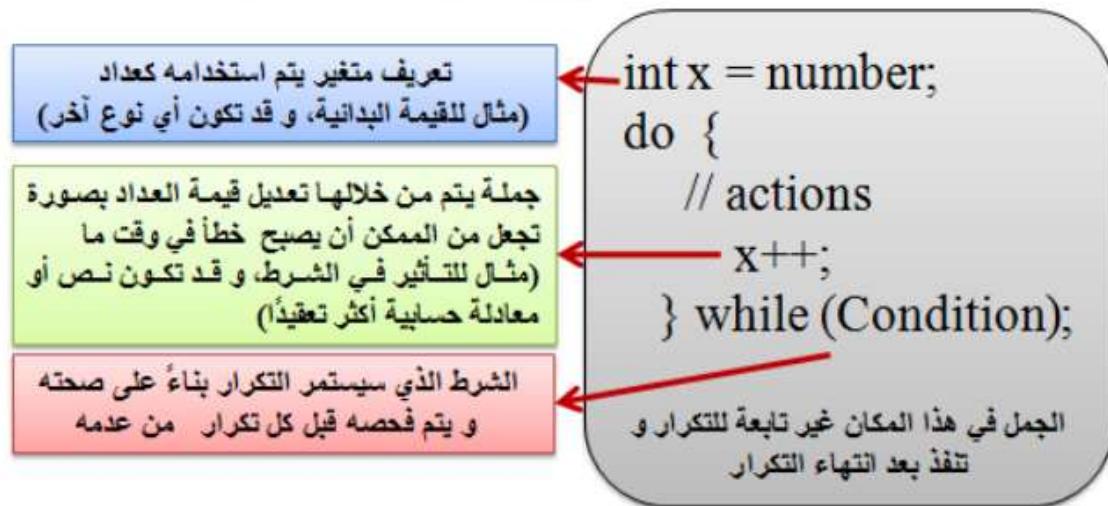
الشكل 5.3 يوضح منطق عملها، كما يوضح الشكل 5.4 طريقة عملها بلغة Java، والذي لا يختلف عن جملة (while) في عناصره فقط اختلاف ترتيب ظهور العناصر.



شكل 5.3: منطق عمل جملة التكرار do ... while

و قبل أن ننتقل للمثال التوضيحي، انتبه جيداً في الشكل 5.4، لتجد أن من مبادئ تطبيق جملة التكرار (do...while) هو وضع فاصلة منقوطة بعد شرط الاستمرارية علماً بأن تجاهلها يؤدي إلى خطأ برمجي وهو ما لم يكن يحدث في جملة (while).

والآن قبل أن نبدأ في تطبيقها برمجياً، دعنا نفكّر قليلاً، فلا أريدك أن تنسى أن البرمجة بحاجة إلى التفكير والتحليل والتخطيط، وإلا فلن تتمكن من كتابة برامج صحيحة غالباً، ولعل نوعية المشاكل البرمجية التي تتعلق بالتكرار واتخاذ القرار سيظهر فيها حاجتنا إلى هذا النمط بشكل واضح.



شكل 5.4: مبدأ عمل جملة التكرار do ... while

**مثال توضيحي 5.3:**

اكتب برنامج يستقبل من المستخدم مجموعة من الأرقام الصحيحة الموجبة، ويطبع مجموعهم، على أن يستمر في استقبال الأرقام ما لم يدخل الرقم صفر.



شكل 5.5: تذکیر بملخص خطوات حل المشكلة برمجياً

في هذا المثال لم يحدد لي بداية أو نهاية واضحة كما كان الحال في مثال 5.2 و 5.1، وبالتالي علينا علينا التفكير لنحدد ما هي البداية وما هي النهاية. في هذا المثال أنا لست مُحدداً بعده من الدورات، وبالتالي فالنهاية غير متعلقة بالوصول لنهاية رقمية محددة (هل أحتاج عدداً؟)، بينما النهاية مرتبطة بقيمة سيدخلها المستخدم وعندما سيتوقف التكرار وهي قيمة الصفر، فإن أدخل صفر يتوقف عن التكرار وإلا يستمر التكرار.

وبالتالي المتغير الذي يؤثر على شرط الاستمرار لن أعطيه قيمة بدایية من عندي وإنما سنترك ذلك للمستخدم ليحدد القيم بما فيها القيمة الأولى التي سيدخلها فإن أدخل القيمة صفر توقف التكرار مباشرة.

الأمر الآخر، أن البرنامج لا بد أن يجمع فقط الأرقام الموجبة ولهذا بعد أن نكتب جملة الإدخال والتحويل، لا بد من وضع شرط بجملة (*if*) فإن كان الرقم المدخل أكبر من صفر يتم اعتماده في المجموع وإلا يتم تجاهله والانتقال لدورة جديدة وهكذا.

بالطبع نحتاج إلى طباعة المجموع في نهاية الأمر، ولهذا نستخدم جملة الطباعة خارج التكرار حتى لا يكرر الطباعة بعد كل قيمة يدخلها المستخدم، بل يطبع فقط في نهاية البرنامج [جزء إدخال جملة الطباعة داخل التكرار].

**تَذَكَّرُ أَنَّ:** شرط عدم المساواة يتم من خلال العلاقة ( $!=$ ).



```

1 import javax.swing.JOptionPane;
2 public class SumUsingDoWhile{
3     public static void main(String[ ] args){
4         int sum =0;
5         int x;
6         do{
7             String input = JOptionPane.showInputDialog("enter num");
8             x = Integer.parseInt(input);
9             if (x>0)
10                 sum = sum + x;
11         } while (x!=0);
12     JOptionPane.showMessageDialog(null,"Sum: "+sum); } }
```

#### مثال توضيحي 5.4 :

اكتب برنامج يستخدم التكرار ، ويطبع مضاعفات العدد 7 الزوجية من 7 وحتى 100 .

هذا المثال، يحتاج مثلاً إلى طباعة مضاعفات العدد 7 ، أي الأرقام التي تقبل القسمة على الرقم 7 دون باقي ، أو هي حاصل ضرب الرقم 7 في الأرقام الأخرى أن تكون زوجية أيضاً.

```

1 import javax.swing.JOptionPane;
2 public class Multi7_2 {
3     public static void main(String[] args){
4         int x=7;
5         String multi = "The Multi of 7 & 2 are \n";
6         do {
7             if ((x%7==0)&&(x%2==0))
8                 multi = multi + x +"\n";
9             x++;
10        }while(x<=100);
11    JOptionPane.showMessageDialog(null,multi);
12 }
13 }
```

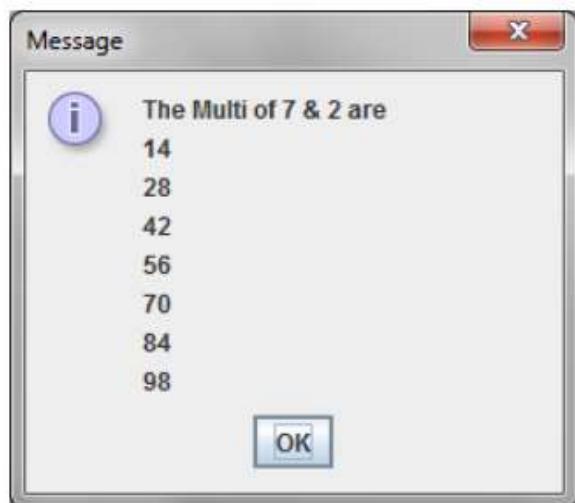
في هذا المثال حدد لك المجال الذي نريد أن نعمل فيه وهو من 7 وحتى 100، وبالتالي نحن نحتاج فقط إلى فحص الأرقام في هذا المجال هل تقبل القسمة على 7 أم لا، هذا الفحص يتم من خلال قسمة العدد على 7 من خلال عملية باقي القسمة % فإن كان الباقي صفر فهذا يعتبر من مضاعفات الرقم 7 وإن كان الباقي غير ذلك فهو لا يعتبر من مضاعفات.

الأمر الآخر أنه لا يريد طباعة كل المضاعفات بل يحتاج فقط إلى طباعة الأرقام الزوجية منها، وبالتالي فنحن بحاجة إلى عمل شرط مركب بحيث يكون باقي قسمة الرقم على 7 يساوي صفرًا وكذلك باقي قسمته على 2 يعتبر صفرًا، ومثال ذلك الرقم 14 فهو يقبل القسمة على الرقم 7 و 2.

إذا تابعت حل المثال فستجد أننا استخدمنا التكرار هنا من أجل المرور على الأرقام الواحد تلو الآخر وفحصها من خلال الشرط المركب  $((x \% 7 == 0) \&\& (x \% 2 == 0))$  فإن كان نتيجته "نعم" يتم ضمه إلى قائمة الأرقام التي تقبل القسمة على 7 و 2.

ولكي نطبع كافة الأرقام في صندوق حوار واحد كما بالشكل 5.6، نقوم بتعريف متغير من نوع (*String*) كمكان لتجمّع القيم فيه ووضعنا فيه العنوان أو النص الأساسي ولتكن في هذه الحالة (The Multi of 7 & 2 are) ثم نضع إحدى العلامات الخاصة للفصل بين القيم مثل السطر الجديد `\n` أو المسافات الثمانية `\t`. بعد ذلك كلما حصلنا على قيمة جديدة وأردنا ضمها إلى القائمة، نقوم بإلصاق القيمة الجديدة على شكل نص بالنص السابق من خلال العملية `(+)` وذلك يتضح في السطر 8 مع إضافة علامة السطر الجديد `\n` ليتم وضع كل قيمة جديدة في سطر جديد.

معادلة تجمّع الناتج دائمًا ما تكون بهذه الفكرة مع تعديلات بسيطة تتناسب مع السؤال، ولكنها دائمًا تكون على نمط (النص = النص القديم + النص الحالي) وفي هذه الحالة نجعل النص هو النص القديم ليتم التعديل عليه أولاً بأول، كما تشاهد في الشكل 5.6 الذي بدأ بجملة واحدة ثم يزداد مع كل رقم من المضاعفات.



شكل 5.6: ناتج مثال 5.4

**خطأ برمجي :** عدم وضع الفاصلة المنقوطة بعد شرط الاستمرارية لجمل `do...while` يسبب خطأ برمجي.



وبما لاحظت للأمثلة والشرح حول جملتي (`do...while`) و(`while`)، تجد أن الفارق الفعلي في الناتج يكمن في أن الأولى لا يمكن أن تتفذ أي عمليات دون أن يكون شرط الاستمرارية صحيحًا بينما الثانية تتفذ على الأقل جمل التكرار مرة واحدة، الشكل 5.7 يوضح الفارق بين النوعين.

عدم وضع الأقواس {} في جملة (`while`) يسبب خطأ منطقى حيث سيتم تنفيذ الجملة التالية لـ (`while`) دون توقف بينما عدم وضعها لجملة (`do...while`) يسبب خطأ برمجي.





شكل 5.7: مقارنة بين جملة while يميناً و do...while يساراً

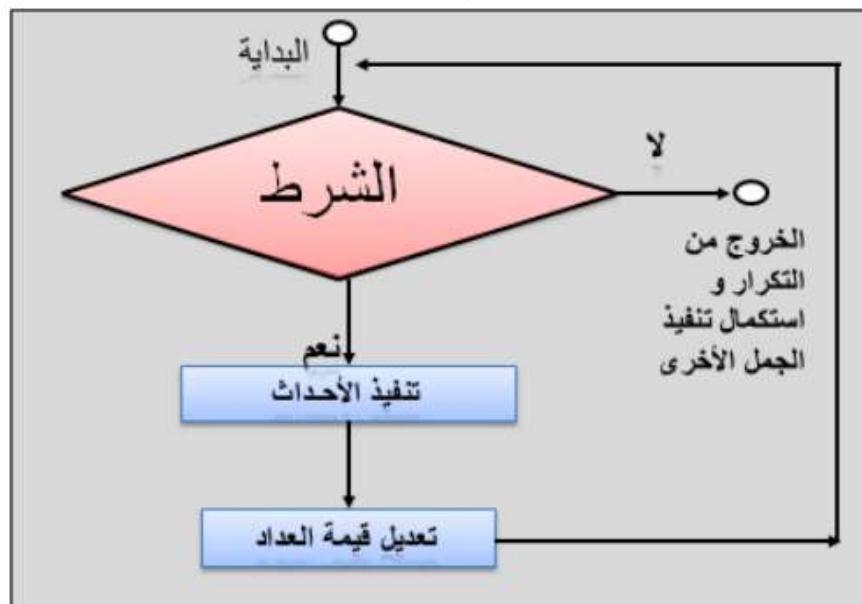
### 5.5 استخدام جملة التكرار for

تتوافق جملة التكرار (`for`) مع جملة (`while`) في كونها لا تتفذ العمليات قبل التأكد من صحة شرط الاستمرارية، كما يتضح ذلك في الشكل 5.8، بينما في الشكل 5.9 يتضح معنا منطق عملها.



شكل 5.8: مبدأ كتابة جملة for بلغة Java

ولعل جملة (`for`) مرتبطة أكثر بوجود عدد مرات تكرار العمليات المراده، وإن كانت يمكن لها أن تتحول لتطابق كافة التطبيقات.



شكل 5.9: منطق عمل جملة التكرار for

لاحظ معي في الشكل 5.8 و 5.9 أن تسلسل التكرار في جملة (*for*) يبدأ من تحديد قيمة البداية وهي تحدد مرة واحدة ولا يعود المترجم لتنفيذها مرة أخرى، بعد ذلك وفي كل دورة من دورات التكرار يتم فحص الشرط فإن كان صحيحاً يتم تنفيذ الجمل الموجودة كجواب للشرط، ثم يتم تنفيذ الجملة الخاصة بتعديل قيمة العدد ثم العودة إلى فحص الشرط وهكذا.

### مثال توضيحي 5.5:

اكتب برنامج يستخدم التكرار لطباعة حاصل مضروب الأعداد من 1 إلى 15.  
مثال: الحاصل =  $1 \times 2 \times 3 \times 4 \times \dots \times 15$

```

1 import javax.swing.JOptionPane;
2 public class multiFor100 {
3     public static void main(String[ ] args){
4         int mull=1;
5         for(int i=1; i<=15;i++)
6             mull = mull * i;
  
```

```

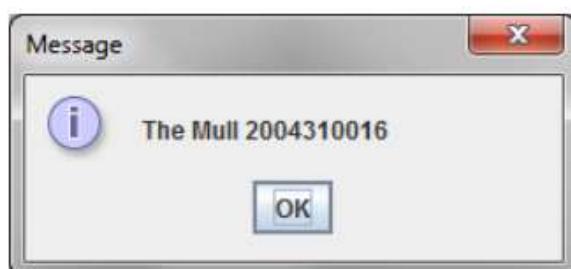
7 JOptionPane.showMessageDialog(null,"The Mull "+mull);
8 }
9 }
```

عملية حاصل ضرب سلسلة من الأرقام، تحتوي على تكرار واضح للرقم فيمن قبله، ولهذا فقد استخدمنا جملة التكرار (**for**) وجعلنا متغير التكرار (**i**) يبدأ من 1 حتى 15 وفي كل مرة نضربه في حاصل ضرب ما سبق وهكذا.

انتبه لتعريف جملة (**for**) في السطر رقم 5، في الجزء الأول تم تعريف المتغير (**i**) من نوع (**int**) وإعطاؤه قيمة بدائية مقدارها 1 لأننا نريد تنفيذ عمليات تبدأ علاقتها من الرقم 1، وهذا المتغير يمكن تعريفه داخل جملة **for** ويمكن تعريفه قبلها. وهذا الجزء يقوم بتنفيذ المترجم مرة واحدة فقط عند بداية تنفيذ جملة (**for**).

في الجزء الثاني تم بناء شرط استمرارية التكرار وهو (**i=15**) وقد تم تحديده بهذا الشكل، لأننا نريد حساب حاصل ضرب الأرقام من 1 وحتى 15، وبالتالي نريد أن يستمر التكرار طالما كان الرقم أقل من أو يساوي 15. وهذا الجزء يقوم المترجم بتنفيذه (فحصه) في كل دورة من دورات التكرار، فإن كان ناتج الشرط (نعم) ينتقل التنفيذ إلى العمليات التي تحتاج تكرارها، وإن كان ناتج الشرط (لا) يتوقف التكرار وينتقل التنفيذ إلى ما بعد جملة التكرار.

في الجزء الثالث يتم وضع المعادلة التي يتم من خلالها التعديل على قيمة متغير التكرار، وهذا الجزء يتم بتنفيذه بعد كل دورة من الدورات، ومن المفترض أن يكون تأثير هذه المعادلة لإيقاف التكرار من خلال جعل ناتج الشرط خاطئ في مرحلة من المراحل.



شكل 5.10: ناتج مثال 5.5

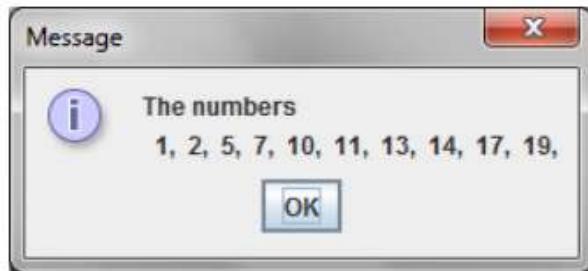
في هذا المثال، يعتمد استخراج النتائج وطباعتها على بناء شرط مركب لطباعة كافة الأرقام التي لا تقبل القسمة على 3 و 4، وهذا يعني أننا نريد الأرقام التي باقي قسمتها على 3 و 4 لا يساوي صفر وهذا ما يتضح في السطر 7.

### مثال توضيحي 5.6:

اكتب برنامج يستخدم التكرار لطباعة كافة الأرقام التي لا تقبل القسمة على 4 و 3 من 1 إلى 20.

```

1 import javax.swing.JOptionPane;
2 public class Xor3_4 {
3     public static void main(String[ ] args){
4         String output ="The numbers\n";
5         for (int i =1;i<=20;i++)
6         {
7             if ((i%3!=0)&&(i%4!=0))
8                 output = output+i+"\n";
9         }
10    JOptionPane.showMessageDialog(null,output);
11 }
12 }
```



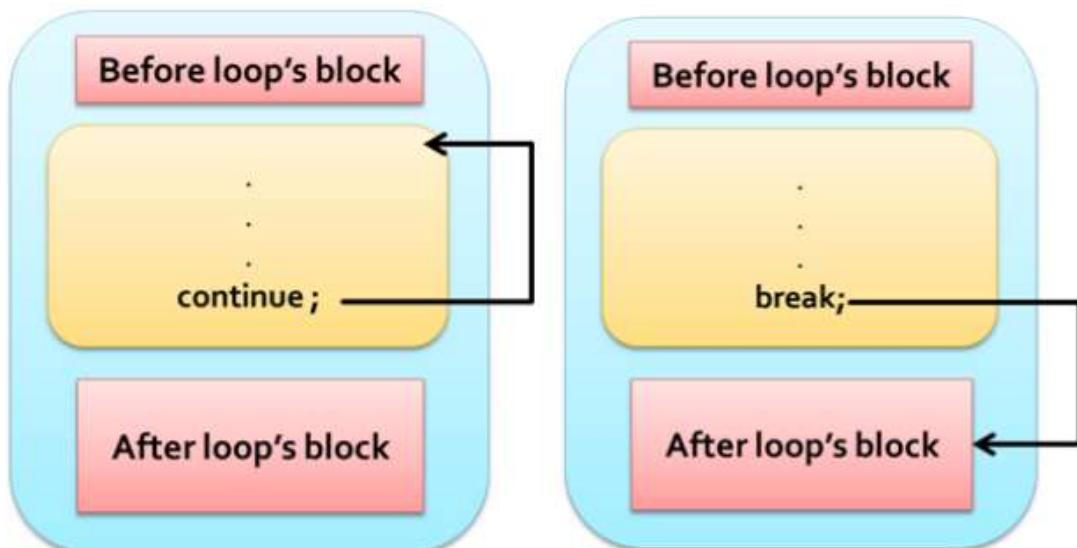
شكل 5.11: ناتج مثال 5.6

## 5.6 استخدام جمل break & continue

تطبيق التحكم في سير العمليات في لغة جافا ليس مقتصرًا على استخدام جمل اتخاذ القرار وجمل التكرار وإنما وفرت جملتين آخرتين للتحكم هما (*continue* و *break*). وقد سبق أن استخدمنا جملة (*break*) أثناء تطبيق جملة (*switch*) لإنتهاء الجملة وتغيير سير العمليات لخارج جملة (*switch*). في هذا الفصل سنلقي الضوء عليهما مع كيفية استخدامهما في جمل التكرار.

فعدد استخدام جملة (*break*) في إحدى جمل التكرار المختلفة يتم إيقاف التكرار منذ تلك الدورة ويتم توجيه التنفيذ إلى أول جملة بعد التكرار، كما تلاحظ في الشكل 5.12.

بينما استخدام الجملة (*continue*) في إحدى جمل التكرار يؤدي إلى تجاهل ما تبقى من الدورة الحالية التي ظهرت فيها هذه الجملة لتسألف جملة التكرار تنفيذها من الدورة التالية مباشرةً كما تلاحظ في الشكل 5.12.



شكل 5.12: منطق عمل جملة *break* يميّزه *continue* بسرا

المثال البرمجي 5.7 و 5.8 يوضحان لنا بشكل مفهوم كلاً منها وتأثيره على سير البرنامج.

### مثال توضيحي 5.7:

اكتب برنامج يستخدم التكرار لطباعة الأرقام من 1 إلى 10 ما عدا 7.

```

1 public class UsingContinue {
2     public static void main(String[ ] args){
3         for (int i=1; i<=10; i++)
4         {
5             if(i==7)
6                 continue;
7             System.out.println("i = "+i);
8         }
9     }
10 }
```

تابع معى في هذا المثال، ستجد أننا نريد طباعة كافة الأرقام من 1 إلى 10 ما عدا الرقم 7، وهذا يحتاج منا إلى تكرار لطباعة الأرقام إلا عندما نصل للرقم 7 فنريد أن نتجاهله ونطبع ما بعده، وهذا واضح في السطرين 5 و 6، فجملة (continue) ستجعل المترجم يتتجاهل الطباعة عندما نصل للرقم 7، تابع نتائج المثال في شكل

.5.13

General Output

Configuration:	
i = 1	لاحظ أن البرنامج
i = 2	تجاهل طباعة الرقم 7
i = 3	
i = 4	
i = 5	
i = 6	
i = 7	
i = 8	وطبع ما قبله وما بعده
i = 9	
i = 10	

Process completed.

شكل 5.13: تأثير استخدام الجملة continue

وليظهر الفارق واضحًا بين الجملتين؛ أعيد لك البرنامج ذاته ولكن باستخدام الجملة (break) وهي التي ستطيع في هذه الحالة الأرقام من 1 إلى 6 وستنتهي عمل جملة التكرار عند هذا الحد.

لا تنس أن لغة الجافا حساسة لحالة الأحرف وبالتالي لا بد أن تكتب كلاً من break و continue بالصورة ذاتها وبأحرف صغيرة وإنما يظهر خطأ برمجي.



**مثال توضيحي 5.8:**

اكتب برنامج يستخدم جملة دوران بها دورات من 1 إلى 10 لطباعة الأرقام من 1 إلى 7.

```

1 public class UsingBreak {
2     public static void main(String[ ] args){
3         for (int i=1; i<=10; i++)
4         {
5             if (i==7)
6                 break;
7             System.out.println("i = "+i);
8         }
9     }
10 }
```

General Output

Configuration:	
i = 1	لاحظ أن البرنامج توقف
i = 2	عند الرقم 7 أي طبع ما
i = 3	قبله ولم يطبع ما بعده
i = 4	
i = 5	
i = 6	

Process completed.

شكل 5.14: تأثير استخدام الجملة `break`

تعتبر كلا من `continue` و `break` جملتي برمجة مستقلتين وليس مجرد كلمات محوزة وبالتالي لا بد أن يتم وضع فاصلة منقوطة بعدها وإلا يظهر خطأ برمجي.

**5.7 أنواع الأخطاء البرمجية**

أثناء قيام المبرمج بمحاولاته المختلفة لتطوير البرامج والأنظمة، تصطدم محاولاته بظهور عدد من الأخطاء التي تحول بينه وبين إنجاز عمله قبل أن يتمكن من معالجتها، هذه الأخطاء تم تصنيفها إلى ثلاثة أنواع هي:

1. الخطأ البرمجي (*Syntax error*) : وهو خطأ ينبع عن خلل في كتابة مفردات ومكونات اللغة من حيث قاعدة كتابتها أو لفاظها، ويدخل في ذلك تغيير حالة الأحرف، استبدال أحرف، عدم موازاة الأقواس بحيث يكتب القوس الأول ولا يغلق وغيرهم، وهذا النوع من الأخطاء يظهر وقت ترجمة البرنامج وبالتالي يمكن اكتشافه، ويسبب في منع البرنامج من التنفيذ قبل أن يتدخل المبرمج لمعالجته.
2. الخطأ المنطقي (*Logical error*) : وهو خطأ ينبع غالباً عن خلل في تحليل المشكلة أو فهمها مما يجعل المبرمج يكتب جملة برمجية على غير المراد، فبدلاً من طباعة أكبر الأرقام يطبع أصغرها وقد يكون سبب ذلك وضع الفاصلة المنقطة في غير موضعها، وتبرز خطورة هذا الخطأ في أنه لا يمكن اكتشافه إلا عند مرحلة الاختبار.
3. الخطأ الاستثنائي (*Exception error*) : وهو خطأ ينبع عن خلل يحدث غالباً في الذاكرة أثناء التنفيذ، فيمنع استكمال التنفيذ، وهذا النوع من الأخطاء يمكن تقاديه بما يسمى "معالجة الخطأ الاستثنائي" قبل حدوثه. ومن الأمثلة على هذا النوع "القسمة على صفر" أو استقبال أرقام عشرية ومحاولة تخزينها في متغير صحيح، وسنعرض لأمثلة إضافية على هذا النوع من الأخطاء عند شرح موضوع المصفوفات.

#### 5.8 جمل التكرار المتداخلة

في هذا الفصل سنقوم بعرض وشرح مجموعة من الأمثلة التي تحتاج فيها إلى استخدام التكرار المتداخل أو المركب، وسيق أن شرحنا هذا المفهوم في الباب الثاني.

#### مثال توضيحي 5.9:

اكتب برنامج يستخدم جمل التكرار لطباعة الشكل التالي:

```

*   *   *   *
*   *   *   *
*   *   *   *
*   *   *   *

```

```

1 import javax.swing.JOptionPane;
2 public class stars8by4 {
3     public static void main(String[ ] args) {

```

```

4 String output= " ";
5 for (int i = 1; i<=4; i++){
6   for (int j= 1;j<=8; j++)
7     output = output+ "*      ";
8   output = output+"\n"; }
9 JOptionPane.showMessageDialog(null,output); } }
```

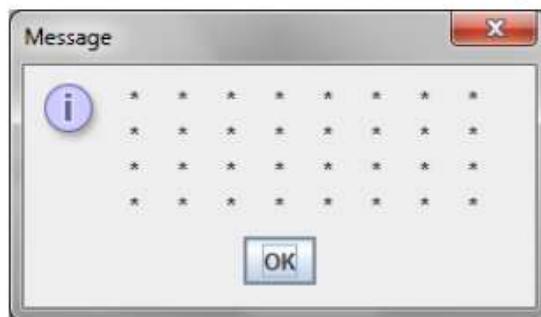
في هذا المثال نحتاج إلى رسم مجموعة من النجوم المرتبة في توزيعها في صفوف وأعمدة، وبالتالي نحن نريد تكرار عملية الطباعة على مستوى 4 أسطر، وداخل كل سطر على مستوى ثمانى نجوم يفصل بين كل واحدة والأخرى عدد من المسافات المتساوية.

للقيام بهذا الأمر وما يشابه من أشكال مختلفة، نحتاج إلى جملتي تكرار ، واحدة داخل الأخرى، فالأولى تكرار على مستوى الصنفوف وبالتالي تكرار من 1 إلى 4، بينما التكرار الذي بداخله على مستوى الأعمدة (طباعة النجوم) يكون من 1 إلى 8، وهذا هو مفهوم التكرار المركب، حيث يحتوي تكرار ضمن آخر.

يبقى أن تحكم المسافات وتحكم أنتا بعد طباعة النجمة الثامنة نتشى سطراً جديداً، أما الأولى فتتم من خلال طباعة النجمة وبعدها ثمانى مسافات وهذه تكتب مرة واحدة كما في السطر رقم 7، أما الثانية فتتم من خلال وضع "ln" بعد نهاية التكرار الخاص بكل صف من الصنفوف كما تشاهد في السطر رقم 8.

تابع معي أن التكرار الخارجي ينفذ داخله جملتين هما التكرار الداخلي وأمر السطر الجديد، بينما التكرار الداخلي لا ينفذ فيه إلا زيادة (رسم) النجمة تلو الأخرى كما في السطر رقم 7.

في النهاية يتم طباعة محصلة الأمر في السطر رقم 9، ولعلك انتبهت إلى أنها قمنا بتجميع النص اللازم للطباعة في متغير واحد كما شرحنا ذلك سابقاً.



شكل 5.14: ناتج مثال 5.9

**مثال توضيحي 5.10:**

اكتب برنامج يستخدم جمل التكرار جدول الضرب للأعداد من 1 إلى 3 على أن يطبع لكل رقم جدوله من 1 إلى 4، بحيث يظهر الناتج في صندوق حوار واحد.

هنا نحتاج إلى طباعة جدول الضرب الذي نعرفه بالصيغة التقليدية ( $2 \times 1 = 2$ ) وهكذا، هذه العملية نريد تكرارها لجدول الضرب 1 و 2 و 3 على أن يكون المدى فقط لأربعة أرقام، وبالتالي أحتج إلى تكرار خارجي أول ينتقل بي من الجدول الأول للثاني للثالث، ثم تكرار داخلي ليطبع لي مضروبات كل رقم من الأرقام الأربع المذكورة. إذا فكرت معي قليلاً، ستجد أن جدول الضرب للرقم 2 مثلاً هو عباره عن  $2 \times 1$  ثم  $2 \times 2$  ثم  $2 \times 3$  ثم  $2 \times 4$  وهذا يعني أن الرقم 2 ثابت والذي يتغير الرقم الثاني، وبعد ذلك سنضطر إلى تغيير الرقم 2 ليصبح 3 ونكرر العملية معه، هذا يأخذ بآيدينا إلى أن التكرار الخارجي سيكون لجدول الضرب والداخلي للأرقام الداخلية في كل جدول ضرب.

```

1 import javax.swing.JOptionPane;
2 public class MultiplicationTable{
3     public static void main(String[] args) {
4         String output = "";
5         int i=1;
6         while (i<=3) {
7             int j=1;
8             while (j<=4)
9             {
10                 output = output + i + " * " + j + " = " + i * j + "\n";
11                 j++;
12             }
13             output = output + "-----\n";
14             i++; }
15 JOptionPane.showMessageDialog(null,output); } }
```

### مثال توضيحي 5.11:

اكتب برنامج يستخدم جمل التكرار لطباعة الشكل التالي:

```
#####
#####
#####
##
#
#
```

كما تشاهد المطلوب طباعة عدد من الرموز بشكل غير متساوي، فالسطر الأول فيه 5 ثم 4 ثم 3 .... ثم 1 وهذا يعني أنه يسير بسلسل من أعلى إلى أسفل، ويعني أنه في السطر الأول يطبع خمس رموز بينما في السطر الخامس يطبع رمز واحد فقط.

هنا نحتاج إلى تكرارين، من النوع ([for](#)) على أن يتم استخدام التكرار إنقاذه العداد بوحد في كل جولة.

```
1 public class Sharp{
2     public static void main(String[ ] args){
3         for (int i=5; i>=1; i--)
4         {
5             for(int j= 1; j<=i; j++)
6                 System.out.print("#");
7             System.out.println();
8         }
9     }
10 }
```

النقطة التي تستحق الوقوف في هذا المثال، كيف يمكنني التحكم في عدد الرموز التي تطبع في كل سطر ولا تكون متساوية في كل الأسطر كما سبق في الأمثلة السابقة، الإجابة: فكر قليلاً، ستجد أن هناك علاقة بين عدد النجوم ورقم السطر، فهنا عدد النجوم تختلف رقم السطر، وبالتالي أنا أحتج لطباعة الرموز طالما العدد أقل من أو يساوي رقم السطر.

إذن الآن رقم السطر خصصنا له التكرار الخارجي وبالتالي فالشرط يكون طالما (العداد الداخلي=> العدد الخارجي) وهذا ما نراه في السطر رقم 5.

### 5.9 أمثلة وتدريبات عامة

#### 1. استخدم الكلمات المناسبة لتعبئه الفراغات في الجمل التالية:

- أ- العمل على تنفيذ جملة أو مجموعة من الجمل البرمجية ككتلة واحدة لعدد من المرات أو بناء على شرط منطقي معين يعرف باسم \_\_\_\_\_.
- ب- من التطبيقات التي تحتاج فيها لاستخدام مفهوم التكرار هي \_\_\_\_\_ و\_\_\_\_\_.
- ت- التكرار يحكمه ثلاثة نقاط هامة هي \_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_.
- ث- بناء على وقت فحص شرط الاستمرارية، تعرف جملة **while** باسم \_\_\_\_\_ بينما جملة \_\_\_\_\_ باسم **do...while**.
- ج- الخطأ الذي يتسبب في استمرار تنفيذ البرنامج دون توقف يصنف على أنه \_\_\_\_\_ ويعرف باسم \_\_\_\_\_.
- ح- وضع فاصلة منقوطة بعد شرط **while** يؤدي إلى \_\_\_\_\_ بينما وضعها بعد شرط \_\_\_\_\_ يؤدي إلى **do...while**.

#### 2. حدد صحة أو خطأ كلٍ من العبارات التالية مع التعليل:

- أ- تعتبر جملة **while** من جمل التكرار التي تعمل على فحص شرط الاستمرارية قبل البدء في تنفيذ أي تعليمية برمجية تابعة للتكرار بعكس جملة **for**.
- ب- استخدام جملة التكرار **for** مناسبة للتطبيقات المحددة بدايتها نهايتها بالأرقام.
- ت- عدم تعديل قيمة متغير التكرار يؤدي إلى خطأ استثنائي.
- ث- استخدام جملة **break** و **continue** في جمل التكرار يؤدي إلى انهاء الدورة الحالية فقط.
- ج- استبدال الفاصلة المنقوطة في جملة **for** بفاصلة (,) يؤدي لخطأ منطقي.

#### 3. مستخدماً التكرار، عَرِّ عن الجمل التالية بجمل مكافئة بلغة جافا:

- أ- طباعة سلسلة أفقية من النجوم عددها 10
- ب- طباعة سلسلة من رموز & بشكل رأسى عددها 10

ت- طباعة مستطيل من النجوم أبعاده (8×5)

4. حدد الأخطاء ونوعها في الجمل البرمجية التالية مع التصحيح:

أ-

```
int x =10;
do {
    x++;
} while (x<=100)
```

ب-

```
for (int i=1, i<=10, i--)
{ System.out.println("i = "+i);}
```

ت- الجملة التالية معدة لطباعة مضاعفات الرقم 5 من 1 إلى 100

```
for(int i=1; i<=100;i=i+5)
System.out.println("i = "+i);
```

5. اكتب برنامج يقوم بطباعة الأرقام من 1 إلى مليون باستخدام جملة طباعة واحدة فقط.

6. اكتب برنامج يقوم باستقبال خمسة أرقام صحيحة موجبة من المستخدم ويطبع أكبرها على أن تستخدم جملة إدخال واحدة فقط.

7. مستخدماً جملة تكرار تسمح بتنفيذ جملة واحدة على الأقل، اكتب برنامج يقوم بطباعة الأرقام الزوجية من 1 إلى .50

8. اكتب برنامج يستقبل من المستخدم عدد من الأرقام الصحيحة ويطبع أصغرها على أن يكون الرقم الأول هو عدد الأرقام التي سيدخلها المستخدم للبرنامج.

9. اكتب برنامج يقوم بحساب متوسط الأعداد الصحيحة من 1 إلى .50.

10. اكتب برنامج يستقبل من الطالب خمس درجات لمساقاته ليطبع مجموعة درجاته ومتوسطه.

## 5.10 تمارين ذاتية الحل

1. حدد الأخطاء في الجمل التالية مع ذكر نوعها وتصحيحها:

أ-

```
int x = 1;
    total;
while (x <= 10)
{
total += x;
++x;
}
```

ب-

```
while (x <= 100)
    total += x;
++x;
```

ت-

```
while (y > 0)
{
    System.out.println(y);
++y;
```

ث-

```
for (i = 100, i >= 1, i++)
System.out.println(i);
```

ج - الكود التالي معد لطبع الأعداد الفردية من 1 إلى 19

```
for (i = 19; i >= 1; i += 2)
```

```
System.out.println(i);
```

ح- الكود التالي معد لطباعة الأعداد الزوجية من 2 إلى 100

```
counter = 2;
do
{
    System.out.println(counter);
    counter += 2 ;
} while (counter < 100);
```

2. حول الجمل التالية إلى جمل برمجية متوافقة مع قواعد لغة Java

- أ- طباعة مضاعفات العدد 5 من 0 إلى 100.
- ب- طباعة الكلمة ("java") طالما كان يدخل المستخدم الرقم 1.
- ت- طباعة خمس نجوم أفقية يفصل بينها ثمانى مسافات، مستخدما جملة طباعة واحدة ونجمة واحدة.
- ث- طباعة عشر صفوف من النجوم بحيث يكون عدد النجوم في كل صف ضعف رقم الصف، مستخدما جملة طباعة واحدة ونجمة واحدة.
- 3. اكتب برنامج يقوم باستقبال عدد من الأرقام الصحيحة بين 0 و100 ويطبع أكبرها وأصغرها على اعتبار أن الرقم المدخل الأول هو عدد الأرقام التي يمكنه إدخاله.
- 4. اكتب برنامج يستقبل من الطالب درجاته ليطبع البرنامج مجموع درجاته ومتوسطه والتقدير، علما بأنَّ الطالب سيدخل عدد مساقاته.
- 5. اكتب برنامج لطباعة مربع الأرقام من 1 إلى 15 مع تجاهل مربعات الأرقام التي تقبل القسمة على 3.  
(ستستخدم جملة *continue*)
- 6. اكتب برنامج لطباعة الأرقام الأولية من 1 إلى 100، (علما بأنَّ الرقم الأولي هو الذي يقبل القسمة فقط على 1 ونفسه مثل 7 و11 و13)
- 7. في الباب الثاني، طلب منك رسم مخطط سير العمليات لتعبئة الخزان بـشكل تلقائي، اكتب برنامج يكافئ هذا النظام.

8. يمتلك تاجر مجموعة قطع قماش طول الواحدة يزيد عن 5 أمتار ويرغب في تطوير نظام محاسب لقطعها إلى قطع طول الواحدة منها 5 أمتار فقط، استخدم جملة `do...while` لحل المشكلة.
9. قامت أحد الجمعيات الخيرية بالإعلان عن مشروع خيري لإقراض الشباب لإنشاء المشاريع الصغيرة على فرض أن السن المناسب للاستفادة من الإقراض من 20 إلى 27 عاماً، اكتب البرنامج اللازم لحوسبة هذا المشروع علماً بأن عدد المتقدمين للمشروع هو 200 شاب.
10. قارن بين أنواع الأخطاء المختلفة في البرمجة.
11. مستقidiأ من طريقة تتبع القطع البرمجية والخوارزميات التي تم شرحها في الباب الثاني، تتبع المثال التوضيحي 5.9.
12. افحص القطعة البرمجية التالية، هل الشرط صحيح دائماً، خطأ دائماً أم يتغير أحياناً وذلك عند النقاط الثلاث المحددة في القطعة.

```
int count = 0;
while (count < 100) {
    // النقطة الأولى
    System.out.println("Welcome to Java!");
    count++;
    // النقطة الثانية
}
// النقطة الثالثة
```

13. افترض أن الأرقام المدخلة هي 4، 5، 6، 2 و 0 ما هي المخرجات في القطعة البرمجية التالية:

```
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number, max;
        number = input.nextInt();
        max = number;
        while (number != 0) {
```

```

number = input.nextInt();
if (number > max)
    max = number;
}
System.out.println("max is " + max);
System.out.println("number " + number);
}
}

```

**للذكرى:** الأمر البرمجي **Scanner** يستخدم لإتاحة الفرصة للمستخدم لإدخال قيم في بيئة **Console** وإدخال أرقام متتالية يتم استخدام الأمر:

**number = input.nextInt();**



14. كم مرة سيتم تنفيذ أمر الطباعة **println** في القطعة البرمجية التالية:

```

for (int i = 0; i < 10; i++)
    for (int j = 0; j < i; j++)
        System.out.println(i * j);

```

## اجابات التدريبات العامة:

1. (أ) التكرار (ب) تعبئة الخزنات بشكل أوتوماتيكي، حساب المضروب (ت) نتيجة الشرط المنطقى، عامل يؤثر على نتيجة الشرط، قيمة بدائية للعامل (ث) Post-test, Pre-test (ج) منطقى، تكرار لا نهائى. (ح) خطأ منطقى، عمل البرنامج بشكل سليم

2. حدد صحة أو خطأ كل من العبارات التالية مع التعليل:

أ- (خاطئة) تعتبر جملتي `for` و `while` من جمل التكرار التي تعمل على فحص شرط الاستمرارية قبل البدء في تنفيذ أي تعليمية برمجية تابعة للتكرار.

ب- (صحيحة).

ت- (خاطئة) عدم تعديل قيمة متغير التكرار يؤدي إلى خطأ منطقى.

ث- (خاطئة) استخدام جملة `continue` في جمل التكرار يؤدي إلى إنتهاء الدورة الحالية فقط واستكمال ما يلى بينما استخدام جملة `break` ينهى التكرار نهائياً.

ج- (خاطئة) استبدال الفاصلة المنقوطة في جملة `for` بفاصلة (،) يؤدي لخطأ برمجي.

3. عَبَرْ عن الجمل التالية بلغة جافا:

أ- طباعة سلسلة أفقية من النجوم عددها 10

```
for(int i=0; i<=9; i++)
    System.out.print("*");
```

ث- طباعة سلسلة من رموز & بشكل رأسى عددها 10

```
for(int i=0; i<=9; i++)
    System.out.println("&");
```

ب- طباعة مستطيل من النجوم أبعاده (8×5)

```
for(int i =1; i<5; i++)
{
    for (int j=1; j<8; j++)
```

```
System.out.print("*");
System.out.println();
{
```

.4

أ- الخطأ: وضع فاصلة منقوطة بعد الشرط

نوع الخطأ: خطأ منطقى

```
if (c < 7)
```

```
JoptionPane.showMessageDialog (null, "c is less than 7");
```

ب- الخطأ: عدم وضع فاصلة منقوطة بعد الشرط

نوع الخطأ: خطأ برمجي

```
int x = 10;
```

```
do {
```

```
x++;
```

```
} while (x<=100);
```

ت- الخطأ:

1. استبدال الفاصلة المنقوطة بين أجزاء جملة for بفاصلة

2. الشرط المنطقى غير متوافق مع تعديل قيمة المتغير

نوع الخطأ:

1. خطأ برمجي

2. خطأ منطقى

```
for (int i=1; i<=10; i++)
```

```
{ System.out.println("i = "+i);}
```

ث- الخطأ:

1. لا يطبع المراد (القيمة البدائية خاطئة)

نوع الخطأ:

1. خطأ منطقي

```
for(int i=5; i<=100;i+=5)
```

```
System.out.println("i = "+i);
```

```
public static void main(String[] args) {
    for (int counter=1; counter<1000000; counter++)
        System.out.println(counter);
}
```

شكل 5.15: حل تمرين 5

```
public static void main(String[] args) {
    int max=0;
    String input;
    int x;
    for ( int counter=1; counter<=5; counter++)
    {
        input = JOptionPane.showInputDialog("Enter the"+counter+"th number");
        x = Integer.parseInt(input);
        if (x > max)
            max =x;
    }
    JOptionPane.showMessageDialog(null, "The Final min number is "+max);
}
```

شكل 5.16: حل تمرين 6

```
public static void main(String[] args) {
    int counter =1;
    String even ="Events\n";
    while (counter<=50)
    {
        if(counter%2==0)
            even= even+counter+"\n";
        counter++;
    }
    JOptionPane.showMessageDialog(null,even);
}
```

شكل 5.17: حل تمرين 7

```

public static void main(String[] args) {
    int min =1000000;
    String input;
    int counter =1;
    int no, next;
    input = JOptionPane.showInputDialog("Enter the numbers you wanted to input");
    no = Integer.parseInt(input);
    while (counter<=no)
    {
        input = JOptionPane.showInputDialog("Enter the next number");
        next = Integer.parseInt(input);
        if (next < min)
            min = next;
        counter++;
    }
    JOptionPane.showMessageDialog(null, "The min is "+min);
}

```

شكل 5.18: حل تمرين 8

يمكنك حل التمرين ذاته بطريقة أخرى، لأن تجعل أول رقم من أرقام المنافسة هو الأكبر بدلاً من جعل الرقم مليون هو الأكبر، لأن المستخدم قد يدخل كل الأرقام أكبر من مليون.

```

public static void main(String[] args) {
    int counter =1;
    int sum =0;
    double avg;
    do{
        sum = sum +counter;
        counter++;
    } while(counter<=50);
    avg = sum/50f;
    JOptionPane.showMessageDialog(null,"The sum is "+sum);
    JOptionPane.showMessageDialog(null,"The avg is "+avg);
}

```

شكل 5.19: حل تمرين 9

```

public static void main(String[] args) {
    String input;
    int grade;
    int counter =1;
    int sum =0;
    double avg;
    do{
        input = JOptionPane.showInputDialog("enter next grade");
        grade = Integer.parseInt(input);
        sum = sum +grade;
        counter++;
    } while(counter<=10);
    avg = sum/10f;
    JOptionPane.showMessageDialog(null,"The sum is "+sum);
    JOptionPane.showMessageDialog(null,"The avg is "+avg);
}

```

شكل 5.20: حل تمرين 10

كثير من الأعمال تتوقف أو تتأخر لأنها غير منتظمة؛ بمجرد أن تنتظم عناصرها يصبح العمل أكثر قابلية للنجاح وفي وقت أقصر.

## الباب السادس

### المصفوفات

#### (Arrays)

يتوقع من الطالب في نهاية هذا الباب أن:

- يدرك مفهوم المصفوفات وتنظيمها في البرمجة.
- يُعدد أنواع المصفوفات.
- يذكر مشاكل برمجية تحتاج في حلها لاستخدام المصفوفات.
- يستخدم المتغير المتسلسل في معالجة عناصر المصفوفات.
- يُحسن استخدام جملة التكرار *for* المحسنة للمصفوفات.
- يُتقن تنفيذ عمليات معالجة مختلفة على عناصر المصفوفات.

يتوفر لهذا الباب شرائح العرض (PowerPoint) وكذلك ملفات مرئية من إعداد مؤلف الكتاب؛ يمكنك الحصول عليها من خلال الموقع الأكاديمي للمؤلف [mfarra.cst.ps](http://mfarra.cst.ps)



## الباب السادس: المصفوفات

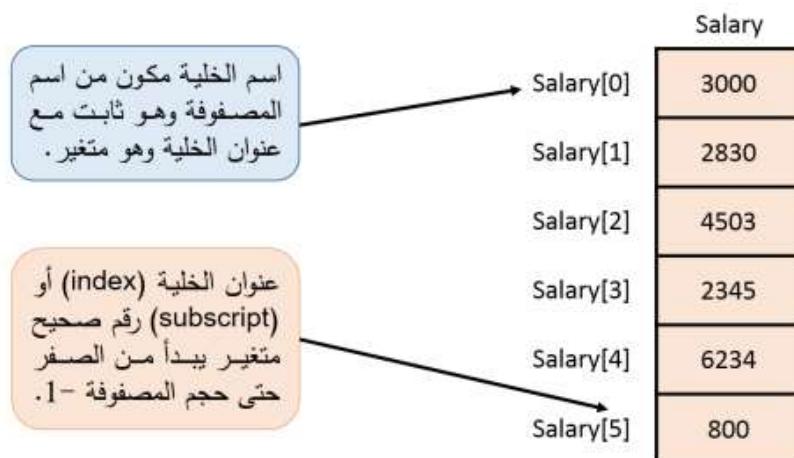
### 6.1 مقدمة

في الأبواب الماضية تمكناً من تعريف المتغيرات وإعطاء كل متغير قيمةٍ مفردة، ولم يكن بمقدورنا أن نعرف عدداً كبيراً من المتغيرات بجملة واحدة ثم يصبحوا جميعاً متاحين لعمليات البحث أو القراءة أو الطباعة أو الترتيب أو نحوهم، فكل متغير يتم تعريفه لا يرتبط منطقاً بالمتغير الآخر مما يجعل الانتقال بين المتغيرات لا يتمتع بالمرنة الكافية. فعندما تكتب 10 متغيرات وتريد أن تبحث بينهم عن قيمة محددة أو ترتيبهم ترتيباً محدداً فلا بد لك أن تمر عليهم واحداً تلو الآخر وبالاسم وهذا أمرٌ مُجهد جداً وقد يستحيل إنجازه برمجياً مع آلاف المتغيرات لهذا فإن البرمجة بلغة جافا تتيح لنا مجموعة من تراكيب البيانات التي تسمح لنا بتخزين أكثر من قيمة من نوع واحد في هيئة محددة تساعدنا في التنقل بينها بسهولة؛ ومن أشهر هذه التراكيب ما يُعرف باسم المصفوفات والتي سنناقش مفهومها وأنواعها في الفصل 6.2، ثم نقوم بتعريف كيفية تطبيقها في بُعد واحد (One-Dimensional) في الفصل 6.3، قبل أن نستعرض الصيغة المحسنة للمصفوفات من الجملة التكرارية (Multi-Dimensional) في الفصل 6.4، بينما نتعرف على كيفية تعريف وتطبيق المصفوفات متعددة الأبعاد (for

.6.5

### 6.2 مفهوم المصفوفات وال الحاجة إليها

المصفوفة هي إحدى تراكيب البيانات المشهورة والمشتقة من علم الرياضيات، حيث يتم فيها رصّ القيم في خلايا بعنوان متسلسلة يزيد فيها عنوان الخلية عن التي قبلها بواحد ويقل عن الذي بعده بواحد كما تشاهد في الشكل 6.1.



شكل 6.1: المفهوم العام للمصفوفات

والمصفوفات تسمح بتخزين نوع واحد فقط من البيانات في كل خلاياها، فلا يمكن في لغة جافا بطريقة مباشرة أن تُثبّت مصفوفة تحتوي على خليط من قيم صحيحة (*int*) وأخرى عشرية (*float*) أو قيم صحيحة صغيرة (*short*) وأخرى صحيحة كبيرة (*long*)، ولكن يمكننا أن نستخدمها لتخزين نوع واحد في المصفوفة الواحدة لأنّ تُصَنَّع مصفوفة للنصوص وأخرى للأرقام الصحيحة الصغيرة وهكذا.

**المصفوفة هي أحد تركيب البيانات ثابتة الحجم** (*static memory allocation*)  
والمكونة من مجموعة متصلة ومتواصلة من موقع الذاكرة لها نوع واحد واسم واحد.



وقد ظهرت الحاجة للمصفوفات في علم البرمجة كنتيجة طبيعية للتطور في تلبية احتياجات الزبائن ومحاولة المبرمجين لإيجاد الحلول للمشاكل التي تتعامل مع بيانات كثيرة تربطها علاقة منطقية ما وتحتاج لمعالجتها جميعاً مثل (ترتيبهم، البحث عن قيمة ما بينهم، تصنيفهم حسب شرط محدد) أو غير ذلك من عمليات المعالجة؛ وهذا لم يكن ممكناً من خلال مجموعة متغيرات متفرقة يصعب التنقل بينهم إلا بذكر كل متغير باسمه. وبالنظر إلى طبيعة المشاكل التي يمكن للمصفوفات المساهمة في حلها، نجد أنها تأخذ هيئتتين هما القيم في اتجاه واحد (رأسي أو أفقي) والقيم في اتجاهين على هيئة جدول ذي بعدين أو أكثر؛ ومن هذه النظرة فقد تم تقسيم المصفوفات إلى:

1. مصفوفات أحادية الأبعاد يُخزن بها مثلاً أرقام الهاتف أو أسماء موظفين أو درجات النجاح لطالب ما أو غير ذلك.
2. مصفوفات متعددة الأبعاد يُخزن بها مثلاً جدول المواعيد أو جدول المحاضرات أو أبعاد أشكال هندسية أو غير ذلك.

وسنعرف على كل نوع منها في الفصول التالية.

### 6.3 تعريف وإنشاء المصفوفات أحادية الأبعاد

لأن المصفوفات لا تعتبر نوعاً أساسياً بل هي نوع مشتق يتم اشتقاقه من الأنواع الأساسية، فإنّ تعريفها لا يتم بشكل مماثل لتعريف المتغيرات، وبالتالي فإننا لتعريف مصفوفة من النوع (*type*) مثلاً وحجمها (*S*) نكتب التالي:

```
type [ ] array_name = new type[S];
```

وهذا يتم بقاعدة عامة موضحة بالشكل 6.2، حيث إنّ:

.*String*, *float*, *int*: هو النوع الذي نريد اشتقاق المصفوفة منه مثل .*String*, *float*, *int* .  
[ ]: هذه الأقواس تستخدم لتمييز تعريف مصفوفة على أنه ليس متغير.

array\_name: هو الاسم التعريفي للمصفوفة، فيتم مثلاً إعطاء الاسم employee أو sal أو غير ذلك.  
new: كلمة محفوظة تُستخدم لتعريف أنواع مشتقة.  
S: حجم المصفوفة وهو عدد العناصر التي تحتاج إلى حجزها.

**type[] array\_name = new type[ x ];**

شكل 6.2: كيفية إنشاء مصفوفة أحادية الأبعاد

وأمثلة ذلك:

- إنشاء مصفوفة صحيحة تضم 10 درجات لطالب.

```
int[ ] grades = new int[ 70 ];
```

- إنشاء مصفوفة نصية لأسماء الموظفين تضم 10 موظفين.

```
String[] employees = new String[ 10 ];
```

- إنشاء مصفوفة تحتوي على مرتبات ثلاثة موظف.

```
float[] salary = new float[ 30 ];
```

يقصد بـ**متغير البيانات الثابتة (static)** أنها لا يمكن تغيير حجمها أثناء مرحلة التنفيذ، بل حجمها يبقى ثابتاً على القدر الذي تم تحديده أثناء كتابة البرنامج.



كما يمكن تعريفها على مراحلتين، بحيث يتم إنشاءها ثم حجز المساحة لها كما في الشكل 6.3.

**type [] array\_name;  
array\_name = new type[ x ];**

شكل 6.3: كيفية إنشاء مصفوفة أحادية الأبعاد

كما يمكن تعريفها وإعطاؤها القيم مباشرة في جملة واحدة، وذلك يحدث عندما تكون على علم مسبق بالعناصر التي نريد تخزينها في المصفوفة، كما هو موضح في الشكل 6.4.

**type[] array\_name = {a1, a2, a3,...};**

العناصر التي يتم تخزينها في المصفوفة،  
مع فصل كل عنصر عن الآخر بفواصل مع  
إحاطة الجميع بالاقواس { }.

شكل 6.4: كيفية إنشاء مصفوفة أحادية الأبعاد

ويمكن بعد إنشائها التفاعل مع خلاياها من خلال اسم المصفوفة ثم رقم الخلية محاطاً بالأقواس [ ] كما يظهر بالشكل 6.5، ومثال ذلك إذا كان لدينا مصفوفة نصية اسمها (names) ونريد التفاعل مع الخلية الخامسة: names[4] = "Ahmad"; والخلية الخامسة يعبر عنها برقم 4 والخلية الأولى يعبر عنها برقم 0 وذلك لأنَّ الترقيم يبدأ من 0.



شكل 6.5: كيفية إعطاء القيم لخلية من خلايا المصفوفة

وب مجرد إنشاء المصفوفة فإنها تأخذ القيم الافتراضية للنوع، فمثلاً القيمة الافتراضية النوع الصحيح هو الصفر، وللنوع `String` هو القيمة `null`، وللنوع العشري هو 0.0 بينما النوع `char` لا قيمة افتراضية له. بينما إن كناً نحتاج إلى إعطاء قيم بدائية للمصفوفة فيمكن فعل هذا من خلال استخدام جمل الدوران ولتكن `for`، فكما تعلم جمل الدوران يمكنها تكرار عملية محددة بناءً على رقم متسلسل، والمصفوفات تحتوي على رقم متسلسل يبدأ من الصفر وحتى الحجم - 1 ولذلك يمكننا التفاعل ومعالجة خلايا المصفوفة دائمًا من خلال تكرار يبدأ من الصفر حتى الحجم - 1 أو أي علاقة أخرى مكافئة لهذه العلاقة، في المثال 6.1 نطبع القيم الافتراضية لمصفوفات من نوع `char`, `String`, `float`, `int` بينما في المثال 6.2 نقوم بإعطاء قيمة افتراضية قدرها 100 لمصفوفة من نوع `int`.

**عدم إعطاء قيمة لحجم المصفوفة عند التعريف ينتج عنه خطأ برمجي.**



### مثال توضيحي 6.1:

قم بإنشاء مصفوفة من الأنواع `char`, `String`, `float`, `int` حجمها 3 خلايا، وقم بطباعة قيمها الافتراضية.

```
1 public class Emp_array {
2     public static void main(String[] args) {
```

```

3 int [ ] sal = new int [5];
4 for (int i=0; i<=4; i++)
5     System.out.println("i: "+sal[i]);
6 String [ ] name = new String[5];
7 for (int i=0; i<=4; i++)
8     System.out.println("i: "+name[i]);
9 float [ ] f = new float[5];
10 for (int i=0; i<=4; i++)
11     System.out.println("i: "+f[i]);
12 char [ ] let = new char[5];
13 for (int i=0; i<=4; i++)
14     System.out.println("i: "+let[i]);
15 }
16 }
```

كما تلاحظ في الأسطر 5 و 8 و 11 و 14 فإننا لطباعة الخلية كتبنا اسم المصفوفة ثم المتغير الخاص بالدوران بين الأقواس وفي كل دورة من دورات الدوران يتم طباعة قيمة من قيم الخلية بداية من الخلية الصفرية (الأولى)، الشكل 6.6 يظهر ناتج الطباعة.

```

General Output
-----
i: 0
i: 0
i: 0
i: null
i: null
i: null
i: 0.0
i: 0.0
i: 0.0
i:
Process completed.
```

شكل 6.6: ناتج مثال 6.1

**مثال توضيحي 6.2:**

قم بإنشاء مصفوفة لأعداد صحيحة حجمها 10 خلايا، مع إعطاء القيمة 100 لجميع خلاياها ثم طباعتها.

```

1 public class Example6_2 {
2     public static void main(String[ ] args) {
3         int [ ] num = new int [10];
4         for (int i=0; i<=9; i++)
5             num[i] = 100;
6         for (int i=0; i<=9; i++)
7             System.out.println("i: "+num[i]);
8     }
9 }
```

كما تلاحظ في السطر 4 و5، فإن الدوران يبدأ من صفر حتى أصغر من أو يساوي 9، وذلك لأنَّ عدد خلايا المصفوفة هي 10 خلايا، في كل دورة من الدورات يتم إعطاء القيمة 100 لخلايا المصفوفة واحدة واحدة بداية من الخلية رقم 0 (الأولى) حتى الخلية رقم 9 (العاشرة).

بينما في السطر 6 و7، فإنَّ الدوران يبدأ أيضاً من صفر حتى 9 لطباعة القيم التي تحتويها الخلايا من خلال وضع اسم المصفوفة ومتغير الدوران مثل (num[i]) وهو يعامل كأنه متغير تتم طباعته القيمة التي بداخله. في المثال التالي، سنقوم بتخزين الأرقام الزوجية من 0 وحتى 100 في مصفوفة من النوع byte، وهذا يحتاج مثلاً إلى البحث عن علاقة تربط فيها بين الدوران المتسلسل وتسلسل المصفوفة من جهة وتسلسل الأرقام الزوجية من جهة أخرى، فعند الملاحظة تجد أنَّ الأرقام الزوجية من 0 إلى 100 هي عبارة الأرقام من 0 إلى 50 مضروبة في الرقم 2 وهذا الناتج سيتم تخزينه في المصفوفة كما تلاحظ في المثال 6.3.

**مثال توضيحي 6.3:**

قم بإنشاء مصفوفة صحيحة لتخزين الأرقام الزوجية من 0 إلى 100.

```

1 public class Example6_3 {
2     public static void main(String[ ] args) {
3         int [ ] num = new int [51];
4         for (int i=0; i< num.length; i++)
5             num[i] = i*2;
6         for (int i=0; i<=50; i++)
7             System.out.println("num["+i+"]: "+num[i]);
8     }

```

في السطر رقم 7 من هذا المثال، ستجد أننا نريد طباعة القيم المخزنة داخل خلية المصفوفة، ولكننا نريد طباعتها بالصورة `num[0]` أي يذكر لي اسم الخلية ورقمها ثم قيمتها المخزنة، وهذا يتم بسهولة من خلال وضع الاسم كنص لأنه ثابت، بينما الرقم المتسلسل نخرجه خارج النص ونضع المتغير `i` ليتم طباعته في كل دورة.

**خطأ استثنائي:** عند المحاولة لمعالجة خلية خارج حدود المصفوفة، يصدر البرنامج خطأ استثنائي، لأن المترجم لا يجد الخلية التي يريدها البرنامج، واسم الخطأ في هذه الحالة `ArrayIndexOutOfBoundsException` (حاول استبدال الحد الأعلى لأي من جمل الدوران السابقة في مثال 7.3 `for (int i=0; i<=50; i++)` وراقب الناتج).



وضع الأقواس `[ ]` قبل اسم المصفوفة أو بعدها لا يؤثر ولا ينتج عنه أي أخطاء.



المصفوفات تحتوي على متغير ينشأ بشكل ضمني عند تعريف المصفوفة ليخزن فيه حجم المصفوفة واسمها `length` ويتم استخدامه كأحد خصائص كانن المصفوفة.



وفي نهاية هذا الفصل عليك أن تعلم أن كافة المشاكل البرمجية التي يحتاج حلها إلى مصفوفات، يكون مفتاح الحل فيها متعلقاً بكيفية الربط بين الرقم المتسلسل للمصفوفة مع البيانات التي تحتاج تخزينها أو عرضها أو معالجتها، وسنستعرض في الأمثلة الخاصة بالتدريبات عدداً آخر من هذه التدريبات.

#### 6.4 جملة `for` التكرارية المحسنة للمصفوفات

نظرًا للعلاقة الوثيقة بين جمل الدوران وخاصة جملة `for` مع المصفوفات وعناصرها، وأنها الأداة التي من خلالها يمكننا معالجة خلية المصفوفات بطرق مختلفة، فقد أتاحت لغة جافا صورة محسنة من جملة `for`.

وستخدم لمعالجة عناصر المصفوفة، وهي مخصصة على قدر عناصر المصفوفة مما يحد من الأخطاء الاستثنائية.

```
for ( parameter : array_name )
    statement
```

شكل 6.7: تركيبة جملة الدوران `for` المحسنة

حيث إن:

`parameter`: نوع العناصر المخزنة في المصفوفة ويتم كتابتها مثل تعريف المتغيرات على الصورة (`int x`).  
`arrayName`: اسم المصفوفة التي نود معالجتها عناصرها.  
وبالتالي فنحن في هذه التركيبة نستغنّي عن تعرّيف عدد مخصوص مع تحديد بدايته ونهايته والشرط المنطقي اللازم له.

#### مثال توضيحي 6.4:

قم بإنشاء مصفوفة صحيحة تحتوي مسبقاً على 10 عناصر ثم استخدم التركيبة المحسنة لجملة الدوران `for` لجمع هذه العناصر وطباعة الناتج النهائي.

```
1 public class Example6_4 {
2     public static void main(String[ ] args) {
3         int mul3 [ ] = {2, 34, 87, 2, 3, 41, 32, 44, 31, 9};
4         int total =0;
5         for(int x: mul3)
6             total = total+x;
7         System.out.print("sum: "+total);
8     }
}
```

#### 6.5 تعريف وإنشاء المصفوفات متعددة الأبعاد

المصفوفة متعددة الأبعاد هي إحدى محاولات البرمجة لمحاكاة الواقع، حيث إن الحياة من حولنا مليئة بالمصفوفات متعددة الأبعاد وفي أبسط صورها تكون ثنائية الأبعاد مثل الجدول الدراسي وجدول المواعيد، والكثير

من التطبيقات التي تتكون من مجموعة من الصنفوف والأعمدة كما في الشكل 6.8 حيث إن تقاطع الصنف مع العمود يُعتبر خلية عنوانها يتكون من رقم الصنف مع رقم العمود على الترتيب، ولهذا فلغة جافا قد أتاحت هذا النوع من المصفوفات، ويتم تعريفه كما بالشكل 6.9 حيث إن  $\times$  عدد الصنفوف ولا هو عدد الأعمدة.

المصفوفة A

	العمود رقم 0	العمود رقم 1	العمود رقم 2	العمود رقم 3
الصنف رقم 0	A[0,0]	A[0,1]	A[0,2]	A[0,3]
الصنف رقم 1	A[1,0]	A[1,1]	A[1,2]	A[1,3]
الصنف رقم 2	A[2,0]	A[2,1]	A[2,2]	A[2,3]

اسم المصفوفة
عنوان الصنف
عنوان العمود

شكل 6.8: المفهوم العام للمصفوفات ثنائية الأبعاد

```
type [,] array_name = new type[ x, y ];
```

شكل 6.9: كيفية إنشاء مصفوفة ثنائية الأبعاد

علما بأأن الطرق الأخرى لإنشاء المصفوفة الأحادية متاحة أيضاً للمصفوفة الثنائية كما في الشكل 6.10.

```
Type name[][];  
name = new Type [ r ][ c ];
```

```
Type name [][] = { { C0, C1, C2 }, {C0, C1, C2}, ... };  
Row0                                                 Row1
```

شكل 6.10: كيفية إنشاء مصفوفة ثنائية الأبعاد

### مثال توضيحي 6.5:

قم بإنشاء مصفوفة ثنائية الأبعاد لتخزين خمس درجات لثلاثة طلبة.

```

1 public class Example6_5 {
2     public static void main(String[] args) {
3         float [ ][ ]marks = {{70, 80, 68, 78, 67}, {80, 90, 78, 77, 89}, {98, 87, 78,
4             65, 90}};
5     }

```

في هذا المثال، تم إعطاء 5 درجات لثلاثة طلبة، ويتم تخزينها في مصفوفة ثنائية الأبعاد، أبعادها (3) صفوف، (5) أعمدة، ويمكن إعطاؤها بالطريقة الموضحة في الشكل 6.9، حيث إن كل صف يتم إعطاؤه بين {}، كما يمكن أن تكون الصدوف غير متساوية مع عدد الأعمدة، بمعنى صف يحتوي على ثلاثة قيم وآخر على خمسة قيم، إلا أن أبعاد المصفوفة تأخذ على الأبعاد الأكبر.

#### **مثال توضيحي 6.6:**

قم بطباعة عناصر المصفوفة التي تم إنشاءها في مثال 6.5.

```

1 public class Example6_6 {
2     public static void main(String[ ] args) {
3         float [ ][ ]marks = {{70, 80, 68, 78, 67}, {80, 90, 78, 77, 89}, {98, 87, 78,
4             65, 90}};
5         for (int i= 0; i< 3; i++) {
6             System.out.print((i+1)+" Student's marks:\n");
7             for (int j=0; j<5 ; j++)
8                 System.out.print(marks[i][j]+"\t");
}

```

9      **System.out.println();**

نهاية جملة الدوران الخارجية // { 10 }

نهاية الدالة الرئيسية// 11 }

12 }

عند الرغبة في معالجة خلايا المصفوفة ثنائية الأبعاد، تذكر دائمًا أنها عبارة عن مستطيل أو مربع، له طول وعرض، أو صفوف وأعمدة، وبالتالي نحن في هذه الحالة بحاجة إلى جملتي تكرار، الأولى خارجية للدوران على الصدوف والثانية داخلية للدوران على الأعمدة وبذلك يصبح لديك في الدورات الداخلية القدرة على معالجة الخلية التي تقع في الموضع **A** الخارجي (الصف) وز **z** الداخلي (العمود) وبذلك أنت الآن تعالج الخلايا (**i**، **j**) وكلماهما يتغيران بتغير الدوران حتى تصل لنهاية الصدوف والأعمدة.

وهذا ما حدث في المثال 6.5، فالأسطر من 4 حتى 10، في بدايتها تم كتابة الدوران الخارجي بعدد الصدوف وهي ثلاثة، لأن المثال يريد خمس درجات لثلاث طلبة، أما الدوران الداخلي وهو خاص بالأعمدة (أي عدد الدرجات) وهم خمس درجات. إذن فالدوران الخارجي سيبدأ من الصفر وحتى الرقم 2، أما الداخلي فيبدأ من الصفر وحتى الرقم 4.

يبقى فقط شكل الطباعة التي نريدها، فهنا وضعت جملة طباعة في السطر رقم 6، يقوم بكتابه رقم الطالب وهو بالتأكيد علاقته مع الصدوف ويزيد الرقم كلما زاد رقم الصف ليصبح الناتج كما بالشكل 6.11.

General Output				
Configuration: Ex				
1 Student's marks:	70.0	80.0	68.0	78.0
2 Student's marks:	80.0	90.0	78.0	77.0
3 Student's marks:	98.0	87.0	78.0	65.0
Process completed.				

شكل 6.11: ناتج المثال 6.6

## 6.6 أمثلة وتدريبات عامة

## 1. استخدم الكلمات المناسبة لتعبئة الفراغات في الجمل التالية:

- أ- من تطبيقات المصفوفات الهامة على مستوى البيانات \_\_\_\_\_ و\_\_\_\_\_.
- ب- يوجد من المصفوفات في عالم البرمجة نوعان هما \_\_\_\_\_ و\_\_\_\_\_.
- ت- تقاطع الصد مع العمود في المصفوفات الثنائية الأبعاد يعطينا \_\_\_\_\_.
- ث- القيمة الافتراضية للنوع *String* هي \_\_\_\_\_ بينما للنوع *int* هي \_\_\_\_\_.
- ج- المصفوفة التي حجمها 101، يمتد رقمها المتسلسل من \_\_\_\_\_ إلى \_\_\_\_\_.
- ح- الخطأ الذي يظهر عندما يتتجاهل المبرمج إعطاء قيمة لحجم المصفوفة عند التعريف هو \_\_\_\_\_.
- خ- المتغير المعروف مسبقاً في لغة جافا ومن خلاله يتم معرفة حجمها هو \_\_\_\_\_.
- د- الخطأ *ArrayIndexOutOfBoundsException* يُصنّف على أنه من الأخطاء \_\_\_\_\_.

## 2. حدد صحة أو خطأ كل من العبارات التالية مع التعليل:

- ( ) أ- لا يسمح لحجم المصفوفة بالزيادة وقت تنفيذ البرنامج إلا في حالات استثنائية.
- ( ) ب- خلايا المصفوفة يتم تخزينها في الذاكرة في مواضع متباينة، إلا أنه يمكن الوصول لها.
- ( ) ت- نوع الرقم المتسلسل للمصفوفة (*index*) من الممكن أن يكون من أي نوع إلا (*string*).
- ( ) ث- الكلمة المحظوظة (*new*) التي تستخدم في تعريف المصفوفات تعتبر اختيارية.
- ( ) ج- عندما يتم إنشاء مصفوفة بعناصر معروفة مسبقاً يتم إعطاءها العناصر داخل أقواس {} ويفصل بين العناصر من خلال الفاصلة \_\_\_\_\_.
- ( ) ح- المصفوفة الواحدة لا يمكن أن تحتوي غير نوع واحد من البيانات في الوقت الواحد.
- ( ) خ- طباعة قيمة العنصر الثالث من عناصر المصفوفة X نكتب في جملة الطباعة [2]X.

## 3. حول الجمل التالية إلى جمل يفهمها مترجم لغة جافا

- أ- تعريف مصفوفة لتخزين 10 نجوم.
- ب- طباعة عناصر مصفوفة ID تحتفظ بأرقام الجلوس لطلبة الثانوية العامة.

- ت- طباعة العنصر الخامس من عناصر المصفوفة names.
- ث- إعطاء القيمة 109 للخلية رقم 7 من المصفوفة ID.
- ج- تعريف مصفوفة لتخزين خمسة مواعيد على مدار خمسة أيام.

4. حدد ثم صلح الأخطاء في الجمل البرمجية التالية:

أ- الجملة التالية لطباعة عناصر المصفوفة sal

```
int [ ] sal = new int [3];
for (int i=0; i<3; i++)
System.out.println("i: sal[i]");
```

ب- الجملة التالية لتخزين الرقم 10 كقيمة لكافة الخلية الفردية في المصفوفة odd ذات الـ 10 خلية

```
int [ ] odd = new int [10];
for (int i=0; i<=10; i++)
{
    if (i%2==0)
        odd[i]=10;
}
```

ت- الجملة التالية لاستقبال 10 أسماء من المستخدم وتخزينها في المصفوفة names

```
String [ ] names = new String[10];
for (int i = 0; i<10; i++)
{
    i = JOptionPane.showInputDialog("Enter names");
}
```

5. تتبع الجمل البرمجية التالية واكتب ناتجها

-أ-

```
int [ ] array = new int [10];
for (int k=0; k<10; k++)
{
    if (k%3==0)
        array[k]=3;
    else
        array[k] = 10;
}
for (int i=0; i<10; i++)
System.out.println("i: "+array[i]);
```

-ب-

```
float [ ][ ]marks = {{70 ,80 ,68 ,78 ,67} ,
{80 ,90 ,78 ,77 ,89} , {98 ,87 ,78 ,65 ,90}};
for (int i= 0; i< 3; i++)
{
    System.out.print("The "+(i+1)+"th Student's marks:");
    for (int j=0; j<5 ; j++)
        System.out.print(marks[i][j]+\t);
    System.out.print("|\n");
}
```

6. اكتب برنامج يقوم بإنشاء مصفوفة لتخزين أرقام صحيحة، على أن يتم تخزين فيها مضاعفات الرقم 2 من 0 إلى 20 ثم طباعتها.

7. اكتب برنامج يقوم بإنشاء مصفوفة لتخزين عشرة أرقام صحيحة يتم استقبالها من المستخدم، ثم يتم بعد ذلك جمع الأرقام الزوجية منها وطباعة الناتج.
8. اكتب برنامج لحفظ مضاعفات الرقم 4 من 0 إلى 100 ثم يقوم بطباعة مجموعها ومتوسطها الحسابي.
9. اكتب برنامج يستقبل من المستخدم ثلاثة أرقام، فإن كان الرقم المدخل من مضاعفات الرقم 3 يتم تخزينه في المصفوفة وإلا يتم تخزين الرقم 0، مع جمع كافة الأعداد وطباعة مع المجموعة.
10. مستقدياً من السؤال رقم 10، اطبع عدد الأرقام التي أدخلها المستخدم وهي من المضاعفات.

### 6.7 تمارين ذاتية الحل

1. حول الجمل التالية إلى جمل برمجية متوافقة مع قواعد لغة جافا
- مصفوفة لتخزين الأرقام الفردية.
  - إعطاء القيمة 100 للخلية الخامسة في المصفوفة `array`.
  - طباعة قيمة الخلية السادسة في المصفوفة `even`.
  - حساب متوسط الأعداد للمصفوفة `numbers`.
  - حساب مضروب الأعداد للمصفوفة `values`.
2. اكتب برنامج يقوم باستقبال عدد من الأرقام الصحيحة بين 0 و100 (إن كان الرقم المدخل خارج المدى يرفضه البرنامج ولا يحسب) وتخزينها في مصفوفة على أن يمنح المستخدم فقط 10 محاولات. ثم قم بطباعة الأرقام ومجموعها.
3. مستعيناً بالسؤال رقم 2، اجعل البرنامج يطبع أصغر وأكبر رقم في الأرقام المدخلة.
4. اكتب برنامج يستقبل عدد من الأرقام ويخزن فقط أول خمسة أرقام زوجية يتم إدخالها من المستخدم.
5. اكتب برنامج يقرأ من المستخدم 20 رقمًا صحيحاً، ثم يفحص هذه المصفوفة ويختار الأرقام التي تقبل القسمة على الرقم 7 ليخزنها في مصفوفة أخرى، مع طباعة أرقام المصفوفة الأخرى.
6. إحدى الكليات التعليمية تمنح طلابها جوائز مادية مقابل تفاعلهم مع برامج التعليم الإلكتروني بالكلية، باعتبار أن الطالب يحصل على 5 نقاط على كل برنامج يشارك به، اكتب برنامج يقرأ من المستخدم

عدد المشاركات لعشرة طلبة، على أن يطبع البرنامج في النهاية جدولًا يضم عمود به عدد المشاركات وأخر به قيمة الجائزة التي حصل عليها الطالب.

7. اكتب برنامج يقرأ معدلات عشرة طلاب، ويخرجها في مصفوفة، ثم يقرأ أسماءهم بالترتيب ذاته ويخرجهم في مصفوفة أخرى، ثم قم بطباعة أسماء الطلبة الذين يزيد معدلهم عن 60%.
8. اكتب برنامجًا لطباعة جدول ضرب السبعة في مصفوفة ذات بعدين.
9. مستخدما الشكل التالي:

2	7	76	4
1	6	5	5
2	5	43	3
34	5	9	7

أ- اطبع أرقام الصف الثالث

ب- اطبع الرقم الثاني والرابع في العمود الثاني

ت- اجمع عناصر الصف الصفرى

ث- ما هو أكبر عدد في الصف رقم 5

ج- ما هي قيمة الخلية (2, 3) و (3, 3)

10. ما ناتج القطعة البرمجية التالية:

```
int x = 30;
int[] numbers = new int[x];
x = 60;
System.out.println("x is " + x);

System.out.println("The size of numbers is " + numbers.length);
```

11. حدد أي من الجمل البرمجية التالية صحيحة:

```
int i = new int(30);
double d[] = new double[30];
```

تأليف، أ. محمود رفيق الفرا

باب السادس، المصفوفات

```
char[] r = new char[1..30];
int i[] = {3, 4, 3, 2};
float f[] = {2.3, 4.5, 6.6};

char[] c = new char();
```

**12.** بالشرح والأمثلة، ما الذي يحصل عندما يحاول البرنامج التفاعل مع عنوان خلية غير موجود؟

**اجابات التدريبات العامة :**

**1.** . (أ) البحث، الترتيب (ب) أحادية الأبعاد، متعددة الأبعاد (ت) خلية (ث) null، صفر (ج) صفر، 100 (ح) خطأ برمجي (خ) length (د) الاستثنائية

**2.** حدد صحة أو خطأ كلا من العبارات التالية مع التعليل:

أ- (خاطئة) لا يسمح ببيانات

ب- (خاطئة) تخزن في مواضع متتالية ومتواصلة

ت- (خاطئة) لا يمكن إلا من النوع *int*

ث- . (صحيحة) وقد تعتبر في بعض الحالات إجبارية.

ج- (صحيحة)

ح- (صحيحة)

خ- (صحيحة)

.3

أ- `char [ ] stars = new char[10];`

ب- `String [ ] ID = new String [5];`  
`for (int i =0; i<5; i++)`  
`System.out.println(ID[i]);`

ت- `System.out.println(names[4]);`

ث- `ID[7] =109;`

ج- `String [][] events = String[5][5];`

.4

أ- نوع الخطأ: منطقي، الخطأ : كتابة متغير المصفوفة داخل علامات التصيص

التصحيح:

```
int [ ] sal = new int [3];
for (int i=0; i<3; i++)
System.out.println("i: " + sal[i]);
```

ب- نوع الخطأ 1 : استثنائي، نوع الخطأ 2: منطقي

نوع الخطأ 1: الشرط المنطقي يصل للخانة رقم 11 لأنه يستخدم `=` بينما الخلايا هم 10

نوع الخطأ 2: الشرط `(i%2==0)` سيجعله يخزن الرقم 10 في الخانات الزوجية.

التصحيح:

```
int [ ] odd = new int [10];
for (int i=0; i<10; i++)
{
    if (i%2!=0)
        odd[i]=10;
}
```

ت- نوع الخطأ: برمجي، الخطأ: البرنامج يحاول تخزين الأسماء في الرقم المتسلسل أ

التصحيح:

```
String [ ] names = new String[10];
for (int i = 0; i<10; i++)
{
    names[i] = JOptionPane.showInputDialog("Enter names"); }
```

.5

أ- يخزن الرقم 3 في الخانات التي تقبل القسمة على 3، والرقم 10 في الخانات الباقيه.

i: 3

i: 10

i: 10

i: 3

i: 10

i: 10

i: 3

i: 10

i: 10

i: 3

ب- يخزن ثم يطبع، خمس درجات لثلاثة طلبة، بشكل مصفوفة مستطيلة الشكل

The 1'th Student's marks: 70.0 80.0 68.0 78.0 67.0 ||

The 2'th Student's marks: 80.0 90.0 78.0 77.0 89.0 ||

The 3'th Student's marks: 98.0 87.0 78.0 65.0 90.0 ||

```
int num [] = new int [21];
for (int i=0; i<=20; i++)
{
    num[i]= i*2;
    System.out.println("num["+i+"]": "+num[i]);
}
```

شكل 6.12: الجمل البرمجية اللازمة لحل تمرين 6

```

public static void main(String[] args) {

    String input;
    int sum =0;
    int num [] = new int [10];
    for (int i=0; i<10; i++)
    {
        input = JOptionPane.showInputDialog("Enter another number");
        num[i]= Integer.parseInt(input);

        if (num[i]%2 == 0)
            sum = sum+num[i];
    }

    JOptionPane.showMessageDialog(null,"the sum of even numbers "+sum);
}

```

شكل 6.13: الجمل البرمجية في الدالة الرئيسية اللازمة لحل تمرين 7

```

public static void main(String[] args) {
    int [] mul4 = new int[26];
    int sum =0;
    float avg=0;
    for (int i=0; i<=25; i++)
    {
        mul4[i]= i*4;
        sum = sum+mul4[i] ;
    }
    avg = sum/26;

    System.out.print("Sum: "+sum+" avg: "+avg);
}

```

شكل 6.14: الدالة الرئيسية لحل تمرين 8

```

public static void main(String[] args) {
    String input;
    int num;
    int sum=0;
    int [ ] array = new int [3];
    for (int k=0; k<3; k++)
    {
        input = JOptionPane.showInputDialog("Enter number");
        num = Integer.parseInt(input);
        if (num%3==0)
            array[k]=num;
        else
            array[k] = 0;
        sum = sum + num;
    }

    System.out.print("Sum: "+sum);
}

```

شكل 6.15: الدالة الرئيسية لحل تمرين 9

```

public static void main(String[] args) {
    String input;
    int num;
    int sum=0;
    int count =0;
    int [ ] array = new int [3];
    for (int k=0; k<3; k++)
    {
        input = JOptionPane.showInputDialog("Enter number");
        num = Integer.parseInt(input);
        if (num%3==0){
            array[k]=num;
            count++;
        }
        else
            array[k] = 0;
        sum = sum + num;
    }
    System.out.print("Sum: "+sum+" and counter: "+count);
}

```

شكل 6.16: الدالة الرئيسية لحل تمرين 10

انتهى الباب

مع كل خطوة للإنسان إما أن يشعر بالإنجاز  
فيتقدم وإما أن يشعر بالخوف فيترد. وفي  
هذا قالوا، السر في النجاح هو عدم  
الاستسلام للصعوبات بتاتاً.

## الباب السابع

### الدوال

(Methods)

يُتوقع من الدارس في نهاية هذا الباب أن:

- يشرح مفهوم الدوال.
- يدرك الحاجة لتطوير واستخدام الدوال في البرمجة.
- يُتقن إنشاء الدوال.
- يشرح طرق استدعاء الدوال المختلفة من حيث القيم الممرة والمرجعة.
- يُتقن تمرير مصفوفة للدالة.
- يُتقن استخدام دوال الصنف (static).
- يتعرف على الدوال الجاهزة في الصنف (Math).
- يميز بين المتغيرات المحلية وال العامة.
- يشرح مفهوم الاستدعاء الذاتي ويستخدمه.
- يفرق بين الاستدعاء الذاتي والتكرار.

يتوفر لهذا كتاب سرفح لعرض (PowerPoint) ومكتبة ملفات مرئية من إعداد مؤلف الكتاب؛ يمكنك الحصول عليها من خلال لموقع الأكاديمي للمؤلف [mfarra.cst.ps](http://mfarra.cst.ps)



## الباب السابع: الدوال

### 7.1 مقدمة

معظم التطبيقات البرمجية التي يتم التعامل معها وتطبيقها من خلال لغات البرمجة تكون أكثر تعقيداً من تلك التي ناقشناها في الأبواب الماضية، مما يزيد من تعقيدها ويُساهم وبالتالي في زيادة الحاجة إلى تعديل أسلوب البرمجة ليعتمد على مبدأ التقسيم المعروف باسم "فرق تسد" (*Divide and Conquer*) حيث يتم تقسيم "الكود" البرمجي إلى أجزاء يمكن إعادة استخدامها والاستفادة منها وهي طريقة أثبتت نجاعتها من التجارب المختلفة.

بهذا المبدأ يتم تقسيم المشاكل الكبيرة إلى أجزاء صغيرة تسمى (*Modules*) يسهل التعامل معها، ومن أصغرها ما يُعرف باسم الدوال أو الطرق أو نحوه (*Methods*، ومستخدمة في لغة جافا حيث أنها تعتبر الجزء الذي من خلاله يتم بناء الأصناف ثم المشاريع الكبيرة كما شاهدت ذلك في الشكل 3.7 من الباب الثالث. ولهذا فإننا في هذا الباب سنعمل على استعراض مفهوم الدوال في الفصل 7.2، ثم طريقة تعریفها وأنواعها في الفصل 7.3 قبل أن نتعرف على طرق وأنواع جمل الاستدعاء في الفصل 7.4، بينما في الفصل 7.5 نستعرض مفهوم دوال الصنف (*static Method*) والفائدة منها وأثرها على الأصناف التي تستخدم داخلها حيث نتعرف في الفصل 7.6 على أحد الأصناف المعرفة مسبقاً في لغة جافا وتشتمل على مجموعة كبيرة من الدوال الثابتة. وفي ختام هذا الباب نقف مع مفهوم في غاية الأهمية وهو الرؤية (*Scope*) والتفاعل مع المتغيرات خاصة في ظل اقسام البرنامج إلى أجزاء مختلفة وتتوفر فرص للتعارض الظاهري بين المتغيرات.

### 7.2 مفهوم الدوال وال الحاجة إليها

الدوال وسيلة مناسبة لإنجاز التطبيقات البرمجية بأسلوب أسهل وأسرع حيث تتيح لنا فرصة إعادة استخدام أجزاء سابقة تم تطويرها؛ لتوضح الصورة، افترض أننا نحتاج لطباعة حاصل ضرب الأرقام من 5 إلى 25 والأرقام من 100 إلى 1000 والأرقام من 10 إلى 90، لعلك تذهب لتنفيذ ذلك بالطريقة التالية:

```
int mull = 1;
for (int i = 5; i <= 25; i++)
mull = mull*i;
System.out.println("Multiple from 5 to 25 is " + mull);
```

```
int mull = 1;
for (int i = 100; i <= 1000; i++)
mull = mull*i;
System.out.println("Multiple from 100 to 1000 is " + mull);
```

```
int mull = 1;
for (int i = 10; i <= 90; i++)
mull = mull*i;
System.out.println("Multiple from 10 to 90 is " + mull);
```

بالتركيز في هذه المحاولات، لعلك انتبهت إلى أن إنجاز المطلوب تم بطريقة واحدة وختلفت فقط البداية والنهاية لكل تكرار، وعليه لماذا لا نفكّر في كتابة قطعة برمجية واحدة تسمح لنا بتمرير قيم مختلفة للبداية والنهاية في كل مرة؟ هذه هي الدوال وال الحاجة لها. وبذلك يمكن للمبرمج من خلال مفهوم الدوال أن يقسم المهام في برنامجه إلى أجزاء منفصلة تسمح باستقبال القيم المختلفة إن لزم وتخرج الناتج المتوقع ويتم استدعاءها وفقما احتاج لها.

والدوال التي سنستخدمها في تطوير برامجنا تتقسم إلى نوعين:

- دوال يتم تعريفها من خلال المبرمج وتعُرف بـ "دوال معرفة بواسطة المستخدم".
- دوال معرفة مسبقاً من خلال لغة Java ومنتها دالة (`print`) و (`println`) التي يقوم المستخدم فقط باستدعائهما لإنجاز مهمة معينة وهي الطباعة على سبيل المثال.

وتبرز الحاجة إلى الدوال في تنظيم البرنامج إلى قطع صغيرة وبسيطة، مما يساعد على اكتشاف الأخطاء بسهولة، كما أنها تساعد على تنظيم الجمل البرمجية الالزامية لإنجاز مهمة معينة في قطعة برمجية يمكن استدعاؤها كلما احتجنا لها بدلاً من إعادة كتابتها مرات عديدة مما سيكون مكلفاً جداً في الوقت والمجهود، مع كبر حجم البرنامج بشكل يؤدي إلى التشتبّت وعدم التركيز.

### 7.3 تعريف وإنشاء الدوال

الدوال منها مثل أي مكون من مكونات لغة Java التي تحتاج إلى تعريفها أو إنشائها، منها في ذلك مثل والمصفوفات والمتغيرات. فالدوال تحتاج مثلاً إلى تعريف أركانها الأساسية وهي:

- اسم الدالة وهو مضبوط بضوابط الأسماء التعريفية.
- نوع القيمة التي تعدها الدالة بعد انتهائها من العمل وقد لا تعهد فيوضع كلمة (`void`).
- نوع القيم التي تستقبلها.

3. الجمل البرمجية التي يؤدي استخدامها إلى تنفيذ المهمة المطلوبة.
- وبناءً على ذلك فالدوال من حيث التعريف تتكون من مكونين رئيسيين هما:
2. التوقيع (*Signature*): وهو المفتاح الرئيسي الذي يتم من خلاله تمييز بين الدوال المختلفة، كما بالشكل 7.1، حيث يضم التوقيع من اليسار إلى اليمين كلاً من :
- ✓ نوع القيمة التي تعدها الدالة بعد تنفيذ المهمة الخاصة بها، فمثلاً إن كانت الدالة تجمع رقمين صحيحين فإنها تعيد قيمة صحيحة، وإن كانت تبحث عن اسم موظف موجود أم لا فإنها تعيد قيمة منطقية وهكذا، وقد لا تعيد الدالة قيمة فنكتب في هذا الموضع كلمة (*void*) وهي من الكلمات المحجوزة.
  - ✓ اسم الدالة وهو اسم تُعرف من خلاله الدالة وتنسقى من خلاله، وينطبق على تسمية الدوال ما ينطبق على الأسماء التعريفية من قواعد، ولا يمكن أن يتم تعريف دالة دون ذكر اسمها.
  - ✓ أنواع القيم الممرة للدالة وهي القيم التي تحتاجها الدالة للعمل -إن احتجت-، فمثلاً إن كانت دالة لجمع رقمين، فإن لم يتم تمرير قيمتين للدالة فلن تعمل، كذلك إن كانت الدالة تبحث في مصفوفة عن بيانات موظف يُرسل لها، فإن لم يُرسل لها مفتاح البحث فلن تعمل أيضاً. وعدد القيم الممرة للدالة غير محدد بعد، قد لا تحتاج الدالة لأي قيم وفي هذه الحالة نترك الأقواس فارغة، وقد تحتاج إلى عدد كبيرٍ من القيم فلا بد أن تذكر جميعاً مع التفريق بينهم بالفواصل (،).
  - ✓ الجسم (*Body*): وهو الجزء الذي يضم الجمل البرمجية التي تعمل على تنفيذ مهمة معينة على مجموعة من البيانات سواء الممرة أو غيرها، كما أنه قد يضم جملة (*return*) إن كانت الدالة ستُعيد ناتجاً حيث أن القيمة (*return\_value*) يوضع بدلاً منها متغير أو قيمة من ذات النوع المعروf في التوقيع. الشكل 7.2 و 7.3 يُظهر تعريف الدالة بشقيها.

شكل 7.1: تركيبة التوقيع (*Signature*) الخاص بالدالة

```
Access_Modifiers returned_type Method_name (Type par1, Type par2, ... )
{
    return return_value;
}
```

الجمل و الأوامر البرمجية

شكل 7.3: تركيبة تعريف الدالة كاملاً

والأنثمة على التعريف العام للدوال دون كتابة تفاصيل الجمل البرمجية داخل جسم الدالة كما يلي:

أ- دالة تستقبل رقم الموظف من نوع **String** وتطبع بيانات معينة دون أن تُعيد قيمة:

```
public void print(String emp_id) {
    // الجمل البرمجية
}
```

ب- دالة تستقبل رقم من نوع **int** وتحث عن بيانات معينة وتُعيد نتيجة البحث قيمة من نوع **boolean**

```
public boolean search(int serial){
    boolean found;
    // ....
    return found;
}
```

ت- دالة لا يلزم عملها استقبال قيم وتحسب المعدل لقيم لديها وتُعيد الناتج قيمة من نوع **float**

```
public float avg () {
    float avg;
    //....
    return avg;
}
```

ث- دالة لا يلزم عملها استقبال قيم وتقوم بترتيب بيانات مصفوفة متوفرة لديها ما ولا تُعيد شيء

```
public void sort () {
//...
}
```

وبالعودة إلى الفرضية التي افترضناها في الفصل 7.2 لطباعة حاصل ضرب مجموعة أرقام يمكننا الآن كتابة الدالة التالية لثلك المشكلة وبأقل عدد من الجمل البرمجية وبمرونة عالية:

```
1 public static int mull(int x, int y) {
2     int result = 1;
3     for (int i = x; i <= y; i++)
4         result = result * i;
5     return result;
6 }
```

بينما يمكن استدعاء الدالة لمعالجة أي قيم نحتاجها دون إعادة كتابة الجمل البرمجية السابقة في كل مرة:

```
1 public static void main(String[] args) {
2 System.out.println("Mull from 5 to 25 is " + mull(5, 25));
3 System.out.println("Mull from 100 to 1000 is " + mull(100, 1000));
4 System.out.println("Mull from 10 to 90 is " + mull(10, 90));
5 }
```

ج - ودراسة كافة الأمثلة السابقة، نجد أن الدوال تنقسم من حيث استقبال القيم (*Parameters*) وإعادة النتائج إلى أربع حالات هي:

1. دوال لا تستقبل قيم ولا تعيد: ومثالها الحالة رقم (ث) في الأمثلة السابقة.

2. دوال لا تستقبل قيم ولكنها تعيد قيم ومثالها الحالة رقم (ت) في الأمثلة السابقة

3. دوال تستقبل قيم ولا تعيد: ومثالها الحالة رقم (أ) في الأمثلة السابقة.

4. دوال تستقبل قيم وتعيد قيم ومثالها الحالة رقم (ب) في الأمثلة السابقة.

ماذا عن مثال حاصل الضرب الأخير، من أي الأنواع الأربعية في اعتقادك؟

### مثال توضيحي 7.1 :

اكتتب دالة تستقبل قيمتين صحيحتين فتجمعهم وتعيد ناتج الجمع.

```
1 public int Sumint(int a, int b)
2 {
```

```

3     int sum = a+b;
4     return sum;
5 }
```

هذا المثال يطلب كتابة دالة تستقبل رقمين صحيحين ثم تعيد قيمة الناتج، بالتأكيد فإن جمع الصحيح ينتج عنه صحيح وبالتالي فإن الناتج سيكون نوعه صحيحا أيضاً، لذلك فإن توقيع الدالة تشتمل بداية على محدد الوصول (*public*) وهذا لا يؤثر في هذه المرحلة وإنما سيؤثر نوع المحدد عندما يكون هناك تفاعل بين كائنات أكثر من صنف كما سيظهر معنا في الباب التاسع. بعد المحدد تم كتابة نوع القيمة المتوقع أن تعدها الدالة وهو النوع الصحيح، ثم اسم الدالة وهو اسم متوافق مع قواعد الاسم التعريفى التي ذكرت في الباب الثالث ثم بين أقواس *return* القيم الممرة تجد أنه طلب قيمتين من نوع (*int*) الأولى رمز لها بالاسم (*a*) والثانية (*b*)، أما في جسم الدالة فنم

**الجملة *return*** ليس شرطاً أن يتم كتابتها في نهاية الدالة، بل قد تكتب في أي موضع داخل الدالة حسب الحاجة. كما أنه من المهم أن الدالة إذا كانت تعيد قيمة فلابد من استخدام الجملة *return* لإعادة قيمة نوعها يتناسب مع ما هو مطلوب في توقيع الدالة.



قد تحتوي الدالة على أكثر من جملة *return* إلا أن ما ينفذ منها بالفعل جملة واحدة فقط، لأن المترجم عندما يرى الجملة *return* داخل الدالة ينفذها ثم يخرج من الدالة ولا يحمل مسیرته.



لغات برمجة أخرى تشير إلى الدوال على أنها (إجراءات أو طرق)، حيث الدالة التي تعيد قيمة تسمى طرق (*Function*) والتي لا تعيد تسمى إجراءات (*procedure*).



تعريف متغير من النوع الصحيح (*int*) ليتم فيه وضع قيمة جمع المتغيرين (القيمتين) *a* و*b*، وأن الدالة تُعيد قيمة صحيحة فقد تم كتابة الجملة (*return*) وبجانبها المتغير الذي يحتوي على قيمة الجمع.

### مثال توضيحي 7.2:

اكتب دالة تستقبل قيمتين عشربيتين موجبتين ثم تعيد أيهما أكبر.

```

1 public float max(float x, float y)
2 {
3     if (x > y)
4         return x;
5     else if (y > x)
```

```

6     return y;
7 else
8     return -1.0f; // هذا يعني أنهما متساوين
9 }

```

كما تلاحظ في السطرين 4 و 6، فقد تم استخدام جملتي (*return*) إلا أنه لن يتم تنفيذ إلا جملة واحدة منهم، فإن كان العدد (x) أكبر من العدد (y)، فإن الدالة تُعيد قيمة المتغير (x) ولا تُتم تنفيذ جمل الدالة، وإن كان المتغير y أكبر من المتغير x فإن الدالة تُعيد قيمة المتغير y ولا يرى المترجم جملة (*return*) الأولى. إذا لم يكن أحدهما أكبر من الآخر يتحقق الشرط الأخير (السطر 7) وتُعيد الدالة القيمة (-1.0f) كإشارة على أنهما متساوين.

### مثال توضيحي 7.3:

اكتُب دالة تستقبل قيمة صحيحة فإن كانت من مضاعفات العدد 2 يُعاد *true* وإلا يُعاد *false*.

```

1 public boolean mul2(int a){
2     if(a%2==0)
3         return true;
4     else
5         return false;
6 }

```

العملية % تُعيد باقي قسمة رقمين، وبالتالي فإنها تُستخدم في كافة التطبيقات ذات العلاقة بالمضاعفات مثل فحص المضروب، فحص الزوجي والفردي وغيرها.



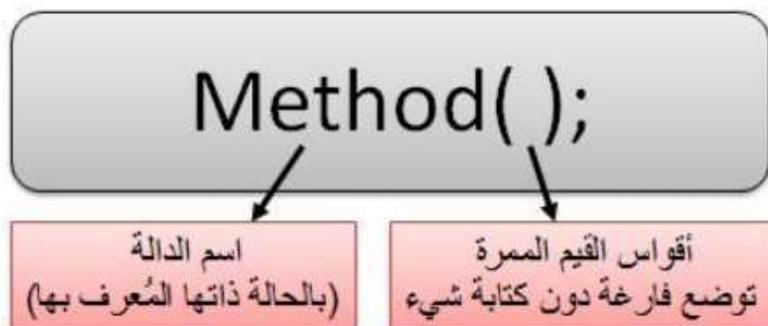
هذه الدوال عند كتابتها، لا يكون لها أي تأثير دون أن يؤثر لها بالعمل، والإذن لها بالعمل يتم من خلال استدعائهما بصورة مباشرة أو غير مباشرة من الدالة الرئيسية (*main*) كما، وهذا ما سنتناشه في الفصل القادم.

### 7.4 استدعاء الدوال

استدعاء الدوال هو ما يُعرف باسم (*Invoke*) أو (*Call*) وهي جملة برمجة نطلب من خلالها من الدالة أن تعمل ثم نزورنا بالنتائج. وكما وضح من دراسة حالات تعريف الدوال في الفصل السابق (الحالات أ إلى ث) نجد أن استدعاء الدوال يتأثر بهذه الحالات على النحو التالي:

#### 7.4.1 استدعاء دوال لا تستقبل قيم ولا تعيد نتائج:

وهذا النوع من الدوال يتم استدعاءه فقط من خلال كتابة اسم الدالة وبعده أقواس الفيم فارغة ( ) ومن ثم الفاصلة المنقطة كما تلاحظ في الشكل 7.3، ويوضح ذلك من خلال المثال التوضيحي 7.4.



شكل 7.4: طريقة استدعاء الدالة التي لا تستقبل ولا تعيد قيم

#### مثال توضيحي 7.4 :

اكتب برنامج يستقبل رقمين من المستخدم ويخزنهم خارج أي من الدوال، ثم اكتب دالة عند استدعائهما تقوم بطباعة القيمة الأكبر.

```

1 import javax.swing.JOptionPane;
2 public class Example7_4 {
3     public static void PrintMax(){
4         if (x>y)
5             JOptionPane.showMessageDialog(null,"X>y");
6         else if (y>x)
7             JOptionPane.showMessageDialog(null,"Y>X");
8         else
9             JOptionPane.showMessageDialog(null,"X=Y");
10    }
11    // متغيرات عامة يمكن التفاعل معها من أي دالة داخل الصنف، حيث تم تعريفهما خارج أي دالة
12    public static int x;
13    public static int y;
14    // الدالة الأساسية وفيها يتم استدعاء الدوال من أجل إظهار النتائج
15    public static void main(String[ ] args) {
16        String input = JOptionPane.showInputDialog("Enter X");

```

```

17 x = Integer.parseInt(input);
18 input = JOptionPane.showInputDialog("Enter Y");
19 y = Integer.parseInt(input);
20 PrintMax();
21 }
22 }

```



شكل 7.5: نتیجة مثل 7.4

في هذا المثال كما تلاحظ، قمنا بتعريف متغيرين من النوع الصحيح خارج الدالة الأساسية في السطرين 12، 13، وهذا كذلك داخل الصنف الأساسي، وبالتالي يصبح من حق أي دالة داخل الصنف الأساسي التفاعل معهم، وهذين المتغيرين تم إعطاءهم قيم من خلال جمل إدخال داخل الدالة الأساسية ثم تم استدعاء الدالة (PrintMax) في السطر 20 وهو استدعاء تم داخل الدالة الأساسية. ونتيجة لهذا الاستدعاء تقوم الدالة (PrintMax) مباشرة بالعمل على مقارنة القيمتين وتنفيذ الإجراءات التي بداخلها وهي إجراءات جمل اتخاذ قرار لطبع الناتج.

الملاحظ أننا قمنا بكتابة الكلمة (static) قبل الدالة (PrintMax) وكذلك قبل المتغيرين، فلماذا؟! هذا يحدث لأن الدالة الأساسية من النوع (static) وبالتالي فهذا النوع من الدوال لا يقبل التعامل مع أي نوع آخر من الدوال أو المتغيرات ليس من جنس نوعه، وبالتالي عندما نريد من الدالة الأساسية التعامل مع أي دالة أو متغير خارجها فلابد أن يكون من النوع (static)، أما عن مفهوم هذه الكلمة فسوف نتعرف عليها بوضوح خلال الفصل 7.5.

**عند عدم استدعاء الدوال داخل الدالة الأساسية فإنها تبقى كامنة بلا تأثير ولا نشعر بها.**



**خطأ برمجي يظهر عند استدعاء دالة أو متغير ليسا (static) داخل الدالة الأساسية.**



#### 7.4.2 استدعاء دوال تستقبل قيم ولا تعيد نتائج:

وهذا النوع من الدوال يحتاج لاستدعائه أن ترسل للدالة قيمة من النوع الذي تطلبه وبالترتيب المطلوب، وترسل القيم بين الأقواس دون كتابة النوع، فإن كانت الدالة تطلب قيمة من نوع (int) ثم (float) فإننا عند الطلب ترسل لها مثلاً القيمة 5 ثم 3.9 دون أن نضع المزيد من المعلومات، كما بالشكل 7.5.

هذه القيم تُرسل للدالة ثم يتم استبدال المتغيرات في توقيع الدالة بالقيم المرسلة وتستكمل الجمل البرمجية حسب الموجود. (هذه الطريقة سيتم توضيحها في الشكل 7.9 في الصفحة 200)



شكل 7.6: طريقة استدعاء الدالة التي تستقبل قيمة و لا تعيد نتائج

#### مثال توضيحي 7.5 :

اكتب دالة تستقبل رقمين من المستدعى وتطبع (*The first is multiple*) إن كان الأول من مضاعفات الثاني أو يطبع (*Not Multiple*) إن لم يكن كذلك، ثم ادمج هذه الدالة داخل برنامج وقم باستدعائهما.

```

1 import javax.swing.JOptionPane;
2 public class Example7_5 {
3     public static void multiOf2(int x, int y){
4         if (x%y==0)
    
```

```

5     JOptionPane.showMessageDialog(null,"The first is multiple");
6 else
7     JOptionPane.showMessageDialog(null,"Not Multiple");
8 }
9 public static void main(String[ ] args) {
10 String input = JOptionPane.showInputDialog("enter number");
11 int x = Integer.parseInt(input);
12 input = JOptionPane.showInputDialog("enter number");
13 int y = Integer.parseInt(input);
14 multiOf2(x,y);
15 } }
```

في هذا المثال، كما تلاحظ في السطر 14 فإننا قمنا باستدعاء الدالة `multiOf2` من خلال تمرير متغيرين من النوع الصحيح كما تطلب الدالة، وناتج هذه الدالة هو فقط طباعة هل القيمة الأولى من مضاعفات الثانية أم لا. وطباعة الجمل لا تعتبر إعادة ناتج وبالتالي هذه الدالة ليست دالة تعيد ناتج بل فقط تستقبل قيم.

**خطأ برمجي** يحدث عندما نهمل المحافظة على ترتيب القيم الممررة عند اختلاف الأنواع.



**خطأ منطقي** قد يحدث عندما نهمل المحافظة على ترتيب القيم الممررة عند تماثل الأنواع.



**جمل الطباعة** لا تعتبر قيمة، وبالتالي فإن الدوال التي يكون ناتجها فقط طباعة فلا تعيد ناتج



القاعدة العامة لطريقة تمرير القيم عند استدعاء الدوال موضحة في الفصل 7.5.

#### 7.4.3 استدعاء دوال لا تستقبل قيمة ولكنها تعيد نتائج:

وهذا النوع من الدوال تحتاج لاستدعائه أن نقوم بمساواة الاستدعاء بمتغير من النوع ذاته الذي تعيده الدالة، دون أن نعطي قيمًا للاستدعاء، فقط تحتاج كتابة اسم الدالة ومعه أقواس فارغة كما بالشكل 7.7.

## Type x =Method( );

هذا النوع مماثل للنوع الذي  
تعيده الدالة التي نستدعيها

اسم الدالة  
(بالحالة ذاتها المعروفة بها)

شكل 7.7: طريقة استدعاء الدالة التي لا تستقبل قيمة ولكنها تعيد نتائج

### مثال توضيحي 7.6 :

اكتب دالة تعيد أكبر عناصر مصفوفة معرفة بحالة رؤية عامة على مستوى الصنف الرئيسي، ثم قم بدمج هذه الدالة داخل برنامج وقم باستدعائهما من الدالة الأساسية.

```

1 public class Example9_6 {
2     public static int MaxArray(){
3         int max = num[0];
4         for (int i=0; i<num.length; i++)
5         {
6             if (num[i]>max)
7                 max = num[i];
8         }
9         return max;
10    }
11    static int [ ] num = {34·87·6·98·54·6·2};
12    public static void main(String[ ] args) {
13        int maximum = MaxArray();
14        System.out.println("Max is "+maximum);
15    }
16 }
```

نحتاج في هذا المثال إلى لفت الانتباه لنقطتين، هما:

✓ أولاً: المصفوفة التي قمنا بتعريفها بحالة رؤية عامة، احتاجت منا مثلاً مثل الدوال والمتغيرات إلى جعلها (*static*) لنتمكن من التعامل معها في الدوال الأخرى والتي هي من نوع (*static*).

✓ ثانياً: طريقة استدعاء الدالة في السطر رقم 13، تجد أن الدالة (*MaxArray*) تعيد قيمة من نوع *int* وبالتالي لا بد عند استدعاء الدالة أن نساويها بمتغير من النوع الصحيح كما تشاهد، حيث يتم تخزين القيمة التي تعدها الدالة في هذا المتغير، وهذا ما يحدث في المثال حيث أن الدالة ستعيد الرقم 98 ثم تخزنه في المتغير (*maximum*) وبعد ذلك تطبعه في السطر 14.

#### 7.4.4 استدعاء دوال تستقبل قيم وتعيد نتائج:

عند استدعاء هذا النوع من الدوال، نحتاج إلى الجمع بين النوعين 7.4.3 و 7.4.2 حيث ترسل قيم الدالة مكافأة لأنواع التي تطلبها كما نساوي الاستدعاء بمتغير من النوع ذاته الذي تعده الدالة، ويظهر ذلك في الشكل 7.8.

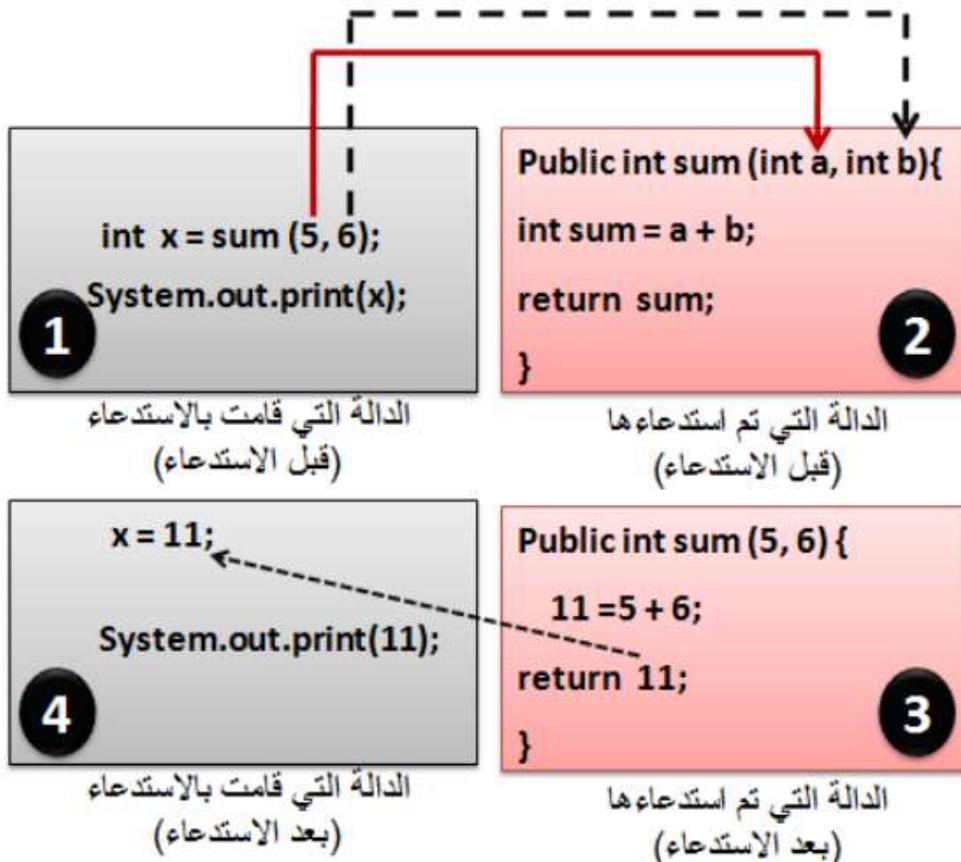


شكل 7.8: طريقة استدعاء الدالة التي تستقبل قيمها وتعيد نتائج

وهذا النوع من الاستدعاء، عندما ترسل القيم من خلاله إلى الدالة المستدعاة، يتم بعد ذلك وضع هذه القيم بدلاً من المتغيرات على الترتيب، ثم تتم العمليات كاملة داخل الدالة حتى النهاية وعندها يتم إعادة الناتج من استدعاء الدالة ليخزن في المتغير الذي تم إعداده لتخزين القيمة المرجعة. هذا المفهوم والاستراتيجية موضحة في الشكل 7.9. كما أن المثال 7.7 يوضح كيفية استدعاء هذا النوع من الدوال، من خلال استعراض دالة تستقبل رقمين وتعيد أيهما أكبر.

**خطا برمجي** يحدث عند إرجاع قيمة يتعارض نوعها مع النوع المعرف في توقيع الدالة.





شكل 7.9: كيفية التعويض بالقيم الممرة بدلاً من المتغيرات داخل الدالة التي يتم استدعاءها

عند استخدام الأمر `return` في الدالة التي تعيد، يعود التحكم لمنطقة الاستدعاء فقط.



### مثال توضيحي 7.7:

اكتب دالة تستقبل رقمين صحيحين وتعيد الأكبر. ثم قم بدمج هذه الدالة في برنامج لاستدعانها وطباعة ناتج الاستدعاء، على أن يتم اعتبار الرقم 1 هو مؤشراً على تساوي القيمتين.

في هذا المثال سنقوم بإنشاء الدالة (`maxnum`) لاستقبال رقمين وتعيد الأكبر، وهذا الرقم سيتم تخزينه في متغير (`ret`) في السطر 13، لأن الدالة عندما تعيد قيمة، يتم التعويض عن استدعاء الدالة كاملاً بالقيمة المرجعة لتصبح جملة الاستدعاء بعد انتهاء الاستدعاء كما يلي (`int ret = 7`) فالناتج يخزن في المتغير وهذا هو الموضح سابقاً.

```

1 import javax.swing.JOptionPane;
2 public class Example7_7 {
3     public static int maxnum(int a, int b)
4     {
5         if(a>b)
6             return a;
7         else if(b>a)
8             return b;
9         else
10            return -1;
11    }
12 public static void main(String[ ] args) {
13     int ret = maxnum(5,7);
14     JOptionPane.showMessageDialog(null,"Max: "+ret);
15 }
16 }
```

في لغة جافا يتم التمييز لفظياً بين المتغيرات التي يتم تتعريفها في توقيع الدالة بأنها **Arguments**.



### 7.5 تمرير المصفوفات للدوال

لقد أصبح واضحًا فيما سبق من هذا الباب أنَّ الدوال يمكنها استقبال قيم من أي نوع سواء كان أساسياً أو مشتقاً، وفي هذا الفصل نتعرف على مهارة هامة جدًا في البرمجة وهي كيفية جعل الدوال تستقبل مصفوفة، وكيف يمكننا إرسال مصفوفة دالة ما لمعالجتها.

لكي تعبر الدالة أنها تستقبل مصفوفة فإنَّ هذا يتم بسهولة باللغة، فبدلاً من كتابة نوع القيمة المطلوبة والاسم التعريفي الظاهري لها هكذا (*int a*) فيتم فقط وضع الأقواس التي تعبر عن المصفوفات ليصبح الأمر هكذا (*int [ ] a*) بذلك تصبح الدالة تستقبل مصفوفة من النوع الصحيح.

بينما عند استدعاء دالة تحتاج مثلاً لاستقبال مصفوفة من نوع `float`، فيتم تعريف مصفوفة من النوع العشري ولكن `(numf)` ثم يتم تمرير اسم المصفوفة للدالة كما يلي `method (numf)`، والمثال 7.8 يزيدوضوح.

### مثال توضيحي 7.8:

اكتب دالة تستقبل مصفوفة عشرية، وتعيد أصغر عناصرها، على أن تقوم بدمجها في برنامج يستدعىها ثم يطبع القيمة الأصغر.

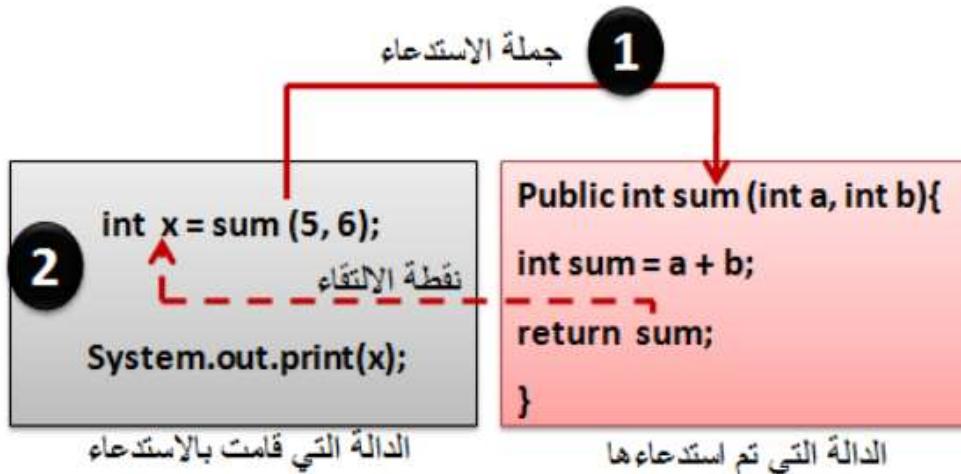
```

1 public class Example9_8 {
2     public static float minarray(float [ ] a)
3     {
4         float min = a[0];
5         for (float b : a)
6             if (b < min)
7                 min = b;
8         return min;
9     }
10    public static void main(String[ ] args) {
11        float [ ] arrayf = {9.7f, 5.7f, 4.3f, 100.7f, 6.7f, 1.1f, 77.8f};
12        float minf = minarray(arrayf);
13        System.out.println("Min: "+ minf);
14    }
15 }
```

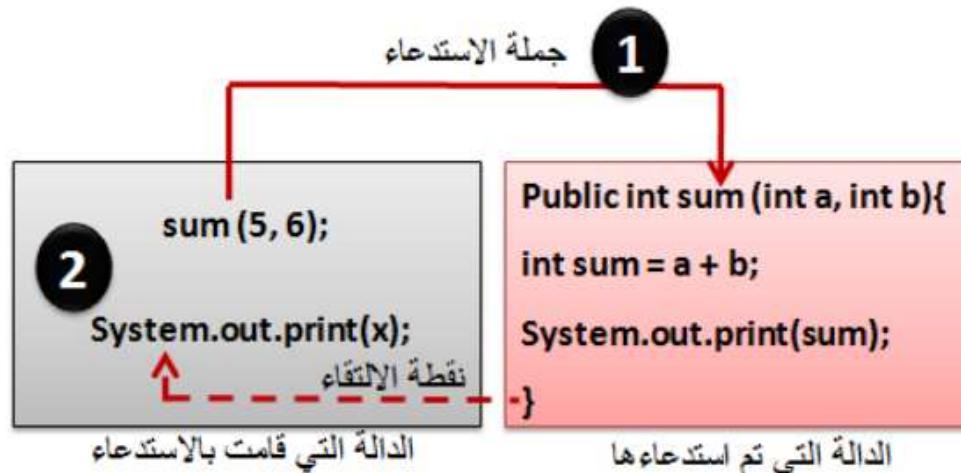
في السطر 11 من هذا المثال، تم إنشاء مصفوفة من النوع العشري باسم `arrayf`، (لاحظ أننا نضع الحرف `f` بجوار كل رقم وإلا سينتج لنا خطأ برمجي)، هذه المصفوفة أرسلناها للدالة `minarray` كقيمة ممرة في السطر 12، حيث أن الدالة تتطلب هذا النوع من البيانات كما تشاهد في السطر 2.

عند استدعاء الدالة ووصول القيمة الممرة وهي المصفوفة في هذه الحالة، يتم التعامل مع هذه المصفوفة في صورة المصفوفة الظاهرية `a` ويتم الفحص للوصول لأصغر قيمة بها من خلال استخدام جملة التكرار `for`

المحسنة. وفي نهاية التكرار نحصل على أصغر قيمة وتوضع في المتغير *min* والذي يتم إعادةه للمستدعي من خلال جملة (*return*) في السطر 8، وهذه القيمة عند إعادتها للمستدعي تخزن في المتغير *min* في السطر 12. ويتبين من الأمثلة السابقة أن خط سير العملية يتغير عند الاستدعاء لينتقل للدالة المطلوبة، ثم بعد الانتهاء من الاستدعاء تعود نقطة سير العملية إلى جملة الاستدعاء إن كانت الدالة تعيد قيمة، أو تعود عند الخطوة التالية لنقطة الاستدعاء إن كانت الدالة لا تعيد قيمة، في الشكل 7.10 و 7.11 تتضح هذه الفكرة الهامة.



شكل 7.10: توضيح كيفية عودة خط سير العمليات بعد انتهاء الاستدعاء (في حالة كانت الدالة تعيد قيمة)



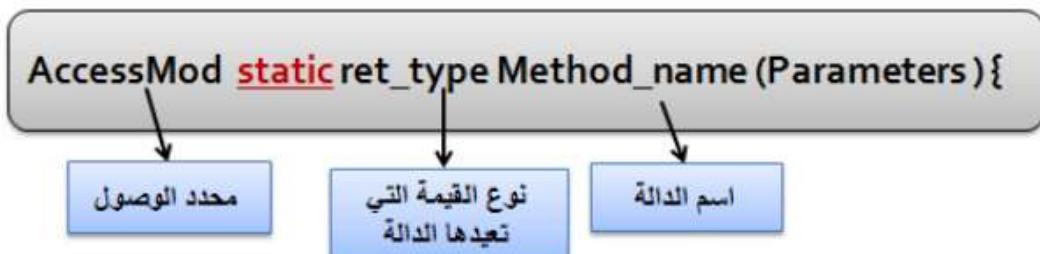
شكل 7.11: توضيح كيفية عودة خط سير العمليات بعد انتهاء الاستدعاء (في حالة كانت الدالة لا تعيد قيمة)

### 7.6 مفهوم الخاصية static للدوال والمتغيرات

أصبح من الواضح لدينا أن لغة جافا تعتمد على أسلوب البرمجة شيئاً فشيئاً التوجه، أي التي تعتمد على تقسيم العمل إلى أصناف بداخلها دوال كما أوضحنا في الباب الثالث، وهذه الأصناف يتم التعامل معها من خلال اشتقاق كائنات منها نطبق عليها الدوال المختلفة وهذا ما سنستعرضه بالتفصيل في الباب التاسع.

فعد الحاجة لاستخدام دالة تتنمي لصنف ما، فرغم أن القاعدة الأساسية والشائعة أن يتم استدعاء الدوال من خلال كائن يتم اشتقاقه من هذا الصنف (كما سيوضح لاحقاً في الباب التاسع). إلا أن هناك حالة تخرج عن هذا المفهوم وهي أن تكون الدوال أو المتغيرات من النوع (**static**)، عندها يُصبح متاحاً استدعاء الدوال أو المتغيرات من هذا النوع دون الحاجة إلى إنشاء كائنات منه.

ولتعريف أي دالة أو متغير على أنه (**static**)، فما علينا إلا أن نكتب الكلمة المحفوظة (**static**) قبل نوع القيمة المرجعة من الدالة كما يظهر بالشكل 7.12 و 7.13 .



شكل 9.12: تعريف الدوال ك **static** وهو تعرف بدوال الصنف أو الدوال الساكنة في بعض المراجع



شكل 9.13: تعريف المتغيرات ك **static** وهي تعرف بمتغيرات الصنف أو المتغيرات المشتركة أو الساكنة

ومثال ذلك:

```

public static int max(int a, int b){ .... }
public static void min (int x [ ]){ .... }
public static char x;
  
```

ولعله الآن قد حان الوقت لكي نفهم لماذا تكتب الدالة الأساسية (`main`) كـ (`static`) والإجابة:

الصنف الأساسي والذي يحتوي على الدالة الأساسية (`main`) هو أول ما يتم استدعاءه من آلية جافا الافتراضية لكي يتم تنفيذ البرنامج، وكما تعلم فإن الدالة الأساسية هي التي تضم الجمل البرمجية التي نريد تنفيذها، فكيف لآلية جافا أن تستدعيها دون إنشاء كائن؟ وأين تقوم بإنشائه إن أردت؟ ليس ممكناً، لذلك عند تعريف الدالة الأساسية (`static`) يصبح بإمكان آلية جافا استدعائها بسهولة وتتنفيذ البرنامج بلا مشاكل.

**الدالة static لا يمكن أن تتعامل أو تستدعي متغيرات أو دوال إلا إذا كانت static**



هذا المفهوم جعل فريق مطوري جافا يعمدون إلى تطوير مجموعة من الدوال التابعة إلى أصناف يمكن استخدامها دون إنشاء كائنات منها، ومثال ذلك: الصنف `math` وهو مهم بالدوال الرياضية المختلفة من حساب (ظا) و(ظتا) و(جا) و(جتا) والقيمة الأكبر والقيمة العشوائية وغير ذلك من الدوال الموضحة في الجدول 7.1. بينما في المثال 7.9، نظهر طريقة استخدامها من خلال تطبيق بعضها ومشاهدة النتائج.

## 7.7 كيفية استخدام الدوال الجاهزة: الصنف Math

الدوال الجاهزة من نوع (`static`) يمكن استخدامها مباشرة من خلال اسم الصنف الخاص بها، دون الحاجة لإنشاء كائنات كما أسلفنا، ومثال ذلك إذا أردنا استدعاء الدالة `x` من الصنف `Math` فإن ذلك يتم بالجملة التالية

`Math.x();`

مع الأخذ بعين الاعتبار هل تعيد قيمة أم لا وهل يحتاج عملها لاستقبال قيمة أم لا، وهذا أمر مهم.

الجدول 7.1، يوضح مجموعة من أهم الدوال التابعة للصنف `Math`.

الدالة	وظيفتها	مثال عليه
<code>abs(x)</code>	تعيد القيمة الموجبة للرقم <code>x</code> الذي قد يكون من أي نوع رقمي.	الدالة <code>abs(-9)</code> تعيد 9 الدالة <code>abs(-3.45)</code> تعيد 3.45 الدالة <code>abs(3.45)</code> تعيد 3.45
<code>ceil(x)</code>	تقوم بتحويل العدد <code>x</code> لأول رقم صحيح أكبر منها.	الدالة <code>ceil(3.5)</code> تعيد 4 الدالة <code>ceil(2.4)</code> تعيد 2
<code>floor(x)</code>	تقوم بتحويل العدد <code>x</code> لأول رقم صحيح أصغر منها.	الدالة <code>floor(3.5)</code> تعيد 3 الدالة <code>floor(2.4)</code> تعيد -3

<p>الدالة <math>\max(3, 7)</math> تعيد 7 الدالة <math>\max(-2, -9)</math> تعيد -2</p> <p>الدالة <math>\min(3, 7)</math> تعيد 3 الدالة <math>\min(-3, -7)</math> تعيد -7</p> <p>الدالة <math>\text{pow}(9.0, 0.5)</math> تعيد 0.3</p> <p>الدالة <math>\sqrt(49.0)</math> تعيد 7.0 الدالة <math>\sqrt(900.0)</math> تعيد 30.0</p> <p>الدالة <math>\text{random}()</math> تعيد 0.590054</p>	<p>تستقبل هذه الدالة رقمين من أي نوع رقمي، وتعيد أحدهما أكبر.</p> <p>تستقبل هذه الدالة رقمين من أي نوع رقمي، وتعيد أحدهما أصغر.</p> <p>تستقبل رقمين، الأول يعتبر الأساس والثاني يعتبر الأس، وثم تعيد قيمة الأول مرفوعاً لقوة الثاني، ولا تستقبل إلا قيمة عشرية.</p> <p>تستقبل قيمة عشرية وتعيد الجذر التربيعي لها.</p> <p>عند استدعاءها تعيد رقم عشوائي بين أكبر وأحياناً غير ذلك فهي عشوائية.</p>	<b>max(x,y)</b>  <b>min(x,y)</b>  <b>pow(x,y)</b>  <b>sqrt(x)</b>  <b>random()</b>
--	--	--

جدول 7.1: بعض الدوال **Math** من الصنف **static**

### مثال توضيحي 7.9:

مستخدماً دوال الصنف **Math**، اكتب برنامجاً لاستخدام الدوال الموضحة في الجدول

7.1

```

1 public class Example9_9 {
2     public static void main(String[] args) {
3         int maxnum = Math.max(5,9);
4         int minnum = Math.min(5,9);
5         double sqrtnum = Math.sqrt(9.0);
6         double floornum = Math.floor(9.7);
7         double ceil = Math.ceil(9.7);
8         double power = Math.pow(2.0, 7.2);
9         double abs = Math.abs(-9.9);
}

```

```

10 System.out.print("Results\n=====\\n");
11 System.out.println("Max of 5 and 9 is:\\t "+maxnum);
12 System.out.println("Min of 5 and 9 is:\\t "+minnum);
13 System.out.println("Sqrt of 9.0 is:\\t "+sqrtnum);
14 System.out.println("Ceil of 9.7 is:\\t "+ceil);
15 System.out.println("Floor of 9.7 is:\\t "+floornum);
16 System.out.println("Power of 2.0 and 7.2 is: \\t"+power);
17 System.out.println("Abs of -9.9 is:\\t "+abs);
18 }

```

```

General Output
----- Configuration: Example9_9
Results
=====
Max of 5 and 9 is: 9
Min of 5 and 9 is: 5
Sqrt of 9.0 is: 3.0
Ceil of 9.7 is: 10.0
Floor of 9.7 is: 9.0
Power of 2.0 and 7.2 is: 147.0333894396205
Abs of -9.9 is: 9.9

Process completed.

```

شكل 7.14: ناتج مثال 7.9

الصنف `Math` أحد أصناف المكتبة `java.lang` وهي تضاف للبرنامج بشكل ضمني، فلا داعي لإدراجها في البرنامج باستخدام الأمر `import` كما يحدث مع صندوق الحوار.



الدوال التابعة للصنف `Math` لا تعتبر كلمات محجوزة (راجع المثال 7.9).

### القيم الثابتة في الصنف `Math`

يحتوي الصنف (`Math`) على متغيرين ثابتين يحتوي كلاً منهما على قيمة رياضية ثابتة، وهما `(Math.PI)` و`(Math.E)`، أما الأولى فهي الثابت 3.14 والذى استخدمناه في حساب مساحة ومحيط الدائرة قبل ذلك، ويستخدم بالصيغة الظاهرية، أما الثاني فهو الأساس الطبيعى للدالة الرياضية المعروفة باسم "لوغاريتم" ومقداره التقريري 2.71828 وكلا القيمتين لا يمكن تغييرهما من خلال البرنامج، حيث أن الثابت في البرمجة لا يُغير وقت تنفيذ البرنامج، وينتَرَى بالكلمة المحجوزة (`final`) كما يلي:

```
public final int x = 5;
```

**خطأ برمجي** يظهر عند محاولة تغيير قيمة المتغير النهائي أثناء عملية التنفيذ.



عند تعريف متغير ما `final static` في الوقت ذاته فلا يهم أن تسبق إدراهما الأخرى.



الدالة `static` تعرف باسم `class method` أيضا نسبة لأنها لا تحتاج كائن تسب له.



## 7.8 مستوى الرؤية (Scope)

في الوقت الذي تم التعامل مع المشاكل البرمجية بقاعدة "فرق تسد" وأصبحنا نبني الحل البرمجي من خلال تجميع مجموعة من الحلول "الأجزاء" الصغيرة، فقد ظهر معنا مفهوم الرؤية ومستواها، بمعنى من يرى من؟ وأين يمكنني رؤية المتغيرات المختلفة في هذا البرنامج ذلك لأن المتغيرات أصبح يتم تعريفها داخل الدوال أحياناً.

هناك عدة مستويات للرؤية هي على النحو التالي:

1. **المتغيرات العامة (Global Variables)**: وهي التي تُعرف خارج الدوال وداخل الصنف، ومستوى رؤيتها كامل على مستوى الصنف، فيمكن لأي دالة التعامل معه، إلا أنه قد يتعارض مع متغيرات تُعرف بذات الاسم، وعندها يتم تفضيل المتغير المحلي عليه شرط أن يحصل التعارض بعد أن يكون المتغيرين قد تم تعريفهم بالفعل وإلا يعتمد المتغير العام.

2. **المتغيرات المحلية للدوال (Local Variables)**: وهي التي تعرف فقط على مستوى دالة من الدوال داخل الصنف، وهذه الدالة لا يمكن التعامل معها إلا داخل هذه الدالة فقط، ومحاولة التفاعل معها من خارج الدالة سيؤدي إلى ظهور خطأ برمجي. كما أن محاولة التفاعل معها قبل تعريفه يؤدي إلى خطأ برمجي واضح.

3. **متغيرات التكرارات (Repetition Variables)**: وهي المتغيرات التي يتم تعريفها داخل التكرار، فهي لا يمكن التفاعل معها خارج التكرار أو طباعتها قيمها، ومحاولة التفاعل معها من خارج التكرار يؤدي إلى ظهور خطأ برمجي.

محاولة التعامل مع متغير في غير مستوى رؤيته يؤدي إلى ظهور خطأ برمجي.



وبتقى حالة واحدة هي أن يتعارض متغيران لهما الاسم ذاته وكل منهما من مستوى رؤية مختلف، ففي هذه الحالة يكون للمترجم (*compiler*) رأياً كالتالي:

- إن كان التعارض حدث داخل دالة أو جملة تكرار، فإن المترجم يعتمد قيمة المتغير المحلي.
  - وإن كان التعارض حدث خارج الدالة أو جملة التكرار، فإن المترجم يعتمد قيمة المتغير الأعم.
- وتوضيح كل ما سبق في المثال 7.10.

### مثال توضيحي 7.10 :

مستخدما كل ما مضى شرحه، قم بتعريف مجموعة من التعريفات والتكرار ليتضح من خلال مفهوم مستوى الرؤية.

```

1 public class Scope {
2     static int x = 5;
3     //i =7;
4     public static void main(String[ ] args) {
5         System.out.println("value of X into main before: "+x);
6         int x =7;
7         for (int i = 0; i<=3; i++)
8             System.out.println("value of i into for is: "+i);
9         //System.out.println("value of i into for is: "+i);
10        System.out.println("value of X into main is: "+x);
11    }
}

```

لاحظ في المثال 7.10، في السطر رقم 2، تم تعريف متغير باسم `x` قيمته 5 وهو متغير عام لأنّه خارج الدوال، بينما داخل الدالة الأساسية تم تعريف المتغير `x` أيضاً في السطر رقم 6 وإعطاءه القيمة 7، في السطر رقم 5 عندما تم طلب طباعة المتغير `x` فكان الناتج قيمة 5، انتبه المتغير تعريفه محلياً وعاماً، فاختار المترجم القيمة العامة (لماذا؟) لأنّ المترجم لا يعتمد متغير قبل تعريفه، والمتغير المحلي `x` لم يتم تعريفه إلا في السطر السادس وبالتالي لم يكن هناك تعارضًا.

في السطر رقم 7 تم تعريف جملة تكرار للعداد (i) وبدء بقيمة 0، في السطر رقم 8 تم طباعة القيمة بنجاح (لماذا؟) لأن جملة الطباعة هذه في مستوى رؤية جملة التكرار، بينما في السطر رقم 9 سيظهر خطأ لأن جملة الطباعة هذه خارج نطاق التكرار، كذلك إذا تم طلب طباعة المتغير خارج الدالة الأساسية سيعطينا المترجم خطأ برمجي.

```
General Output
-----Configuration: Scope
value of X into main before: 5
value of i into for is: 0
value of i into for is: 1
value of i into for is: 2
value of i into for is: 3
value of X into main is: 7
Process completed.
```

شكل 7.15: نتائج المثال التوضيحي 7.10

### 7.9 الاستدعاء الذاتي

الاستدعاء الذاتي هو مفهوم متعلق بالدوال ويُطلق على الدالة التي تقوم باستدعاء ذاتها، أي أنها تكتب أمر الاستدعاء داخلها.

والاستدعاء الذاتي نوعان:

1. استدعاء ذاتي مباشر بحيث لا يكون هناك وسيطاً للاستدعاء بل يتم مباشرة كما في الشكل 7.16.
2. استدعاء غير مباشر بحيث يكون هناك وسيطاً أي أن الدالة A تستدعي الدالة B التي بدورها تستدعي الدالة A وبهذا تكون الدالة A كأنها استدعت ذاتها كما بالشكل 7.17.



شكل 7.16: الاستدعاء الذاتي المباشر



شكل 7.17: الاستدعاء الذاتي غير المباشر

ومن أمثلة الاستدعاء الذاتي حساب مضروب الأعداد، والذي سبق أن قمنا بحسابه من خلال جمل التكرار، وهناك العديد من التطبيقات التي يساعدنا مبدأ الاستدعاء الذاتي في إنجازها بشكل أسرع.

### مثال توضيحي 7.11 :

اكتب دالة لحساب مضروب العدد  $n$ ، ثم قم باختبارها من خلال استدعائهما في الدالة الأساسية لحساب مضروب الأعداد من 0 إلى 10 علماً بأنها تحسب بالمعادلة :

$$n! = n * (n-1) * (n-2) * \dots * 1$$

كما تلاحظ، مضروب العدد يحسب من خلال ضرب العدد في العدد الذي يقل عنه بواحد والناتج يضرب في العدد الذي يقل عنه بواحد وهكذا حتى نصل للعدد 1 فمضروبته يساوي 1 وكذلك الصفر.

هذا يعني أننا نقوم ببناء دالة تستقبل قيمة صحيحة وفي كل مرة الناتج يكون القيمة مضروبة في القيمة مضطرب منها 1 حتى تصبح القيمة 1 عندما يتوقف الاستدعاء لأن مضروب العدد 1 يساوي 1.

```

1 public class FactorialUsingRecursion {
2     public static long factorial(long number)
3     {
4         if (number <= 1)
5             return 1;
6         else
7             return number * factorial(number - 1);
8     }
9     public static void main(String[ ] args) {
10        for (int counter = 0; counter <= 10 ; counter++)
11            System.out.println(counter+"!\t"+ factorial(counter));
12    }
13 }
```

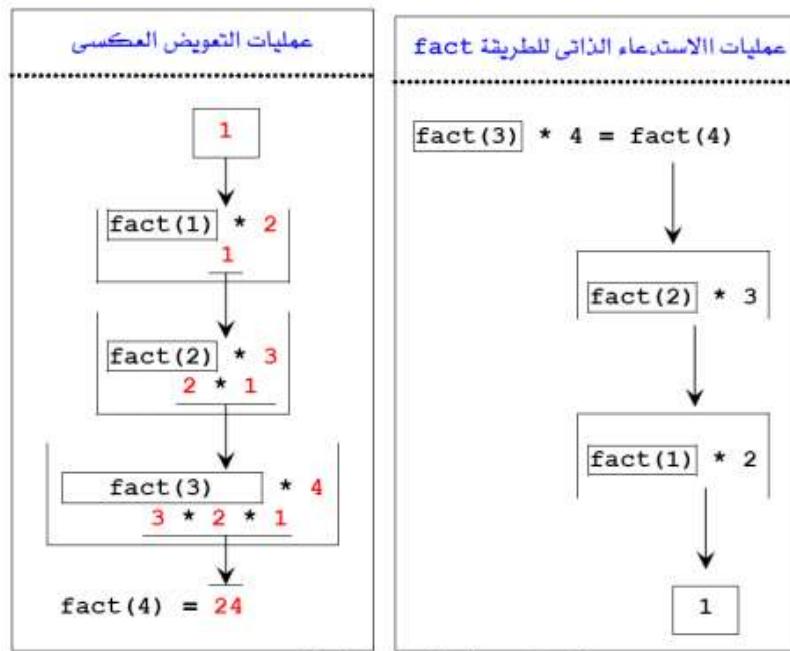
لاحظ أن الدالة (*factorial*) تعيد قيمة من النوع (*long*) لأن الناتج المتوقع منها دائمًا كبيرًا، داخل الدالة يتم دائمًا السؤال هل القيمة الممررة للدالة أقل من أو يساوي الواحد، وذلك لأن مضروب القيم أقل من 1 دائمًا تعتبر

ولذلك دائمًا نفحص القيمة فإن كانت كذلك ينتهي الاستدعاء الذاتي ونعيد قيمة المضروب، وإن كانت القيمة أكبر من الواحد، نعيد قيمة الرقم الممر مضروباً في استدعاء الدالة (factorial) بقيمة جديدة هي الرقم مطروحاً منه واحد، ويستمر الأمر على ذلك حتى يتم استدعاء الدالة برقم قيمته واحد أو أقل لينتهي الاستدعاء الذاتي عنده. أما التكرار في الدالة الأساسية فهو من 0 وحتى 10 ليتم طباعة مضروب كافة هذه الأرقام حيث أننا نستدعي دالة (factorial) في كل دورة من الدورات، وهذا هو الناتج في الشكل 7.18.

General Output	
	Config
0!	1
1!	1
2!	2
3!	6
4!	24
5!	120
6!	720
7!	5040
8!	40320
9!	362880
10!	3628800
Process completed.	

شكل 7.18: ناتج مثال 7.11

وتم عملية حساب قيمة المضروب من خلال حساب القيم بشكل عكسي عند انتهاء (توقف) الاستدعاء الذاتي كما تشاهد في الشكل 7.19 (التصميم من سلسلة مناهج المملكة العربية السعودية).



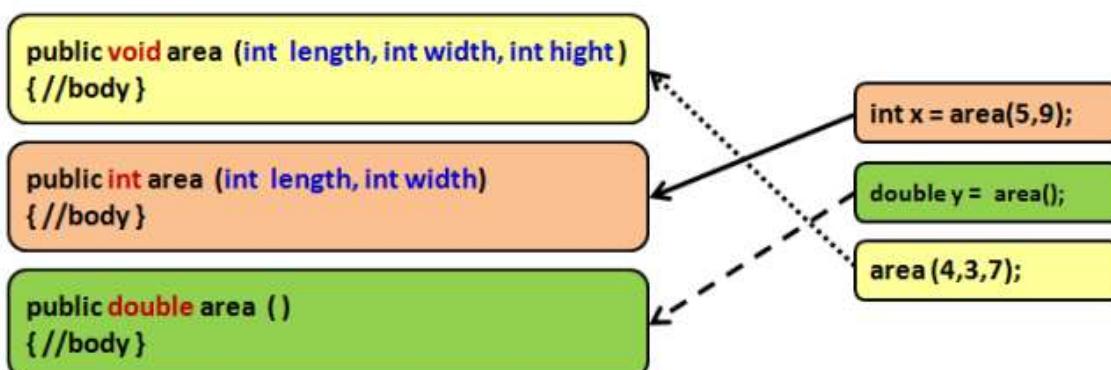
شكل 7.19: طريقة حساب قيمة المضروب

والسؤال الذي يطرح نفسه الآن، استخدام جمل التكرار واستخدام الاستدعاء الذاتي، كلاهما يتحكم في سير العمليات فما الفارق بينهما؟ الفارق أن الاستدعاء الذاتي يعتمد على جمل الاختيار، بينما التكرار يعتمد على بنية جمل التكرار ذاتها. كما أن كلاهما يضم مفهوم التكرار ولكن الاستدعاء الذاتي يتضمن التكرار من خلال تكرار استدعاء الدالة ذاتها وهو تكرار ضمني أما جمل التكرار فتتضمن التكرار من خلال الاستخدام الصريح لأدوات وجمل التكرار التي تعرضنا لها سابقاً.

#### 7.10 التحميل الزائد للدوال (Overloading Methods)

التحميل الزائد للدوال هو كتابة أكثر من دالة لهم اسم واحد في التصنيف ذاته مع التمييز بينهم بتعديل نوع أو عدد القيم الممرة للدالة. أي أننا يمكننا التمييز بين هذه الدوال من خلال التوقيع الخاص بها.

وعند استدعاء الدالة التي لها أكثر من حالة واسم واحد، يتم التمييز بينهم من خلال الفارق في القيم الممرة، تابع مع الشكل 7.20 وكيف يتم اختيار الدالة المناسبة للاستدعاء المناسب؟



شكل 7.20: مفهوم التحميل الزائد للدوال

والتحميل الزائد للدوال نجأ له عندما نحتاج لإنجاز مهمة معينة بأكثر من نوع بيانات أو بأكثر من عدد من البيانات، فمثلاً نحتاج لإنشاء دالة تجمع رقمين وتعيد الناتج، بالتأكيد الجمع قد يحدث لأرقام صحيحة أو أرقام عشرية بمختلف تصنفياتهم، وبالتالي يتم كتابة أكثر من دالة للجمع وتختلف فقط أنواع البيانات الممرة وبالتالي قد تختلف القيم المرجعة.

وتبرز الحاجة لهذا المفهوم في البرمجة شيئاً فشيئاً التوجه حيث بناء الأصناف يحتاج دائماً لوجود أوجه مختلفة للسلوك، فمحاكاة الحساب البنكي يحتاج منا إلى بناء دالة للسحب، والسحب قد يكون بعملات أنواعها مختلفة أو أن أنواع السحب ذاتها مختلفة وبالتالي تحتاج إلى أكثر من صورة للدالة الواحدة.

**مثال توضيحي 7.12:**

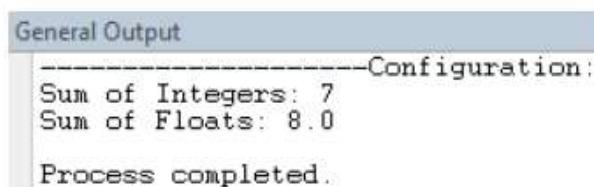
اكتب برنامجاً يحتوي على دالتين لحساب مجموع الأرقام الصحيحة وال العشرية.

```

1 public class Example7_11 {
2     public static int sum(int x, int y){
3         int sum = x+y;
4         return sum;
5     }
6     public static float sum(float x, float y)
7     {
8         float sum = x+y;
9         return sum;
10    }
11    public static void main(String[] args) {
12        System.out.println("Sum of Integers: "+sum(4,3));
13        System.out.println("Sum of Floats: "+sum(4.9f,3.1f));
14    }

```

لاحظ في هذا المثال، تم كتابة دالتين، الأولى تستقبل رقمين صحيحين، وتعيد قيمة صحيحة أما الثانية تستقبل رقمين عشريين وتعيد قيمة عشرية، وبالتالي عند الاستدعاء يظهر لك المحرر أن هناك أكثر من صورة يمكنك استدعاءها والتمييز يتم من خلال القيم الممررة.



شكل 7.21: ناتج مثال 7.11

## 7.11 أمثلة وتدريبات عامة

1. استخدم الكلمات المناسبة لتعبئنة الفراغات في الجمل التالية:

- أ- تعتبر الـ \_\_\_\_\_ هي أصغر الـ *Modules* في لغة جافا.
- ب- مجموعة من التعليمات البرمجية التي تكتب مرة واحدة ولها بداية ونهاية وتتفذ مهمة معينة ويمكن استدعاءها كلما احتجنا لها هي \_\_\_\_\_.
- ت- يضم توقيع الدالة ثلاثة مكونات أساسية هي \_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_.
- ث- جسم الدالة يضم جملة *return* عندما \_\_\_\_\_.
- ج- عملية طلب العمل من الدالة يطلق عليه اسم \_\_\_\_\_.
- ح- يؤدي استدعاء دالة *static* داخل الدالة الأساسية إلى \_\_\_\_\_.
- خ- عند إرجاع قيمة يتعارض نوعها مع النوع المعرف في توقيع الدالة يظهر خطأ \_\_\_\_\_.

2. حدد صحة أو خطأ كلا من العبارات التالية مع التعليق:

- ( ) أ- الدالة التي تطبع قيمة المرتب الصافي هي دالة تعيد قيمة عشرية.
- ( ) د- الدالة التي تقوم بتبادل قيم متغيرين يرمز فيها للقيمة المرجعة بـ *void*.
- ( ) ب- الدوال تتقسم من حيث استقبال القيم إلى ثلاثة أنواع.
- ( ) ذ- الجملة *return* يمكن كتابتها في أي موضع داخل الدالة حسبما يرأيه المبرمج وقد يستخدم أكثر من واحدة.
- ( ) ت- في حالات استثنائية يسمح بكتابة دالة داخل غيرها.
- ( ) ث- عند عدم استدعاء الدوال داخل الدالة الأساسية فإنها تبقى كامنة بلا تأثير ولا نشعر بها.
- ( ) ج- عند تمرير مصفوفة للدالة يتم كتابة اسم المصفوفة بجانب [ ].

3. حول الجمل العربية الآتية إلى جمل يفهمها مترجم جافا:

- أ- التوقيع الخاص بدالة *max* تستقبل قيمتين صحيحتين وتعيد أكبرهما.

بـ- التوقيع الخاص بدالة تستقبل مصفوفة من النوع النصي وقيمة نصية وتعيد نتيجة البحث (موجود أم لا).

تـ- تعريف دالة تحتاج للتعامل معها دون اشتقاق كائن من الصنف.

4. اكتب دالة لحساب سلسلة فایبوناسی والتي تحسب حسب المعادلة التالية:

$$\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$$

5. اكتب دالة تستقبل ثلاثة قيم صحيحة من المستدعى ثم تعيد متوسطهم الحسابي، اطلب الدالة من الدالة الأساسية.

6. اكتب دالة تستقبل مصفوفة من النوع الصحيح وتعيد الدالة مجموع عناصر هذه المصفوفة. ثم قم باستدعائها داخل الدالة الأساسية.

7. اكتب برنامج يقرأ رقم من المستخدم رقم حقيقي ثم يطبع

- جذر التربيع

• حاصل قوته للعدد 4

• تقریبه لأقرب أكبر رقم وأصغر رقم

8. اكتب دالة لحساب الوزن المثالي، علماً بأن الوزن المثالي يتم حسابه من المعادلة :

الوزن المثالي يساوي الطول مطروحاً منه 100

9. اكتب دالة تستقبل رقمًا صحيحاً وتحدد هل هو زوجي أم فردي ثم قم باستدعاء الدالة وطباعة الناتج.

## 7.12 تمارين ذاتية الحل

1. مستخدما الدوال الجاهزة في لغة جافا قم بكتابة برنامج لحساب المعادلة التالية:

$$X = \sqrt{\frac{150 - Y}{Y}} + |Y|$$

2. اكتب دالة تستقبل مصفوفة نصية وقيمة نصية ثم تعيد هل القيمة النصية موجودة داخل المصفوفة أم لا.

3. مستخدما الدوال الجاهزة والعملية %، اكتب دالة تستقبل قيمة عشرية وتعيد باقي قسمته على الرقم 2 عند تقریبه لأقرب رقم أكبر منه.

4. مستخدماً إمكانیات الصنف **Math**, اكتب دالة تستقبل نصف القطر وتعيد مساحة ومحیط الدائرة.
5. مستخدماً مفهوم التكرار ثم الاستدعاء الذاتي، اكتب برنامجاً لحساب مجموع الأعداد من 1 وحتى N حسب المعادلة التالية:

$$\sum_{i=1}^N i = N + \sum_{i=1}^{N-1} i = N + N - 1 + \sum_{i=1}^{N-2} i$$

6. اكتب دالة تستقبل مصفوفة على أن تعيد الدالة **true** إن كانت كافة عناصر المصفوفة قبل القسمة على 2 إلا تعيد **false**.

7. وضح بالأمثلة أنواع الدوال من حيث استقبال القيم والاستدعاء.

8. وضح المقصود بمفهوم التحميل الزائد للدوال مع ضرب الأمثلة.

9. ما المفهوم بالدالة **static** مع ذكر ثلاثة دوال من هذا النوع وتوضيحهم بالأمثلة.
10. حول الجمل العربية التالية إلى جمل عربية تفهمها لغة جافا

أ- دالة تستقبل مصفوفة صحيحة وتعيد مجموع عناصرها الزوجية.

ب- دالة تطبع كلمة "محمود رفيق الفرا" عشر مرات كلما تم استدعاءها.

ت- دالة تستقبل درجة الحرارة بالسلزيس وتعيدها بالفهرنيت.

8. حدد الأخطاء ثم قم بتصحيحها في القطعة البرمجية التالية:

```
public class Test {
    public static method1(int n, m) {
        n += m;
        method2(3.4);
    }
    public static int method2(int n) {
        if (n > 0) return 1;
        else if (n == 0) return 0;
        else if (n < 0) return -1; }
}
```

9. بالرسم والأمثلة، ما هو مفهوم الرؤية للمتغيرات المحلية؟

١٠. اكتب دالة تقوم بطباعة الأعداد في سطر واحد بما يكفي رقم السطر ، وعدد الأسطر يمررها المستخدم.

**إجابات التدريبات العامة:**

١. (أ) الدالة (ب) الدالة (ت) نوع القيمة المرجعة، اسم الدالة، القيم الممررة للدالة (ث) تعيد الدالة قيمة (ج) الاستدعاء (ح) سير البرنامج بلا مشاكل (خ) برمجي

حدد صحة أو خطأ كلاما من العبارات التالية مع التعليل:

أ- (خاطئة) الطباعة لا تعتبر قيم مرجعية.

ب- (صحيحة)

ت- (خاطئة) تنقسم إلى نوعين تستقبل ولا تستقبل

ث- (صحيحة)

ج- (خاطئة) لا يسمح البتة

ح- (صحيحة)

خ- (خاطئة) فقط اسم المصفوفة

.٣

أ- int max (int a, int b)

ب- String search (String [ ]x, String key)

ت- public static void x() { /\*---\*/ }

.٤

```
public static long fibonacci( long number )
{
    if ( ( number == 0 ) || ( number == 1 ) ) // base cases
        return number;
    else // recursion step
        return fibonacci( number - 1 ) + fibonacci( number - 2 );
}
```

```
public static void main(String[] args) {
    for ( int counter = 0; counter <= 10; counter++ )
        System.out.println("Fibonacci of "+counter+" is "+fibonacci(counter));
}
```

شكل 7.22: حل تمرین 4

.5

```
public float average (int a, int b,int c)
{
    float avg = (a+b+c)/3f;
    return avg;
}
```

```
float ret = average(3,3,5);
System.out.println("Hello World!"+ret);
```

شكل 7.23: حل تمرین 5

.6

```
public class method_test {
    public static int sum_array(int [] arr)
    {
        int sum =0;
        for (int i =0 ; i< arr.length; i++)
            sum= sum + arr[i];

        return sum;
    }

    public static void main(String[] args) {
        int []array = {10, 20, 23, 21, 11, 243};
        int x = sum_array(array);
        System.out.print(x);

    }
}
```

شكل 7.24: حل تمرین 6

.7

```

import java.lang.Math;
import javax.swing.JOptionPane;

public class method_test {

    public static void main(String[] args) {

        String input = JOptionPane.showInputDialog("Enter real numbers");
        float x = Float.parseFloat(input);
        System.out.println(x);
        System.out.println(Math.pow(x, 4));
        System.out.println(Math.sqrt(x));
        System.out.println(Math.ceil(x));
        System.out.println(Math.floor(x));
    }
}

```

شكل 7.25: حل تمرين 7

.8

```

public static int calcweg ( int len )
{
    int waig = len - 100;
    return waig;
}

public static void main(String[] args) {
    String y = JOptionPane.showInputDialog("enter your length");
    int z = Integer.parseInt(y);
    int x = calcweg(z);
    JOptionPane.showMessageDialog(null,"The perfect weight is "+x);
}

```

شكل 7.26: حل تمرين 8

```

public class MethodEvenOdd {

    public boolean EvenOrOdd(int num) {

        if (num %2 ==0)
            return true;
        else
            return false;
    }

    public static void main(String[] args) {

        boolean f = EvenOrOdd(7);
        if (f)
            System.out.println("even");
        else
            System.out.println("odd");
    }
}

```

شكل 7.27: حل تمرین 9

انتهى الباب

الضمير صوت هادئ يخبرك بأنَّ أحداً  
ينظر إليك، والإيمان بالله صوت محبوب  
يُخبرك أنَّ لكل مخلص نصيب. فابحث  
بترتيب عن هذا الصوت تُنعم وتُغنِّم.

## الباب الثامن

### خوارزميات البحث والترتيب في المصفوفات

(Algorithms of Search and Sort in Arrays)

يتوقع من الدارس في نهاية هذا الباب أنَّ :

- يُدرك الحاجة إلى توفر خاصية البحث في التطبيقات.
- يُعدد تطبيقات تحتاج لتوفر خاصية البحث.
- يُدرك الحاجة إلى توفر خاصية الترتيب في التطبيقات.
- يُعدد تطبيقات تحتاج لتوفر خاصية الترتيب.
- يُقدر الحاجة إلى ترتيب البيانات وتطبيقاته.
- يشرح خوارزميتي البحث الخطى والبحث الثنائى بالرسم والخطوات.
- يتقن تطبيق خوارزميتي البحث الخطى والبحث الثنائى عن البيانات.
- يشرح خوارزميتي الفقاعات الهوائية والاختيار لترتيب البيانات بالرسم والخطوات.
- يتقن تطبيق خوارزميتي الفقاعات الهوائية والاختيار لترتيب البيانات.

يتوفر لهذا الباب شرائح العرض (PowerPoint) وكذلك ملفات مرئية من إعداد مؤلف الكتاب؛ يمكنك الحصول عليها من خلال الموقع الأكاديمي للمؤلف [mfarra.cst.ps](http://mfarra.cst.ps)



## الباب الثامن، خوارزميات البحث والترتيب

### 8.1 مقدمة

تُعدُّ معالجة البيانات هي الشاغل الأكبر لأصحاب الأعمال من وجهة نظرى، ذلك لأنَّ هذا المفهوم يؤثر في كثير من جوانب الأعمال التي من شأنها الدفع بعملية المعالجة لأفضل حالاتها، خاصةً عندما نتحدث عن تطبيقات تعامل مع آلاف وعشرات الآلاف من السجلات المخزنة في هيئات مختلفة من تراكيب البيانات مثل المصفوفات وغيرها، فلِكى تتحذَّر قرازاً فأنت بحاجة إلى البحث في هذه التراكيب عن قيم بعض البيانات، وبالتاليَ تأكيد عندما تكون بياناتك مرتبة ترتيباً ما تكون عملية البحث أكثر سهولة وسرعة.

في هذا الباب سنشرح مفهومي البحث والترتيب في المصفوفات الأحادية لما لهما من أهمية كبيرة مع استعراض مثالين من خوارزمياتهم، علماً بأنَّ ما سأقدمه من خوارزميات في هذا الصدد لا يعتبر كل شيء وإنما هو ما يكفى للدارس المستهدف ليحصل على تصور مناسب عن هذين المفهومين.

في الفصل 8.2 سنناقش الحاجة إلى عمليات البحث وتطبيقاتها ثم نستعرض خوارزميتين للبحث في الفصل 8.3، بينما في الفصل 8.4 نستعرض الحاجة إلى خوارزميات الترتيب وتطبيقاتها قبل أن نستعرض خوارزميتين للترتيب منها في الفصل 8.5.

### 8.2 الحاجة لعملية البحث وتطبيقاتها العملية

لعله لا يغيب عن أحدنا في هذا العصر حجم التطبيقات البرمجية والحياتية التي تحتاج فيها إلى البحث عن بيانات سواء في المستشفيات، المؤسسات التعليمية، الوزارات، جهاز الحاسوب الشخصي، موقع الإنترنٌت وغيرها؛ فكل هذه الأماكن تحتاج فيها إلى تطبيق البحث. ففي المستشفيات تحتاج للبحث عن بيانات مريض من خلال اسمه أو رقمه، أو عن بيانات علاج محدد من خلال اسمه، أو عن موضع غرفة من خلال رمزها، بينما في المؤسسات التعليمية تحتاج البحث عن كشف درجات طالب من خلال رقمه الجامعي أو البحث عن الجدول الدراسي لأحد المدرسين من خلال رقمه الوظيفي أو اسمه، وغير ذلك الكثير.

هذه من تطبيقات البحث في المؤسسات المختلفة التي يتم تنفيذها يدوياً، أليس كذلك؟. الآن فكر قليلاً، لماذا يحتاج العاملون في هذه المؤسسات للبحث المعتمد على الحوسبة؟!، لماذا لو أنَّ البيانات الضخمة التي تنشر يومياً على شبكات المؤسسات المختلفة الخاصة وعلى الشبكة العالمية لا يمكننا البحث فيها إلا يدوياً؟، كيف ستكون السهولة والسرعة؟ بالتأكيد لن نستطيع الحصول على نتائج سريعة بل قد يشعر الموظف المختص بالملل

قبل أن يجد ما يبحث عنه. لهذا كان البحث من خلال خوارزميات واضحة ومرتبة ويتم تطبيقها من خلال برنامج محوسب هو أفضل ما يمكن تقديمها الآن في هذا الصدد.

### 8.3 خوارزميات البحث

تتمحور خوارزميات البحث حول عملية لتحديد وجود قيمة محددة في المصفوفات من عدمها، وستناقش في هذا الفصل نوعين من الخوارزميات التي يمكن استخدامها للبحث في أي مجموعة من البيانات وخاصة في المصفوفات. حيث تناولت:

- خوارزمية البحث الخطى (Linear search).
- خوارزمية البحث الثنائى (Binary search).

#### 8.3.1 خوارزمية البحث الخطى:

تقوم هذه الخوارزمية بفحص كافة الخلايا في المصفوفة بشكل متسلسل للبحث عن القيمة التي تتطابق مع القيمة التي نبحث عنها وتعرف باسم (Key value)، فإذا وُجِدَت الخلية التي تحتوي على القيمة المطلوبة يتم إيقاف البحث وإعادة عنوان الخلية التي تضم القيمة المطلوبة، وإذا لم تجد القيمة يستمر البحث حتى نصل لنهاية المصفوفة ثم يتم إعادة قيمة سالبة ولتكن (-1) (لماذا نختار قيمة سالبة كإشارة لعدم وجود القيمة المطلوبة؟). ومثال على ذلك، افترض الأرقام التالية من اليسار إلى اليمين هي مصفوفة ونريد البحث عن القيمة 11:

43    65    54    23    12    9    11    79    21    3

للبحث باستخدام خوارزمية البحث الخطى نقوم بما يلى:

1. نبدأ دوراناً يبدأ من الخلية الأولى ويستمر حتى آخر خلية.
  2. في كل دورة يتم فحص الشرط التالي: هل قيمة الخلية تساوى القيمة 11؟
- أ- إن وُجِدَت القيمة يتم إعادة عنوان الخلية التي تضم القيمة وهي موجبة محصورة من 0 إلى (حجم المصفوفة - 1) وفي حالتها سيعيد القيمة 6.
- ب- إن لم يجدها في نهاية الدوران يتم إعادة قيمة سالبة.

#### مثال توضيحي 8.1:

قم بتطوير دالة لتطبيق البحث الخطى على مصفوفة من النوع الصحيح.

```

1. public static int linearSearch(int[] list, int key) {
2.     for (int i = 0; i < list.length; i++) {
3.         if (key == list[i])
4.             return i;
5.     }
6.     return -1;
7. }
```

هذه الخوارزمية تحتاج في المتوسط لفحص نصف عناصر المصفوفة حتى تجد العنصر المطلوب إن كان موجوداً وبالتالي فإن الوقت الذي تحتاجه الخوارزمية لإنجاز عملها يزداد طردياً بزيادة عدد العناصر ولذلك تُعد غير مناسبة للمصفوفات ذات العناصر الكثيرة.

عند الحاجة لفحص عناصر المصفوفة ذات القيم التصية يمكنك استخدام الدالة *equals* أو *compareTo* كما ذكرنا في الباب الرابع.



### 8.3.2 خوارزمية البحث الثاني:

نظراً للتكلفة الزمنية الكبيرة لتنفيذ خوارزمية البحث الخطى بحث المبرمجون ومحظوظون النظم عن خوارزميات أكثر سهولة وفعالية وسرعة في التنفيذ على مستوى معالجة البيانات الوصول للنتيجة الصحيحة (موجود، غير موجود)، ومن هذه الخوارزميات خوارزمية البحث الثاني وهي خوارزمية لا تعمل إلا مع بيانات مرتبة ترتيباً ما (تنازلي، تصاعدي، أبجدي).

وتلخص هذه الخوارزمية فيما يلي على اعتبار ترتيب العناصر من اليسار إلى اليمين:

1. تحديد عنوان الخلية الوسطى (الخلية التي تقع في منتصف المصفوفة) ويتم حسابها من المعادلة:

$$\text{عنوان الخلية الوسطى} = (\text{عنوان أول خلية} + \text{عنوان آخر خلية}) \div 2 \text{ مع تجاهل الكسور}$$

2. فحص إن كانت قيمة هذه الخلية تساوي القيمة المطلوبة، فإن كانت يتم إعادة عنوان الخلية.

3. وإلا يتم فحص إن كانت القيمة المطلوبة أصغر من قيمة المنتصف أو أكبر وبناءً على ذلك يتم اعتماد نصف المصفوفة وتتجاهل النصف الآخر كالتالي:

أ- إن كانت قيمة الخلية الوسطى أكبر من القيمة المطلوبة فيتم تجاهل النصف الثاني (الأيمن)

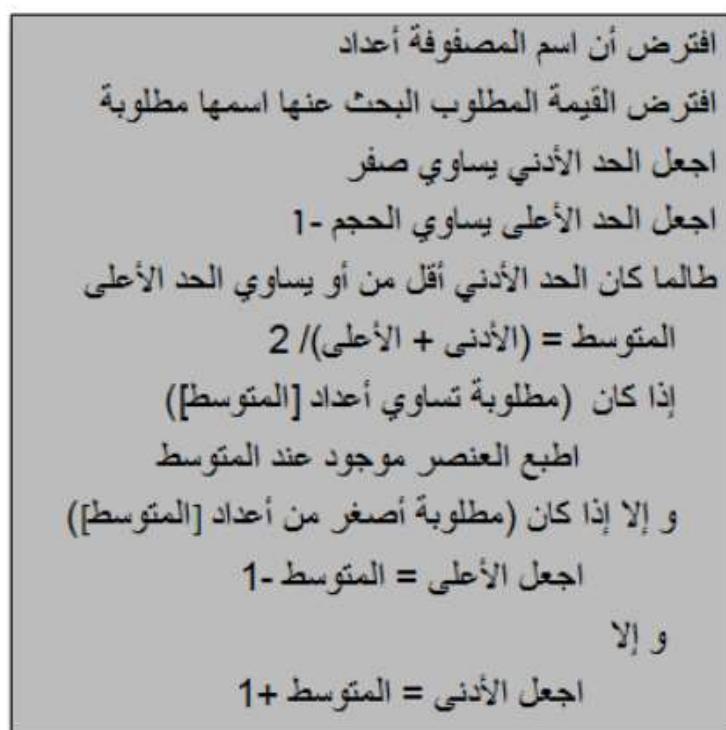
واعتماد الأول (الأيسر) وبذلك يُصبح عنوان آخر خلية يساوي (عنوان الخلية الوسطى - 1)

وعنوان الخلية الأولى كما هو.

بـ إن كان قيمة الخلية الوسطى أصغر من القيمة المطلوبة فيتم تجاهل النصف الأول واعتماد الثاني وبذلك يُصبح عنوان أول خلية يساوي (عنوان الخلية الوسطى + 1) وعنوان الخلية الأخيرة كما هو.

4. يتم تكرار هذه الخطوات طالما كان عنوان الخلية الأولى أقل من أو يساوي عنوان الخلية الأخيرة وإلا يتم إعادة قيمة سالبة تعبيراً عن عدم وجود القيمة.

الخوارزمية موضحة باختصار في الشكل 8.1 ومزيد من الوضوح في المثال التوضيحي 8.2 بالرسم والخطوات.



شكل 8.1: خوارزمية البحث الثاني

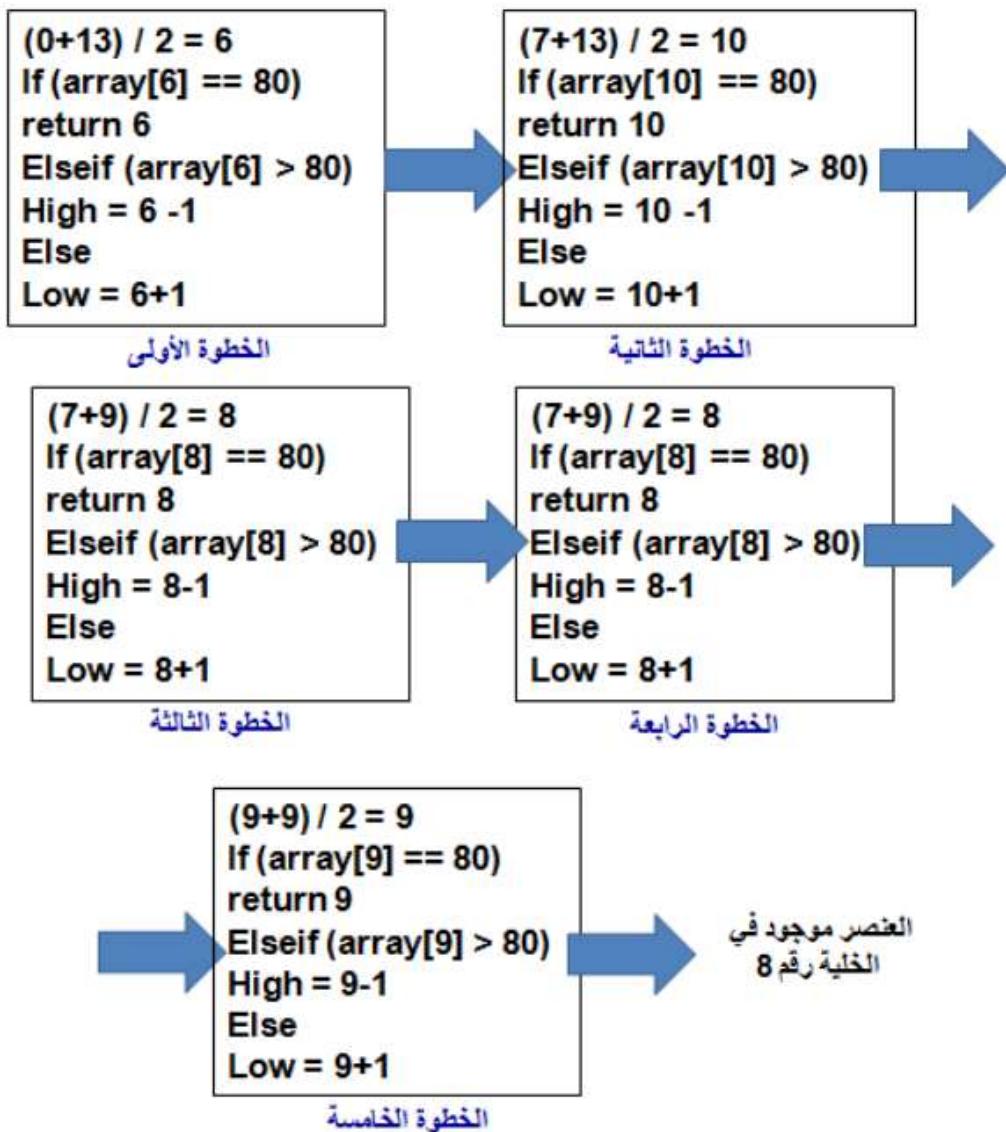
### مثال توضيحي 8.2:

مستخدماً الرسم، قم بتوضيح الخطوات الكاملة للبحث عن القيمة 18 باستخدام خوارزمية البحث الثاني في المصفوفة التالية:

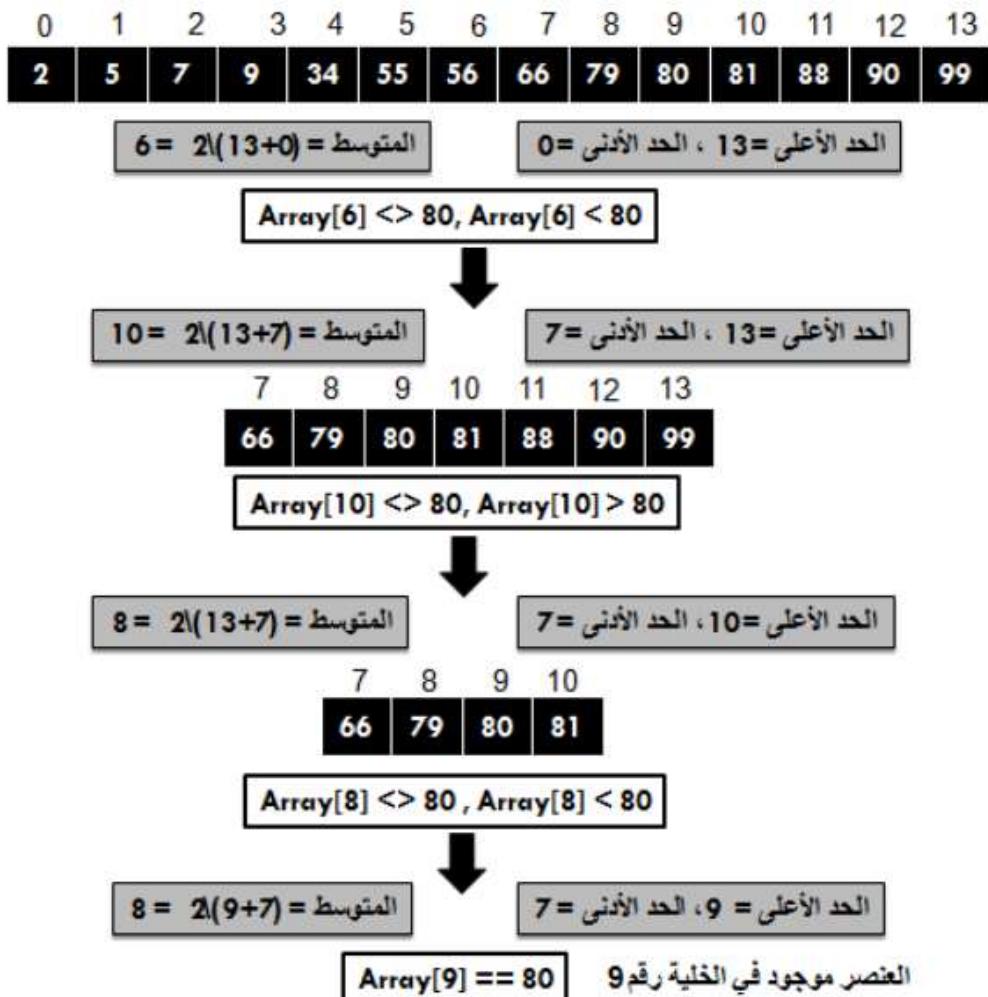
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
array	2	5	7	9	34	55	56	66	79	80	81	88	90	99

## مختصر مفهوم الحل:

كما أوضحنا سابقاً، نحدد عنوان الخلية الوسطى من خلال قسمة مجموع (عنوان العنصر الأخير وعنوان العنصر الأول) على العدد 2 وهو  $(0+13)/2 = 6$  وبالتالي يكون العنصر رقم 6 هو المنتصف، فنفحص الآن هل  $(array[6] == 18)$  إن كان نعم وبالتالي يكون الرقم تم إيجاده وتُعيد عنوان الخلية (الرقم 6)، وإلا نسأل هل  $(array[6] > 18)$  إن كان نعم فهذا يعني أنَّ الرقم 18 يوجد في النصف الأيسر فنعتبر في هذه الحالة أنَّ العنصر الأخير هو النصف - 1 (أي = 5) والعنصر الأول هو ذاته ونعيد المحاولة مع القيم الجديدة ونكرر هذه العمليات طالما كان الحد الأدنى أقل من الحد الأعلى وإلا نعتبر القيمة غير موجودة.



وفي الشكل التالي، تابع التمثيل للمصفوفة في الخطوات الخمس الماضية لمزيد من الفهم.



الشكل 8.2 : تمثيل خطوات تطبيق خوارزمية البحث الثاني

### مثال توضيحي 8.3

قم بتطوير دالة لتطبيق البحث الثاني على مصفوفة من النوع الصحيح.

```

1 public static int binarySearch(int[] list, int key) {
2     int low = 0;
3     int high = list.length - 1;
4     while (high >= low) {

```

```

5 int mid = (low + high) / 2;
6 if (key < list[mid])
7     high = mid - 1;
8 else if (key == list[mid])
9     return mid;
10 else
11     low = mid + 1;
12 }
13 return -1;
14 }
```

هذه الخوارزمية في أسوء حالاتها عند البحث في 4000 عنصر ستحتاج فقط للمقارنة 11 مرة، حيث أنك تستطيع تقسيم هذا العدد على الرقم 2 فقط 11 مرة، وكما تابعنا فإن الخوارزمية تركز العمل على تقسيم المصفوفة في كل مرة على قسمين تعامل مع أحدهما وتتجاهل الآخر.

عند مقارنة ذلك مع خوارزمية البحث الخطى فإن البحث الخطى يحتاج في أسوء الحالات إلى 4000 عملية فحص مما يجعل خوارزمية البحث الثنائى أكثر مناسبة مع المصفوفات المرتبة ذات الحجم الكبير.

في المثال 8.3، عند استبدال الشرط  $high \geq low$  بالشرط  $high > low$  سينتج عنه خطأ منطقى، قم بفحصه الآن.



عند البحث عن قيمة يتكرر وجودها في المصفوفة، فإن البحث لا يتوقف ويعيد عنوان أحد القيمتين حسب عمليات البحث عن عنوان الخلية الوسطى. (قم بتجربتها الآن)



#### 8.4 الحاجة إلى عملية الترتيب وتطبيقاتها العملية

عملية الترتيب هي العمل على وضع العناصر في ترتيب محدد سواء كان ذلك تصاعدياً من الأصغر إلى الأكبر أو تنازلياً من الأكبر إلى الأصغر وكذا أبجدياً. ومثلها مثل البحث تعتبر العملية من أكثر العمليات أهمية وشيوعاً في حياتنا بشكل عام وفي الحاسوب بشكل خاص فالعمل دائمًا مع الأشياء التي من حولنا وهي مرتبة يساعدنا على سرعة وسهولة الإنجاز، فأنت في مكتبك إن كانت الأوراق والكتب مرتبة بكيفية محددة وأردت أن تبحث عن شيء فيها، فإن هذا سيكون أسهل كثيراً من التعامل معها وهي غير مرتبة، وكذا ملفات العملاء في الشركات والطلبة في المدارس والكليات والمرضى في المستشفيات وأسماء أصدقائك في المحمول وغير ذلك من التطبيقات التي لا غنى فيها عن ترتيب البيانات ذلك لأن الترتيب يساعد هذه المؤسسات بكل تأكيد على إنجاز المهام بسرعة أكبر. ولعلك انتبهت إلى الفارق بين سرعة خوارزمية البحث الخطى وخوارزمية البحث الثنائى،

فالاولى تعامل مع عناصر غير مرتبة، بينما الثانية تعامل مع عناصر مرتبة، مع الأخذ بعين الاعتبار قوة الخوارزمية ذاتها أنها تعتمد على مصفوفة مرتبة.

### 8.5 خوارزميات الترتيب

العديد من خوارزميات الترتيب سعى إلى الوصول للترتيب إما بأسرع وقت وإما بأسهل طريقة برمجة أو غير ذلك، مما أنتج لنا العديد من الخوارزميات المتقاونة في المميزات؛ إلا أنَّ كافة هذه الخوارزميات في نهاية الأمر قادرة على ترتيب المصفوفة وببقى الفارق بينهم في الوقت المستغرق والمساحة المستخدمة والجهد البرمجي اللازم لإنجازها.

في هذا الفصل سنستعرض خوارزميتين للترتيب هما:

- خوارزمية الفقاعات الهوائية (*Bubble Sort*)
- خوارزمية الاختيار (*Selection Sort*)

وكلاهما يعتمد على عملية تبديل القيم بين موقعين وهو ما يُعرف باسم (*Swap*) والتي تتلخص في أننا نحتاج لاستخدام متغير إضافي يعمل كمخزن مؤقت يمكننا من استبدال القيم وذلك كما يلي:

```

1 int x=10;
2 int y=15;
3 hold = x;
4 x= y;
5 y = hold;
```

بعد الجمل (3، 4 و 5) تُصبح قيمة المتغير (x) تساوي 15 بينما قيمة المتغير (y) تساوي 10.

وهاتين الخوارزميتين ما هما إلا مثال على خوارزميات الترتيب حيث يتتوفر الكثير غيرهم مثل خوارزمية الدمج (*Merge sort*) التي تعتمد مبدأ "فرق تسد" وخوارزمية الإدراج (*Insertion sort*) التي تعتمد على اختيار المكان المناسب للقيمة وإدراجها فيها وغيرهم من الخوارزميات.

#### 8.5.1 خوارزمية الفقاعات الهوائية (*Bubble Sort Algorithm*)

عند استخدام هذه الخوارزمية لترتيب العناصر ترتيباً تصاعدياً مثلاً، فإنها تعتمد على مقارنة عناصر المصفوفة عنصرين عنصرين بالترتيب فإن كان الأول أكبر من الثاني يتم عمل تبديل لقيمهما (*swap*) وإن كان الأول

أصغر من الثاني يبقى الحال على ما هو عليه، وهكذا يتم تكرار العملية حتى نصل للترتيب المناسب وهذا يتطلب المرور على المصفوفة أكثر من مرة، حيث تحتاج هنا مصفوفة حجمها ( $s$ ) من العناصر، إلى جملتي دوران متداخلتين على النحو التالي:

- الأولى "الخارجية": يتم فيها الدوران بعدد ( $s-1$ ) مرات وهي تستخدم لتكرار المرور على المصفوفة أكثر من مرة حتى يُضمن الترتيب، لأننا لا يمكننا ترتيب المصفوفة بالمرور عليها مرة واحدة فقط حيث أن المصفوفة التي يقع فيها عنصر قيمته 1 في آخر خلية بينما يقع عنصر قيمته 20 في الخلية الأولى، فإنه لا يمكن أن نصل لمقارنتهم ببعضهم البعض من جولة واحدة بل تحتاج إلى أكثر من دوران.
  - الثانية "الداخلي": يتم فيها الدوران بعدد ( $s$ ) مرات تبدأ من الصفر حتى ( $s-1$ )، وهي تُستخدم للمرور على عناصر المصفوفة ومقارنة عناصرها زوجاً بعد زوج.
- والخوارزمية الخاصة بهذه الطريقة موضحة بالشكل 8.3، حيث أن:

*arraylength* = (b)

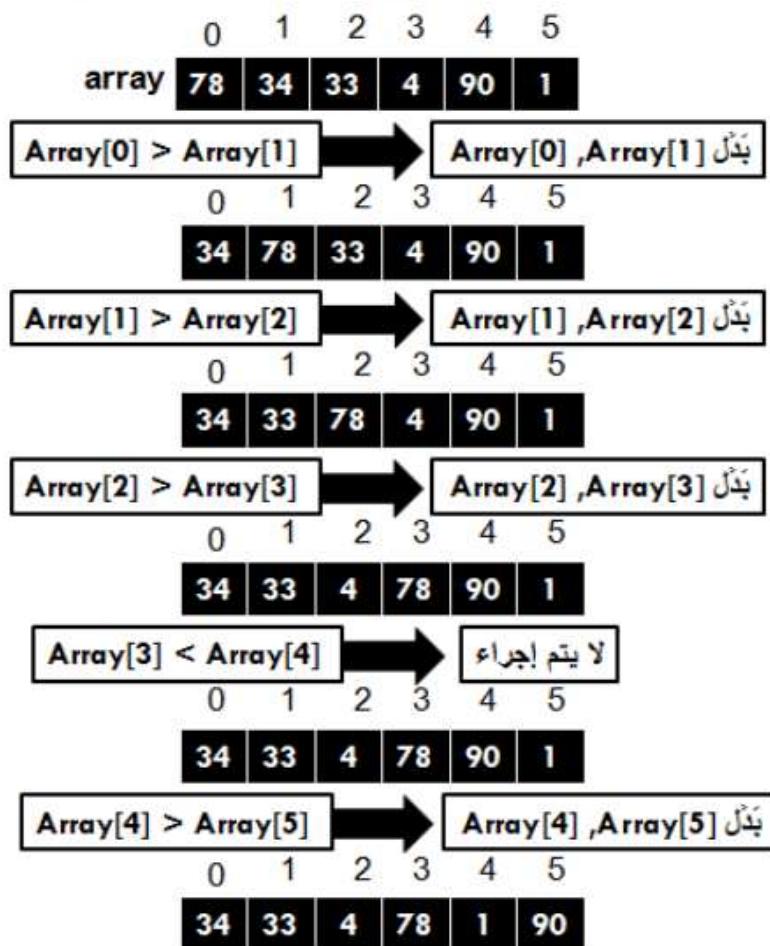
```
for ( int pass = 1, pass < arrayLength, pass++ )
    for ( int i = 0; i < arrayLength - 1; i++ )
        if ( b[ i ] > b[ i + 1 ] )
            بدل (b[ i ], b[ i+1 ] )
```

شكل 8.3: خوارزمية الاختيار للترتيب

#### مثال توضيحي 8.4:

مستخدماً خوارزمية الفقاعات الهوائية، قم بترتيب المصفوفة التالية مستخدماً الرسم التوضيحي

78، 1، 34، 4، 33، 90



شكل 8.4: الرسم التوضيحي لدوران خارجي واحد من مثل 8.4

وهذا الدوران الخارجي، ينكرر بعدد عناصر المصفوفة - 1 أي (5 مرات)، وفي كل دورة جديدة تقترب المصفوفة من الترتيب أكثر فأكثر، لنصل في النهاية إلى الشكل 8.5.

0	1	2	3	4	5
1	4	33	34	78	90

شكل 8.5: الناتج النهائي لمثال 8.4

انتبه إلى أنَّ الرقم 1 في الخلية الخامسة سيحتاج إلى أربع جولات (تكرارات) داخلية جديدة حتى يصل إلى المكان المفروض وهذه التكرارات الأربع هي المتبقية بالفعل في المثال، حيث إننا استفادنا تكرارًا داخلياً واحد حتى الآن.

في المثال التوضيحي 8.5 القادم، سنعمل على تطبيق هذه الخوارزمية من خلال لغة جافا، وباستخدام المصفوفة ذاتها، وبنك يصبح مبدأ عمل هذه الخوارزمية أكثر وضوحاً.

### مثال توضيحي 8.5:

مستخدماً خوارزمية الفقاعات الهوائية، اكتب دالة لترتيب المصفوفة المستخدمة في مثال 8.4.

```

1 public void bubbleSort(int num[ ]) {
2     for (int i =1; i<num.length; i++)
3         for (int j = 0; j<num.length-1;j++)
4             {
5                 if(num[ j ]>num[ j+1 ])
6                 {
7                     hold = num[ j ];
8                     num[ j ]= num[ j+1 ];
9                     num[ j+1 ]=hold;
10                }
11            }

```

في الجمل البرمجية السابقة، لاحظ أنَّ الأسطر من 7 حتى 9 هي للقيام بتبادل القيم الخاصة بالخلايا كما شرحناها سابقاً، حيث يتم تخزين قيمة المتغير الأول مثلاً في متغير احتياطي (*hold*)، ثم يتم وضع القيمة الثانية في المتغير الأول وبعدها يتم وضع قيمة المتغير الاحتياطي في المتغير الثاني.

وبدراسة هذه الخوارزمية وفهم برمجتها يتضح لنا أنها في أفضل حالاتها تحتاج لترتيب عدد (*s*) من العناصر إلى (*s-1*) من الخطوات وذلك عندما تكون المصفوفة مرتبة بالفعل، أما في أسوأ الحالات فإننا بحاجة إلى المرور على كافة العناصر بعدد *s<sup>2</sup>* وذلك على اعتبار أن كل عنصر يحتاج ترتيبه عدد جولات يساوي عدد العناصر - عدد العناصر التي تم ترتيبها (1 ثم 2 ثم 3 وهكذا).

**خوارزميات البحث والترتيب يمكن استخدامها لبيانات من أي نوع بما في ذلك الأساسية والمشتقَّة مع الأخذ بعين الاعتبار قواعد كل نوع من حيث المقارنة والتعامل المنطقى.**



### 8.5.2 خوارزمية الاختيار (Selection Sort Algorithm)

هذه الخوارزمية تعتمد على مبدأين هما اختيار الأنساب واستبدال القيم، حيث تبدأ من بداية المصفوفة ويتم مقارنة أول عنصر مع بقية عناصر المصفوفة، ثم يتم اختيار العنصر الأصغر واستبداله مع قيمة الخلية الأولى، وهكذا في الجولات الأخرى؛ في كل مرة تبدأ المقارنة من العنصر التالي ونختار الأنساب ونضعه في الخلية التي عليها الترتيب وهكذا نكرر هذه العملية بحيث يكون كل عنصر ما عدا الأخير هو بداية التكرار ومقارنه بالآخرين.

وبذلك فإن هذه الخوارزمية تبحث في كل جولة عن العنصر الأصغر؛ ففي الجولة الأولى تبحث عن العنصر الأصغر ويتم تخزينه من خلال الاستبدال في الخلية الأولى، ثم في الجولة الثانية تبحث عن ثاني أصغر عنصر ويتم تخزينه أيضاً من خلال الاستبدال في الخلية الثانية وهكذا.

من خلال هذه التفاصيل يتضح لنا أن هذه الخوارزمية بسيطة في برمجتها لكنها بطيئة في الإنجاز لأنها تحتاج إلى المرور على المصفوفة بمرات تكافئ مربع عدد العناصر؛ ولزيادة التوضيح، تابع معنا المثال 8.6 والذي يتبع هذه الخوارزمية بالخطوات والأرقام كمثال.

#### مثال توضيحي 8.6:

مستخدماً خوارزمية الاختيار، قم بترتيب المصفوفة التالية مستخدماً التتابع التوضيحي  
30، 60، 5، 7، 66، 15

**في الجولة الأولى:** تبدأ من الخلية الأولى يتم اختيار القيمة 5 كأصغر عنصر ويتم استبداله مع القيمة 30 بصفتها في الخلية الأولى

**5، 66، 30، 15**

**في الجولة الثانية:** تبدأ من الخلية الثانية ويتم اختيار القيمة 7 كثاني أصغر عنصر (أصغر قيمة في العناصر ما بعد الأولى) ويتم استبدالها مع القيمة 60 بصفتها في الخلية الثانية.

**5، 15، 60، 30، 66**

بذلك بعد الجولة الثانية نضمن ترتيب أول عنصرين، وهكذا في الجولات التالية.

**في الجولة الثالثة:** تبدأ من الخلية الثالثة ويتم اختيار القيمة 15 كثالث أصغر عنصر (أصغر قيمة في العناصر المتبقية ما بعد الثانية) ويتم استبدالها مع القيمة 30 بصفتها في الخلية الثالثة.

**5، 7، 15، 30، 60، 66**

وهكذا حتى نصل للجولة الأخيرة لتصبح فيها المصفوفة على النحو التالي:

**5، 7، 15، 30، 60، 66**

في المثال التوضيحي 8.7 القادم، سنعمل على تطبيق هذه الخوارزمية من خلال لغة جافا، وباستخدام المصفوفة ذاتها، وبذلك يصبح مبدأ عمل هذه الخوارزمية أكثر وضوحاً.

### مثال توضيحي 8.7

مستخدماً خوارزمية الاختيار، اكتب دالة لترتيب المصفوفة المستخدمة في مثال 8.6.

```

1 public void selectionSort(int num[])
2 {
3     int smallest; // لتخزين عنوان أصغر عنصر
4     for (int i = 0; i < num.length - 1; i++)
5     {
6         smallest = i; // عنوان أول عنصر فيما تبقى من المصفوفة
7         for (int index = i + 1; index < num.length; index++) // لإيجاد عنوان الأصغر
8             if (num[ index ] < num[ smallest ])
9                 smallest = index;
10    int temporary = num[ i ]; // استبدال قيمة الأصغر مع الخلية الحالية
11    num[ i ] = num[ smallest ];
12    num[ smallest ] = temporary;
13 }
14 }
```

وبدراسة هذه الخوارزمية وفهم برمجتها يتضح لنا أنها في أفضل حالاتها تحتاج لترتيب عدد ( $s$ ) من العناصر إلى ( $s-1$ ) من الخطوات في الدوران الداخلي ومتلهم في الخارجي وهذا يعني أن المصفوفة التي تضم 20 عنصراً تحتاج في أسوأ الحالات إلى (361) جولة للترتيب أي  $s^2$  وهذا مماثل تقريباً لخوارزمية الفقاعات الهوائية.

### 8.6 أمثلة وتدريبات عامة

#### 1. استخدم الكلمات المناسبة لتعبئة الفراغات في الجمل التالية:

- أ- تطبيقات البحث كتطبيق محوسب متعدد ومنها \_\_\_\_\_ و \_\_\_\_\_.
- ب- خوارزمية البحث الثنائي يتشرط في تطبيقها بنجاح أن تكون البيانات \_\_\_\_\_.
- ت- يعتبر استخدام خوارزمية البحث الخطى مناسبة للبيانات \_\_\_\_\_ بينما استخدام خوارزمية البحث الثنائي مناسبة للبيانات \_\_\_\_\_.
- ث- يمكن استخدام خوارزميات الترتيب للأنواع هو \_\_\_\_\_ بينما النوع *int* هو \_\_\_\_\_.
- ج- من خوارزميات الترتيب \_\_\_\_\_ و \_\_\_\_\_.
- ح- تتعدد خوارزميات البحث منها \_\_\_\_\_ و \_\_\_\_\_.
- خ- عند جعل التكرار الخارجي في خوارزمية الاختيار للترتيب يتكرار بعدد عناصر المصفوفة ينتج عن ذلك اسمه \_\_\_\_\_.

#### 2. حدد صحة أو خطأ كلا من العبارات التالية مع التعليل:

- ( ) أ- يمكن استخدام خوارزميات الترتيب لكافة الأنواع باستثناء الأصناف.
- ( ) ب- خوارزميات البحث يمكن تطبيقها فقط على الأنواع الأساسية لسهولة المقارنة.
- ( ) ت- لا يمكن اجتماع خوارزمية البحث الثنائي مع خوارزمية الاختيار للترتيب في تطبيق واحد.
- ( ) ث- عنوان العنصر الأوسط للمصفوفة يساوي قيمة عنوان العنصر الأخير على 2.
- ( ) ج- من المؤسسات التي تحتاج مفهوم الترتيب في تطبيقاتها البرمجية مركز الحاسوب الحكومي.

### 8.7 تمارين ذاتية الحل

- #### 1. أحد مؤسسات التعليم العالي في فلسطين، لديها تطبيق برمجي مخزن فيه بيانات الموظفين على أنّ الموظف يعتبر صنف وبه معلومات (الاسم الأول، الاسم الأخير، الرقم الوظيفي، الراتب الشهري، رقم المحمول)، اكتب برنامج تستخدم فيه خوارزمية الاختيار لترتيب معلومات الموظفين تصاعدياً على حسب الرقم الوظيفي الممثل في 1 بait فقط.

2. مستقىداً من السؤال رقم 1 ، استخدام خوارزمية البحث الثنائي للبحث عن بيانات الموظف من خلال رقمه الوظيفي.
3. على نمط الشكل 8.2، قم بالبحث عن الرقم 15 في المصفوفة التالية (5، 9، 78، 90، 91، 94، 102) المرتبة ترتيباً تصاعدياً.
4. مستقىداً من السؤال رقم 3، اكتب برنامج جافا للبحث عن رقم يدخله المستخدم داخل المصفوفة المذكورة.
5. بالنمط المستخدم في الشكل 8.4 كاملاً، قم بترتيب المصفوفة التالية ترتيباً أبجدياً تصاعدياً حتى يصبح يزن قبل أحمد مع ترتيب ما بينهما:
- علي ، محمد ، خالد ، حسام ، أحمد ، علاء ، يزيد ، يزن
6. اكتب برنامج جافا لتطبيق الحل الخاص بالسؤال رقم 5.
7. مستقىداً من السؤالين 5 و6، اكتب برنامج للبحث عن الأسماء في المصفوفة المذكورة في السؤال رقم 5.
8. اكتب برنامج يستقبل من المستخدم أرقاماً صحيحاً على أن يتم تخزين الأرقام المدخلة في المصفوفة دون أن يتم تخزين رقمين متساوين، على أن يتوقف استقبال الأرقام من المستخدم بمجرد امتلاء المصفوفة.
9. إذا كانت المصفوفة مرتبة بالفعل، كم مقارنة ستقوم خوارزمية الفقاعات الهوائية بتنفيذها؟
10. يتوفر لديك أربع مصفوفات على النحو التالي:

- ✓ الأولى لتخزين أرقام الموظفين وهي من النوع الصحيح (*int*)
- ✓ الثانية لتخزين أسماء الموظفين وهي من النوع النصي (*String*)
- ✓ الثالثة لتخزين حالة الموظف (حضور، غياب) وهي من حالة منطقية (*boolean*)

إذا طلب منك ترتيب هذه المصفوفات، بأي المصفوفات تبدأ وأي الخوارزميات تختار؟

**اجابات التدريبات العامة:**

1. (أ) المستشفىات ، الوزارات (ب) مرتبة (ت) القليلة و الغير مرتبة، الضخمة والمرتبة (ث) الاختيار، الدمج (ج) الخطى ، الثنائي (ح) خطأ استثنائي، *ArrayIndexOutOfBoundsException*

2. حدد صحة أو خطأ كلا من العبارات التالية مع التعليل:

- أ- (خاطئة) بلا استثناء فالأصناف مكونة من أنواع أساسية يمكن الترتيب بالاعتماد عليها.
- ب- (خاطئة) وكذلك الأنواع المشتقة لأنه يتم الاعتماد على الأنواع الأساسية التي داخل المشتقة للبحث.
- ت- (خاطئة) بل يمكن لأن خوارزمية البحث الثاني تحتاج إلى بيانات مرتبة قد تتجهها خوارزمية الاختيار.
- ث- (خاطئة) نحصل عليه من خلال قسمة (الحد الأدنى + الحد الأعلى) على 2، علما بأن الحد الأدنى في البداية = 0 بينما الحد الأعلى = حجم المصفوفة - 1
- ج- (صحيحة)

انتهى الباب

محاولات الإنسان في سبيله لمحاكاة  
خلق الله تعالى لم ولن تنتقطع ليزداد  
في كل يوم إيماناً بقدرة الله تعالى  
على الخلق في أحسن صوره.

## الباب التاسع

### أساسيات البرمجة شيئاً فشيئاً التوجه

#### (Object-Oriented Programming)

يتوقع من الطالب في نهاية هذا الباب أن:

- يدرك مفهوم البرمجة شيئاً فشيئاً التوجه.
- يقدر الفائدة من البرمجة شيئاً فشيئاً التوجه.
- يشرح مفهوم الأصناف والكائنات.
- يدرك المقصود بمحددات الوصول.
- يتفق إنشاء ومعالجة مصفوفة من الكائنات.
- يفرق بين محددات الوصول المختلفة.
- يشرح مفهوم الوراثة في البرمجة شيئاً فشيئاً التوجه.
- يميز بين التوارث الفردي والثاني.
- ضوابط توارث الصفات والدواں
- يستخدم كلمة *extends* لتطبيق الوراثة.
- يشرح مفهوم تعدد الأشكال.
- يميز بين الأصناف المجردة (*abstract*) والأصناف المحددة (*concrete*).
- يستخدم مفهوم (*Overridden Method*) لتطبيق مفهوم تعدد الأشكال.

يتوفر لهذا الباب شرائح العرض (PowerPoint) وكذلك ملفات مرئية من إعداد مؤلف الكتاب؛ يمكنك الحصول عليها من خلال الموقع الأكاديمي للمؤلف [mfarra.cst.ps](http://mfarra.cst.ps)



## الباب التاسع: أساسيات البرمجة شبهية التوجه

### 9.1 مقدمة

تعتبر لغة جافا من لغات البرمجة التي تعتمد في عملها وتنظيمها على أسلوب برمجة يُعرف باسم البرمجة الشبيهة أو الهدافية أو شبيهة التوجه (*Object-Oriented Programming*) ويختصر إلى *OOP* وجميع هذه التسميات نظراً لكونها تتعامل مع أي مشكلة برمجية على أنها مكونة من مجموعة كائنات أو أشياء.

في الفصل 9.2 سنتناقش هذا المفهوم ثم نتعرف على مميزات هذا الأسلوب في الفصل 9.3 بينما سنتناقش مفهوم الأصناف والكائنات والعلاقة بينهما وطرق تطبيقهم في لغة جافا في الفصل 9.4 حيث تعتبر هي العناصر الرئيسية لهذا الأسلوب. في الفصل 9.5 نستعرض مفهوم محددات الوصول (*Access Modifiers*) لما له من أهمية في فرض قواعد الأمان للتطبيقات. بعد ذلك سنجمل هذه المفاهيم وتطبيقها من خلال مثال توضيحي كامل في الفصل 9.6 لتصبح هذه المفاهيم أكثر وضوحاً. في الفصل 9.7 نعود للحديث عن المصفوفات، ولكن هذه المرة الحديث عن مصفوفة الكائنات، حيث تضم الخلية الواحد أكثر من نوع من البيانات على هيئة ما يعرف بالكائن.

وفي نهاية الباب، في الفصول 9.8 و 9.9 سنتناول بشيء من التفصيل مفهوم الوراثة وتعدد الأشكال وهي من المفاهيم المشهور استخدامها في التطبيقات المتقدمة من البرمجة شبيهة التوجه، كما أنّ بُنية لغة جافا قائمة عليها بشكل واضح.

### 9.2 مفهوم البرمجة شبيهة التوجه

شهدت تقنيات وأساليب البرمجة منذ بدايات ظهورها تطوراً واضحاً سعى من خلاله المبرمجين إلى زيادة الإنتاج وتسهيل عمليات حل المشاكل البرمجية وتقليل معدل التعقيد، فعلى مستوى لغات البرمجة عالية المستوى انتقلنا من البرمجة الخطية التي يتم فيها برمجة كل خطوة بمفردها وتكرار كتابتها بالعدد المطلوب ثم الوصول لمفهوم جمل التحكم في سير العمليات، من خلال جمل الدوران والاختيار، ثم انتقلنا إلى نمط برمجي جديد يُعرف باسم البرمجة الإجرائية (*Procedural Programming*) حيث تم تقسيم البرامج والتطبيقات إلى مجموعة من الدوال أو الإجراءات التي يمكن استدعاءها والاستفادة منها وقت الحاجة.

كل هذا التطوير لم يكن كافياً للوصول إلى المستوى المُرضي لكفاءة البرمجيات، من هنا ظهر مفهوم البرمجة شبيهة التوجه حيث يتم تطوير الحل لأي مشكلة برمجية على أنها مكونة من مجموعة من الكائنات (*Objects*) أو (الأشياء) وهذه الكائنات تتفاعل مع بعضها لمعالجة البيانات وهذه الكائنات مُشتركة من أصناف عامة.

ولكي نصل لفهم أوضح، انظر الآن حولك في الشارع، فماذا ترى؟ نعم، ترى سيارات، أشخاص، معارض تجارية وأشجار. ولكن هل تهتم كيف تعمل السيارات؟ أو كيف تحدث الحركة عند الأشخاص؟ وما هي الموصفات الدقيقة لهذه الأشياء؟ بالتأكيد لا، بل إننا نستخدم ما حولنا من أشياء دون معرفة كيف تفاصيل عملها من الداخل، فقط يهمنا معرفة كيف لنا أن نستخدمها وهذا هو مفهوم البرمجة شيئاً فشيئاً التوجه.

فالسيارات والأشخاص وغيرهم في الحقيقة كائنات مشتقة من صنف عام اسمه سيارة أو شخص، وكل سيارة عند التدقيق فيها نجد أن موصافاتهم واحدة وتخالف فقط قيمة هذه الموصفات، فمثلاً جميع السيارات لها لون واسم مالك وعام تصنيع ونوع وتخالف هذه السيارات في قيمة هذه الصفات فقط.

هذا النمط الجديد من البرمجة يعمل وفق طريقة "فرق تسد" (*Divide and Conquer*) التي تعرضنا لها بالشرح في الباب السابع عند شرح الدوال، وكيف أن التقسيم يساعد بشكل كبير على تبسيط وتسهيل الحل برمجياً.

ولعل من أبرز الأمثلة التي يمكننا أن ندرجها هنا، الطفل الصغير الذي يلعب برجل آلي، فيتحرك الرجل ويصدر صوتاً بصور مختلفة، والطفل لا يعلم ما الذي يحدث داخل الرجل الآلي ولكنه يعرف كيف يجعله يتحرك أو يُصدر صوتاً من خلال الضغط على زر أو مفتاح محدد.

إذن فالقصد والغاية من البرمجة شيئاً فشيئاً التوجه أن يتم تقسيم المشاكل الكبيرة إلى أجزاء صغيرة تتحاطب وتتفاعل مع بعضها البعض دون معرفة التفاصيل الصغيرة والدقيقة لكيفية إنجاز كل جزء لعمله وهذا يتحقق البساطة.

ولعل هناك تعرضاً مختصراً للبرمجة شيئاً فشيئاً التوجه بأنها تقسيم المشكلة إلى مجموعة من الأشياء وأن كل شيء مكون من صفات وسلوك يتم محاكاتهم في البرمجة من خلال المتغيرات والدوال. كما أن الأشياء يتم تمثيلها في وحدة عامة تُعرف باسم الصنف (*Class*) ويُعرف في داخله المتغيرات كمحاكاة للصفات ويُعرف بداخله الدوال كمحاكاة للسلوك.

بعض المصنفات تطلق مفهوم البرمجة الهدافية لأنها تعتبر هذا الأسلوب قائم على محاكاة الكائنات أو الأهداف (*Objects*).



مصنفات أخرى تطلق اسم البرمجة الشينية على اعتبار تقسيم المشكلة إلى مجموعة أشياء وكل شيء يتم تمثيله في البرمجة على أنه صنف (*Class*)



والآن بعد هذه المقدمة المختصرة للبرمجة شيئاً فشيئاً التوجه، لعلك تستطيع الاستبطاط أن هناك أربعة مفاهيم أساسية يتم من خلالها الوصول لمفهوم البرمجة شيئاً فشيئاً التوجه وهي موضحة بالشكل 9.1 وسنسرحها فيما يلي.



شكل 9.1: المفاهيم الأساسية للبرمجة شبيهة التوجه

. ١. **التغليف (Encapsulation)**: وهو عملية وضع الخصائص والسلوك لكل عنصر من عناصر المشكلة ضمن وحدة واحدة تسمى الصنف (Class) ويمكن بذلك إخفاء البيانات عن الآخرين ويُسمح لهم بالوصول لها بصلاحيات معينة تحددها محددات الوصول التي سيتم مناقشتها في الفصل 9.5 وفي هذه الحالة فإن الدوال تستخدم للتفاعل مع المتغيرات للتغيير والطباعة ونحو ذلك. ولذلك يُسمى هذا المفهوم أيضاً إخفاء البيانات (Data Hiding). الشكل 9.2 يختصر هذا المفهوم الذي يعتبر محاكاة للواقع حيث أن كل شيء من الأشياء التي حولنا لا تعتبر بيانته واستخدامها متاحاً للجميع بل هو متاح للبعض ومحفي عن البعض.

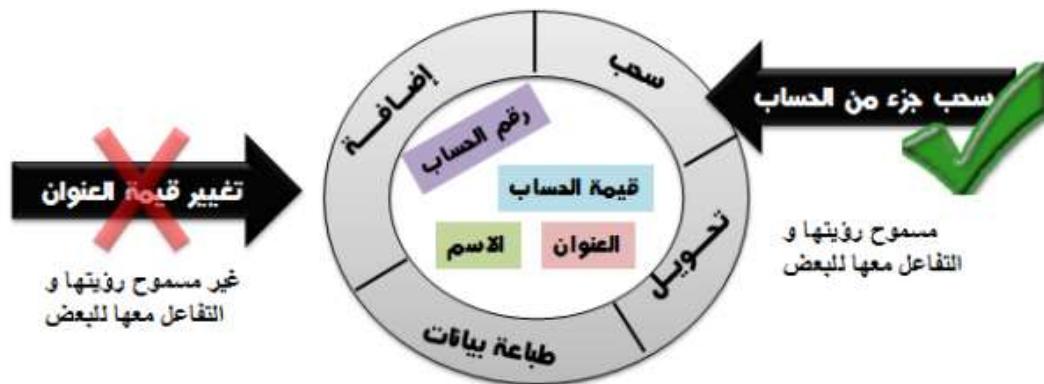
فعلي سبيل المثال الحساب البنكي لعميل ما من خلال الصرف الآلي، لا يمكنه تعديل رصيده إلا من خلال عمليات السحب والإيداع، بينما لا يمكنه مباشرة تغيير رصيده بالزيادة أو النقصان. ولذلك تسمى الأصناف أنواع البيانات المجردة (Abstract data type - ADT). هذا المفهوم سنتعامل معه في كافة الأمثلة القادمة في هذا الباب.

**الأصناف في البرمجة تخفي بياناتها عن الآخرين وهذا ما يعرف باسم التغليف.**



**التغليف** هو أن يتم تغليف البيانات وإخفاء التطبيق عن المستخدم للأصناف.





شكل 9.2: مفهوم التغليف

2. **مفهوم التجريد (Abstraction):** وهو التعامل مع عناصر المشكلة بتجدد أي دون الاهتمام بكيفية تنفيذ كل عنصر لعمله، وما هي الخطوات الداخلية التي ينفذها أي عنصر لإخراج النتائج، فما يهمنا فقط كيف نستخدم كل عنصر، ومثال ذلك هل يهم المستخدم للة الحاسبة كيف تقوم بحساب المعادلة وكيف تعطي الأولويات وكيف تحول الأرقام وتقيمها أم يهمه فقط أنها كيف تُستخدم؟، كذلك السيارة لا يهمني معرفة كيف تعمل من الداخل، كيف يتحرك الوقود في مولد السيارة؟، كيف يتم حرق الوقود؟ كيف تتحرك التروس عندما نطلب منها العودة للخلف السير للأمام؟، بل فقط ما يهمنا كيفية نستخدم السيارة ونستفيد منها بغض النظر عن تقييدات التشغيل. هذا يُعرف بمفهوم التجريد والهدف منه الوصول لقدر كبير من تسهيل العمل والإنتاج، ويظهر هذا بشكل كبير أثناء استخدامنا للصنف (*JOptionPane*)، فنحن نستخدم الدوال التي بداخله ونطبع البيانات في صناديق النص دون أن نعرف كيف تم تطويرها وكيف يتم عملها من الداخل. وبعد مفهوم التغليف والتجريد وجهاً لعملة واحدة، حيث أنهما مترابطان، لا يحضر أحدهما إلا بحضور الآخر.

**التجريد** هو الفصل بين تفاصيل كيفية التطبيق عن تفاصيل كيفية الاستخدام للسلوك.



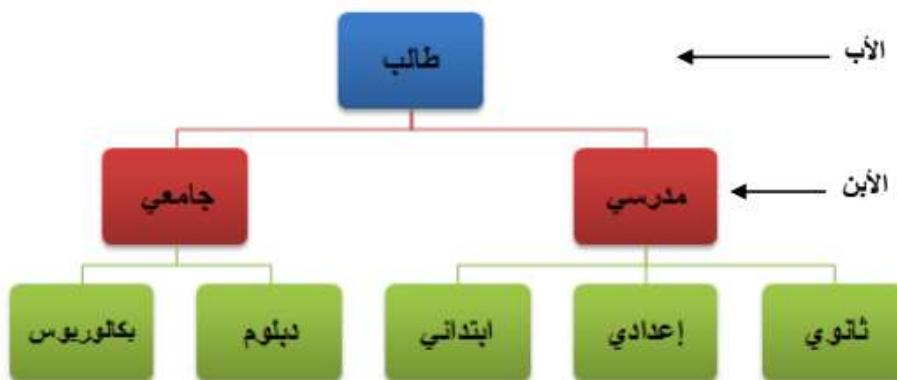
**التجريد** هو عملية إخفاء طريقة تطبيق السلوك أو العمل داخل الصنف عن الآخرين.



3. **مفهوم التوارث (Inheritance):** هو مفهوم مشتق من علم الوراثة في الكائنات الحية، حيث أن الابن يرث صفات ومهارات معينة من أبيه، بالإضافة لوجود الفرصة لامتلاكه صفات ومهارات أخرى لا يمتلكها والديه. ففي البرمجة شيئاً فشيئاً التوجه يمكننا جعل أحد الأصناف كأب من خلال ربطه بصنف أو مجموعة من

الأصناف كأبناء، وتصبح بعض أو كل صفات الأب موجودة في الابن مع زيادات أخرى من صفات سلوك للابن.

ومن الأمثلة التي يمكن استخدام الوراثة فيها ما تشاهد في الشكل 9.3، أن هناك صفات سلوك عام بين كل الطلبة من مختلف الأنواع، إلا أن طالب الثانوية لها صفات إضافية تختلف قليلاً عن صفات طالب الابتدائي، وهذا يوفر الكثير من الوقت في بناء التطبيقات من الصفر ويزيد من الانتاج.



شكل 9.3: مفهوم الوراثة في البرمجة شينية التوجه

**الوراثة هي القدرة على إنشاء أصناف جديدة من أصناف موجودة بالفعل.**



هذا المفهوم سُئِرَدَ له الفصل 9.8 للحديث عنه بشيء من التفصيل مع ذكر مثال مختصر يوضح الفكرة، علماً بأن مفاهيم البرمجة شينية التوجه تحتاج لمساق خاص بها للتدريس.

4. **مفهوم تعدد الأشكال (Polymorphism):** هو إعطاء القدرة للكائن الابن أن يشير لكائنات الأب، وهذا ما يُساهم في البرمجة المرنّة، حيث تعطي الفرصة لبرمجة الأصناف بشكل عام وتتيح الفرصة للتخصيص بعد ذلك. هذا يتطلب مثلاً تطوير مجموعة من الدوال في الآباء والأبناء بشكل مختلف وذلك لمحاكاة الواقع حيث أننا نقوم بحساب مساحة الأشكال الهندسية ولكن كل مساحة يتم حسابها بشكل مختلف، أي القدرة على توفير أكثر من تطبيق لسلوك واحد. ومثال ذلك في الحياة الواقعية أن كل الكائنات تتحدث إلا أنها تتحدث بطريق مختلفة، هذا المفهوم سيتم التطرق له بشيء من التفصيل في الفصل 9.9.

**تعدد الأشكال تعني أنَّ الكائنات المشتقة من الأبناء يمكن أن تشير إلى كائنات الآباء.**





شكل 9.4: مفهوم تعدد الأشكال في البرمجة شبيهة التوجه

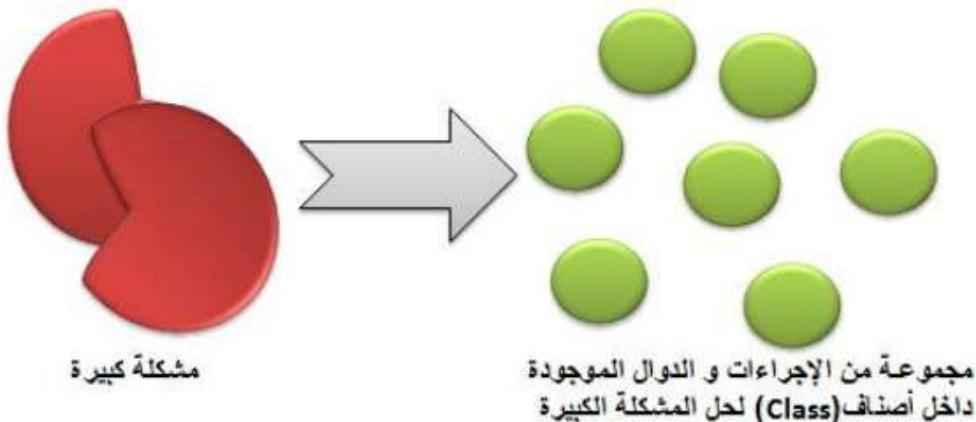
ويقصد من استخدام هذا المفاهيم الأربع:

- ✓ زيادة الإنتاج من خلال توفير الوقت والجهود اللازم لبناء العديد من التطبيقات.
- ✓ رفع درجة المحاكاة مع الواقع.
- ✓ تبسيط استخدام البرمجة.

### 9.3 مميزات البرمجة شبيهة التوجه

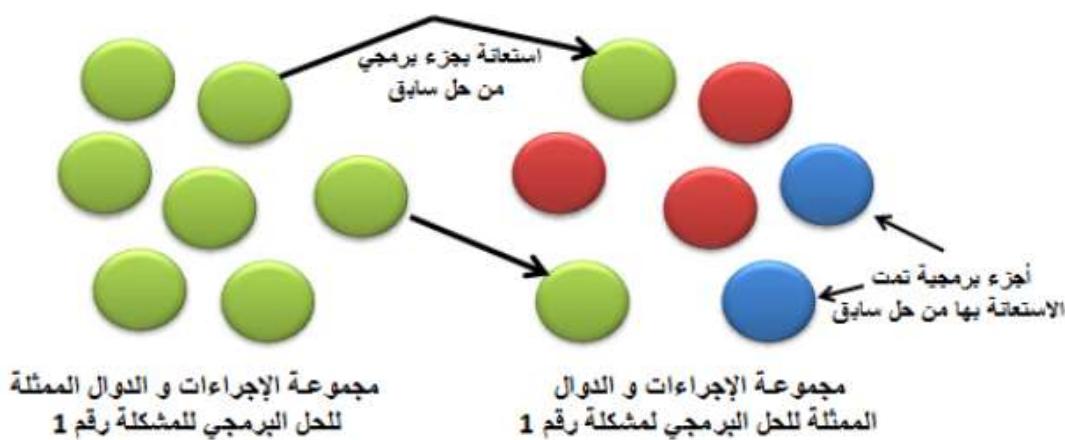
لائلاً أن التطور الذي تشهده لغات البرمجة يوماً بعد يوم لا يكون من فراغ بل بحثاً عن مزيد من المميزات التي تتعلق بالسهولة في التعامل والسرعة في الإنتاج والقليل من الوقت، وهذا ما أحدثه البرمجة الشبيهة من خلال عدد من المميزات الجديدة لها، إلا أنَّ هذا النمط من البرمجة وهذه المميزات لا يمكن ملاحظتها إلا من خلال بناء تطبيقات برمجية كبيرة ومعقدة، وهذا هو حال البرمجيات التي يتم بناءها وتطويرها للمؤسسات الكبيرة، أما البرامج التي سبق كتابتها في الأبواب السابقة فهي تعرف على أنها البرامج المستخدمة في التعليم لشرح الأفكار والمعلومات. وتتميز البرمجة شبيهة التوجه بعدد من المميزات، اخترت أكثرها عدداً منها وهي:

1. **التجزئة (Modularization):** ويقصد به تجزئة المشكلة الكبيرة إلى مشاكل صغيرة يمكن حلها بشكل منفصل وبسهولة أكبر، ثم يتم تجميع هذه الحلول الصغيرة للوصول لحل المشكلة الكبيرة وقد ناقشنا هذا المفهوم عند الحديث عن الدوال. الشكل 9.5 يوضح لهذه الميزة.



شكل 9.5: توضيح لميزة التجزئة (Modularization)

2. إعادة الترکيب (Composability): حيث نتمكن من استخدام الأصناف التي تم بناءها سابقاً في إنتاج تطبيقات جديدة مختلفة من خلال إعادة تركيبها وتشكيلها حسب حاجة التطبيق الجديد. الميزة الأولى تساهُم في الوصول للميزة الثانية حيث يمكن استخدام أجزاء من مشكلة في حل مشكلة أخرى مما يساهُم في الوصول للمميزات الأساسية التي نسعى لها. ولعل استخدامك للدوال الخاصة بطباعة المخرجات في صناديق حوار من الصنف ( JOptionPane ) ليدل على تطبيق هذا المفهوم، حيث أن هذا الصنف يعتبر جزء من تطبيق كبير تمت تجزئته لحل مشكلة أساسية ثم تم استخدامه لحل مشاكل أخرى، الشكل 9.6 يقدم توضيحاً تقريرياً لهذه الميزة.



شكل 9.6: من مميزات البرمجة شينية التوجّه مفهوم إعادة الترکيب

3. بالإضافة إلى ما سبق يعتبر مفهومي التجريد والتغليف من أبرز المميزات مما لها من أثر بالغ في تسريع وتسهيل العملية الإنتاجية وقد سبق التعرض لها بالشرح في الفصل الماضي.

**التطوير الذي تشهده لغات البرمجة يهدف إلى مزيد من الانتاج من خلال تقليل الوقت المستخدم في بناء أجزاء يعتمد بناءها كما يهدف إلى زيادة سهولة التعامل معها وفهمها وزيادة قدرتها على محاكاة الواقع، وهذا ما يقترب منه أسلوب البرمجة شيئاً فشيئاً التوجه.**



في ختام هذا الفصل من المهم أن نعرض الفارق بين هذا النمط من البرمجة وغيرها من الأنماط وهي:

1. **البرمجة الخطية:** تعتمد على برمجة كافة الأهداف في أسطر متتالية وعند الحاجة لنكرار أي أمر فإنه يتوجب علينا إعادة كتابتها وهي وبالتالي تحتاج إلى وقتٍ طويٍ وجهدٍ كبيرٍ.
2. **البرمجة الإجرائية:** تعتمد على كتابة الدوال والإجراءات وعند الحاجة إليها يتم استدعاءها والاستفادة منها، وهي أفضل حالاً من سابقتها، إلا أنها لم تصل بعد للمستوى المناسب من محاكاة الواقع أو من السرعة الكافية في إنجاز المهام خاصة التطبيقات الكبيرة.
3. **البرمجة شيئاً فشيئاً التوجه:** تعتمد على محاكاة الأشياء من الواقع لعالم البرمجة وتعتبر كل ما حولنا عبارة عن كائن من صنف ما وهذا الصنف مكون من مجموعة صفات يتفاعل ويُعالجها سلوك ما.

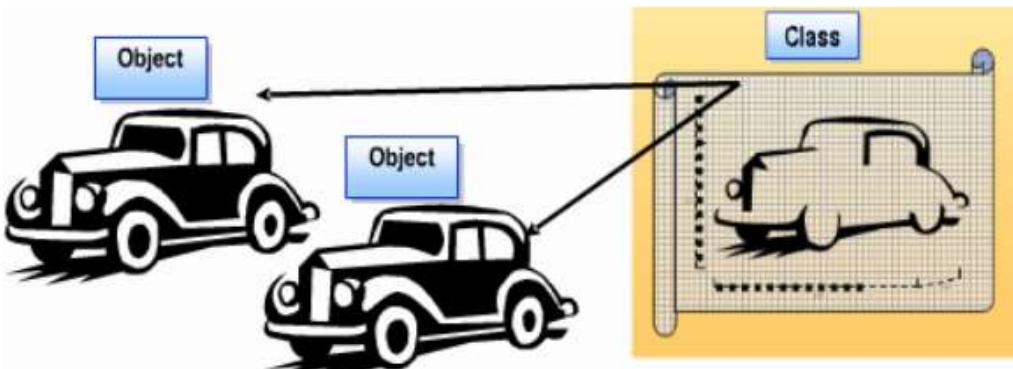
#### 9.4 مفهوم الأصناف والكائنات (Classes & Objects)

تعتبر الأصناف والكائنات هي المكون الرئيسي لتطبيق أسلوب البرمجة شيئاً فشيئاً التوجه وبقدر اتقان الطالب للتعامل معهما يمكن من إتقان هذا الأسلوب من البرمجة، ومن خلال مكوناتهما تظهر مكونات ومميزات البرمجة شيئاً فشيئاً التوجه، في هذا الفصل سنلقي الضوء عليهما، ونهدف من خلال هذا الفصل أن يصبح الدارس قادرًا على فهم مكونات الأصناف والكائنات فيما دقيقاً وواضحاً وأن يصبح قادرًا على بناء الأصناف الأساسية للمشاكل البرمجية المختلفة.

##### 9.4.1 المفهوم العام والفارق بينهما:

ويقصد بالأصناف هي الفئات من حولنا دون الدخول في تفاصيل دقيقة، فعندما سألك في الفصل 9.1 ماذا ترى حولك في الشارع قلت لي: (أشخاص، سيارات، معارض تجارية، أشجار....) فهذه الأشياء تعتبر فئات أو أصناف (Class) دون ذكر تفاصيل دقيقة تجعلني أضع يدي على شيء حقيقي وواقعي على أرض الواقع، وبالتالي هي أقرب إلى خارطة بناء البيت فهي فقط تحتوي على عالم عامّة ذات تفاصيل واقعية واضحة. كما أنها أقرب إلى نموذج ملف عرض تقديمي (Power Point) قبل أن يتدخل المستخدم بتغيير أي صفة من صفاتها، ستجد أنه ملف له صفات عامة دون أن تعرف هذا الملف مخصص لمن.

بهذا المفهوم فإننا نعتبر الشخص، البيت، السيارة، الأشجار، وغيرهم تعتبر أصنافاً فهي لا تدلنا على كائن حقيقي يمكن لنا التعرف عليه ومخاطبته ومعرفة قيم صفاته بالضبط. بينما إذا طلبت منك أن تصف لي شخصاً ما من رأيهم في الشارع، فقلت لي طوله 170 سم، وزنه تقريباً 80 كيلو جرام، لونه قمحي، يرتدي نظارات، شعره أسود، عمره 29 عاماً، رقم بطاقة الوطنية 907675642 بهذا التحديد أصبح بإمكانني معرفة شخص محدد تماماً أميزة عن الأشخاص الآخرين وينتمي للصنف الذي اسمه أشخاص. هذا ما يعرف باسم الكائن (Object). فالكائنات تُشتق من الأصناف من خلال تحديد صفاتها بقيم محددة وكذلك سلوكها، ويظهر ذلك في الشكل 9.7 حيث يظهر لك مخطط لصنف سيارة (Car Class) دون مواصفات تدللك عليها في الواقع، بينما على أرض الواقع أصبحت أكثر وضوحاً من خلال إعطائها قيم لمواصفاتها من لون وهيئة فأصبحت كائناً محدداً (Car Object).

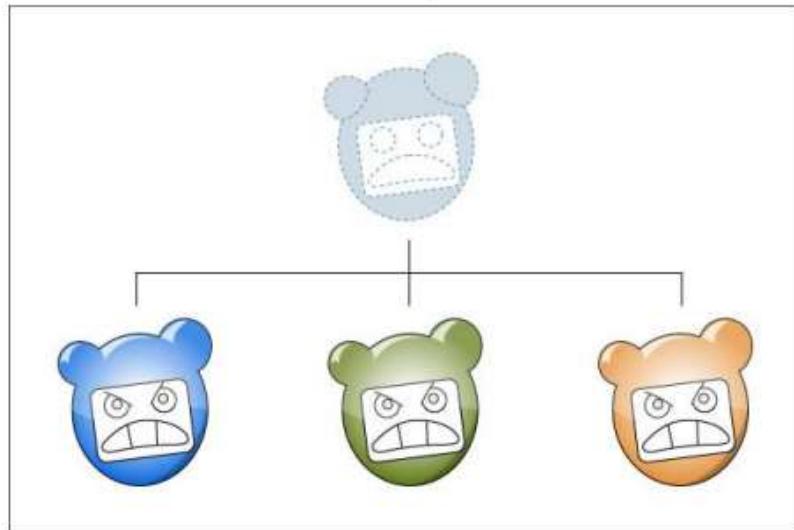


شكل 9.7: مفهوم الأصناف والكائنات من خلال مثال السيارات

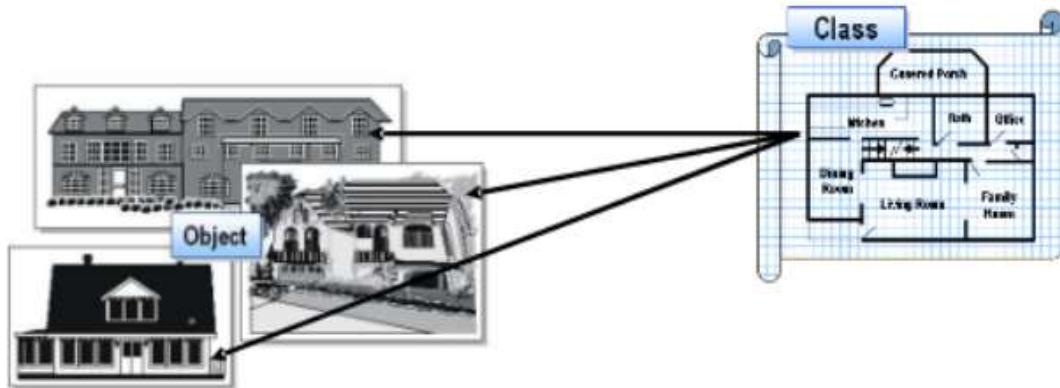
الشكل 9.8 يُظهر لك صنف الوجه الغاضبة (Angry Face Class) بينما بمجرد اشتقاء نسخة منه وإعطاءها قيم لصفتها أصبحت كائناً (Angry Face Object) مميّزاً لوجه له صفات محددة يمكن تمييز الوجه بها. بينما في الشكل 9.9، يُوضح لك مفهوم الصنف لبيت وهو عبارة عن مخطط لبيت لا تميز القيم الخاصة بصفاته ولكنه يحتوي على المواصفات العامة (Home Class) وبالتالي لا يمكن التعرف عليه على أرض الواقع، بينما بمجرد إعطاء قيم لمواصفاته يظهر لنا بيئاً حقيقياً (Home Object) يمكن تمييزه.

**الأصناف** تقوم بتعريف الصفات والسلوك الخاص بالكائنات.





شكل 9.8: مفهوم الأصناف والكائنات من خلال مثال الوجوه الغاضبة

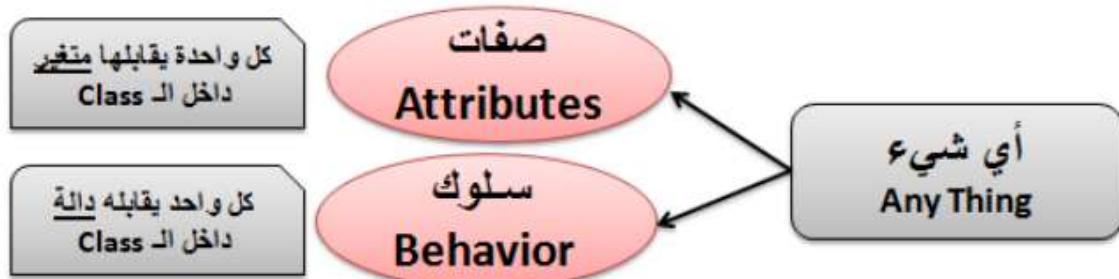


شكل 9.9: مفهوم الأصناف والكائنات من خلال مثال البيوت

بهذا أصبح واضحًا لنا الفارق بين الصنف والكائن وأصبح واضحًا أن الكائن بعد أن يتم استئصاله من الصنف يمكننا التعامل معه والتعديل على قيم صفاتيه بينما الصنف مجرد تحطيط يوضح الصفات والسلوك بشكل عام دون قيم محددة.

#### 9.4.2 كيف نطور الأصناف برمجيًا؟

تعرفنا بالفعل على مفهوم الأصناف، والآن كيف لنا أن نطبق هذا المفهوم إلى لغة الجافا بالفعل، دائمًا عندما تريد تطوير صنفًا برمجيًا جديداً، ابدأ بالتفكير في هذا الأمر من منطلق الشكل التوضيحي 9.10، أن أي شيء (عنصر) من حولنا عبارة عن مجموعة من الصفات والسلوك ويتم تحويلها إلى ما يقابلها برمجيًا متغيرات ودوال على التوالي.

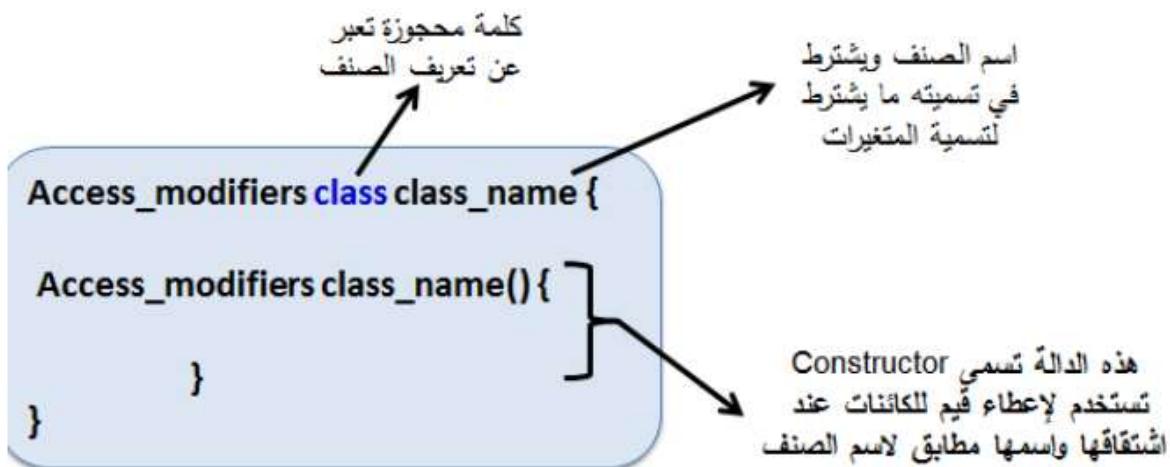


شكل 9.10: مبدأ التكير لتطوير صنفاً برمجياً

الصنف الجديد يعامل معاملة الأنواع المشتقة في لغة البرمجة وبالتالي يمكنك تعريف متغيرات أو كائنات (*Objects*) منه.



فالصنف في البرمجة له تركيبة محددة موضحة في الشكل 9.11، فتعريفه كما تلاحظ يبدأ بأحد محددات الوصول ثم الكلمة الممحوزة (*class*) التي توضح أن القطعة التالية الكاملة هي صنف برمجي، ثم اسم الصنف وهو اسم تعريف يخضع لكافة قواعد الأسماء التعريفية، يلي ذلك أقواس { }، أما جسم الصنف فيحتوي على تعريف لكافة المتغيرات (المتغير يقابل صفة) والدوال اللازمة للصنف (الدالة تقابل سلوك) كما يحتوي على الدالة البنائية (*Constructor*) وهي دالة من نوع خاص مسؤولة عن إعطاء قيم بدائية لصفات الكائن الجديد إذا لم يحدد المستخدم قيمًا من عنده ووجودها مرغوب فيه بينما ليس إجبارياً.



شكل 9.11: تركيبة الأصناف في لغة جافا

فالآن يمكنك البدء في التكير لتطوير أول صنف برمجي لك، ولتكن مثلاً للطاولة التي تذكرة عليها الآن، فنبداً حسب الشكل 9.10، ما صفاتها؟ طول، نصف قطر، ارتفاع، لون، نوع الخشب... كل واحدة من هذه الصفات

بالتأكيد يمكن تحويله لمتغير، فمثلاً الطول يعتبر متغير من نوع *Length* واسمه *float* أما نوع الخشب فيعتبر *Material* واسمه *String*.

ثم ما السلوك الذي تقوم به الطاولة؟ فقط طباعة بياناتها وبالتالي تقوم بتعريف دالة لها اسمها *PrintData* (التعرف على الدوال والتعامل معها)، تم مناقشته بتوسيع كامل من خلال الباب السابع، بعض الأصناف يكون لها سلوك متعدد مثل تغيير قيمة، سحب من الرصيد، إضافة للرصيد، تحويل مبالغ، حركة للإمام وغير ذلك.

### مثال توضيحي 9.1:

طور صنفنا برمجيًا مبسطًا للصنف طاولة.

```

1 public class Table {
2     float length;
3     String material;
4     float radius;
5     public Table() {
6         length = 5.1;
7         material = 'mdf';
8         radius = 1.2;
9     }
10 }
```

كما تلاحظ في السطر رقم 1، تم تعريف الصنف بإعطاءه اسم (*Table*) ومن المتعارف عليه أن يكون الحرف الأول من اسم الصنف بحالة كبيرة. في السطر رقم 2 إلى 4 تم تعريف المتغيرات، ثم في السطر رقم 5 تم تعريف الدالة البنائية، ويكون تعريفها من:

1. محدد وصول دائمًا من النوع *public* ثم
2. اسم الدالة ولا بد أن يكون اسمها مطابق لاسم الصنف ثم

3. أقواس القيم الممرة والافتراضي تكون الأقواس فيه فارغة بلا قيمة بينما عند تعريف أكثر من دالة بنائية يتم تمرير قيمة لها.

4. أما جسم الدالة فتضمن المتغيرات الخاصة بالصنف مع إعطاء كل متغير قيمة بدائية سيتم إعطاؤها للإثباتات التي سيتم اشتغالها من هذا الصنف دون أن يحدد المستخدم فيما محددة يرغب فيها.

5. يمكن أن يحتوي الصنف البرمجي على أكثر من دالة بنائية، فمثلاً قد يتم تعريف دالة بنائية أخرى تسمح للمستخدم بإعطاء قيمة للإثباتات عند اشتغالها، وهذا تطبيقاً لمفهوم الحمل الزائد للدواوين (*Overloading*) وقد تم التطرق لهذا المفهوم في الباب السابع.

الآن استطعت أن تطور صنفاً برمجياً ولكنه كما تعلم لا يمكننا التفاعل معه أو التعديل على قيم متغيراته أو استخدام الدواوين إلا من خلال اشتغال كائن (*Object*) منه، وهذا ما سنعرف عليه فيما يلي.

#### 9.4.3 كيف نشتق إثباتات من أصناف موجودة؟

اشتقاق الإثباتات من أصناف موجودة يتم من خلال الجملة البرمجية الموضحة في الشكل 9.12



شكل 9.12: اشتغال الإثباتات في لغة جافا

ومثال على ذلك، سنشتغل كائناً من الصنف (*Table*) كما يلي:

`Table t1 = new Table();`

وقد يتم اشتغال عدد غير محدود من الإثباتات من صنف واحد، وكل كائن منهم له صفات خاصة وقيمة خاصة التي لا تتأثر بقيمة الإثباتات الأخرى.

القيم المختلفة التي يتم تمريرها للدالة البنائية عند اشتغال كائن جديد، ينتج عنها كائن مختلف في كل مرة، والاختلاف في قيم الصفات بينما الصفات والسلوك مشترك.



#### 9.4.4 كيف نتفاعل مع بيانات الكائنات؟

وهذا سؤال بكل تأكيد يطرحه الآن الدارس، فكيف لنا أن نغير قيم المتغيرات الخاصة بالكائن، أو طباعتها أو معالجتها؟ والإجابة أن التفاعل مع متغيرات دوال الكائنات يكون بجملة بسيطة على الهيئة التالية:

```
objectName.variable = value;
```

```
objectName.Method();
```

فيتم كتابة اسم الكائن ثم (.) وتلتفظ (*Dot*) ثم اسم المتغير أو الدالة المراد التفاعل معها، وهذا مثال للتفاعل مع المتغير (*length*) حيث نريد تغيير قيمته لتصبح 34.5f :

```
t1.length = 34.5f;
```

وعند كتابة اسم الكائن ثم (.) فإن المحرر يظهر لك كافة المتغيرات والدوال التي يمكنك رؤيتها والتعامل معها كما في الشكل 9.13، ونقول يمكنك رؤيتها، لأن عدداً منها قد يكون محظياً رؤيته حسب محددات الوصول التي سنناقشها في الفصل 9.5.



شكل 9.13: التفاعل مع بيانات الكائنات في لغة جافا

والسؤال الذي يطرح نفسه الآن، إذا تم منعنا من خلال هذه المحددات من رؤية المتغيرات، ما الحل؟ الإجابة هو استخدام نوع من الدوال اسمها دوال الوصول (*Access Methods*) وهي دوال تسمح لنا باسترجاع قيم أو إعطاء قيم للمتغيرات الخاصة بالكائنات ولها اسم مشهور (*Set & Get Methods*) وهي دوال سيتم التعرف عليها بوضوح خلال الأمثلة التوضيحية التالية.

**الأنواع المشتقة** في لغة جافا هي الأنواع التي يقوم المبرمج بتعريفها واشتقاقها من أنواع أساسية موجودة، والحال ذاته مع الأصناف فهي نوع جديد مكون من أنواع أخرى أساسية، ثم بعد ذلك يتم تعريف المتغيرات منها بطريقة تختلف عن تعريف المتغيرات من **الأنواع الأساسية** كما تابعنا.

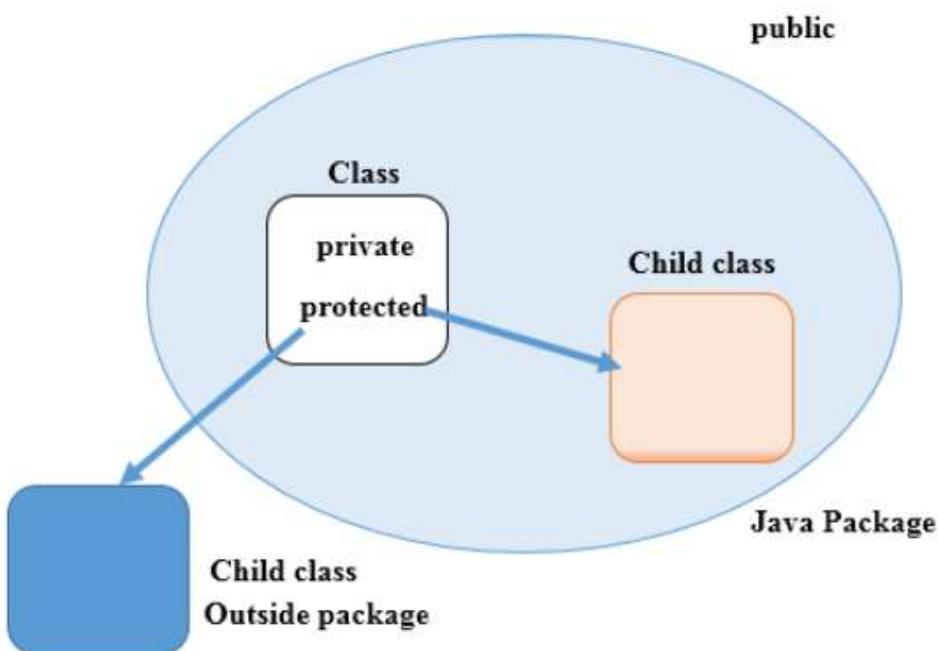


### 9.5 محددات الوصول (Access modifiers)

هي مجموعة من الكلمات الممحورة يتم من خلالها التحكم في التفاعل مع المتغيرات والدوال سواء بتعديل قيمها أو قراءتها. ومحددات الوصول أربعة أنواع؛ يمكن ترتيبها من حيث الأكثر سماحةً للوصول للأقل على النحو التالي:

1. **عام (public)**: العنصر الذي يسبق تعريفه هذا المحدد يكون متاحًا استخدامه والتعامل معه من كافة الأصناف الموجودة في هذا المشروع.
2. **محمي (protected)**: العنصر الذي يسبق تعريفه هذا المحدد يكون متاحًا للاستخدام والتعامل معه داخل أي صنف من أصناف الحزمة البرمجية (Package) التي ينتمي لها وكذلك الأصناف الأبناء في الحزم الأخرى.
3. **بدون محدد وصول (default)**: إذا لم نضع أي محدد قبل العنصر فهذا يعني أنه يحمل محدد وصول (default) وبالتالي يكون متاحًا للاستخدام والتعامل داخل أي صنف من أصناف الحزمة الحالية.
4. **خاص (private)**: العنصر الذي يسبق تعريفه هذا المحدد يكون متاحًا للاستخدام والتعامل معه فقط داخل الصنف الحالي ولا يمكن مشاهدته والتعامل معه إطلاقاً خارج هذا الصنف.

المحددات الأربع يمكن تلخيصها في الشكل التوضيحي 9.14 التالي:



شكل 9.14: تأثير محدد الوصول *protected* في لغة جافا

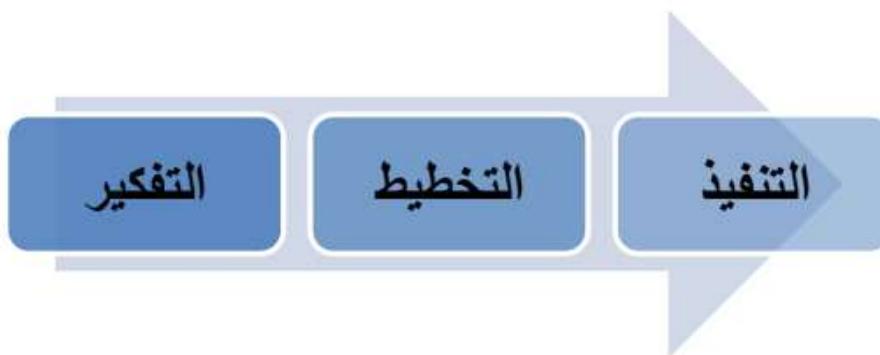
وفي الجدول 9.1 يتم اختصار الرؤية لكلٍ من هذه المحددات:

الحزم الأخرى	الابن	الحزمة الحالية	الصنف الحالي	محدد الوصول
متاح	متاح	متاح	متاح	<i>public</i>
غير متاح	متاح	متاح	متاح	<i>protected</i>
غير متاح	غير متاح	متاح	متاح	<i>default</i>
غير متاح	غير متاح	غير متاح	متاح	<i>private</i>

وبحثًا عن الأمان وعن تقيين استخدام العناصر الموجودة في الكائنات فإننا نحدد المتغيرات بالمحدد (*private*) وبالتالي فالوصول لهذه الصفات وتغيير قيمها لا يمكن أن يحدث إلا من خلال دوال الوصول (*set & get*) لتغيير قيمها أو الحصول على قيمها وذلك ما سنتابعه في المثال التوضيحي التالي 9.6.

#### 9.6 مثال شامل توضيحي

هذا المثال سنقوم بمناقشته من خلال المراحل الثلاث الموضحة بالشكل 9.15 وذلك لمناقشة كافة أساسيات البرمجة شيئاً فشيئاً التوجه التي تحثنا عنها.



شكل 9.15: خطوات حل المشاكل برمجيًا

#### مثال توضيحي 9.1:

أنشئ نوع جديد باسم *Student* تتمكن من خلله من تمثيل أي طالب بالكلية مع توفير الدوال التي تمكنا من التعامل مع بياناته.

### 9.9.1 مرحلة التفكير

في هذه المرحلة، نقوم بالتفكير في عنصر المشكلة الذي نريد تطوير أو إنشاء صنفًا برمجيًا له، ونبداً في تحليل الصفات التي تميز هذا الصنف وما هو السلوك البرمجي الذي يلزمها؟

فعد النظر إلى الصنف "طالب" ستجد مثلاً أن له اسم ورقم جامعي وغير ذلك، كذلك له سلوك مثل طباعة البيانات وتغييرها وتسجيل مساق وتغيير شعبة مساق وغير ذلك، وقد أجملت عدداً من هذه الصفات والسلوك في الشكل 9.16 بغض النظر وإلا فإننا بحاجة لتحليل أعمق للوصول لكافه الصفات والسلوك المتاح.



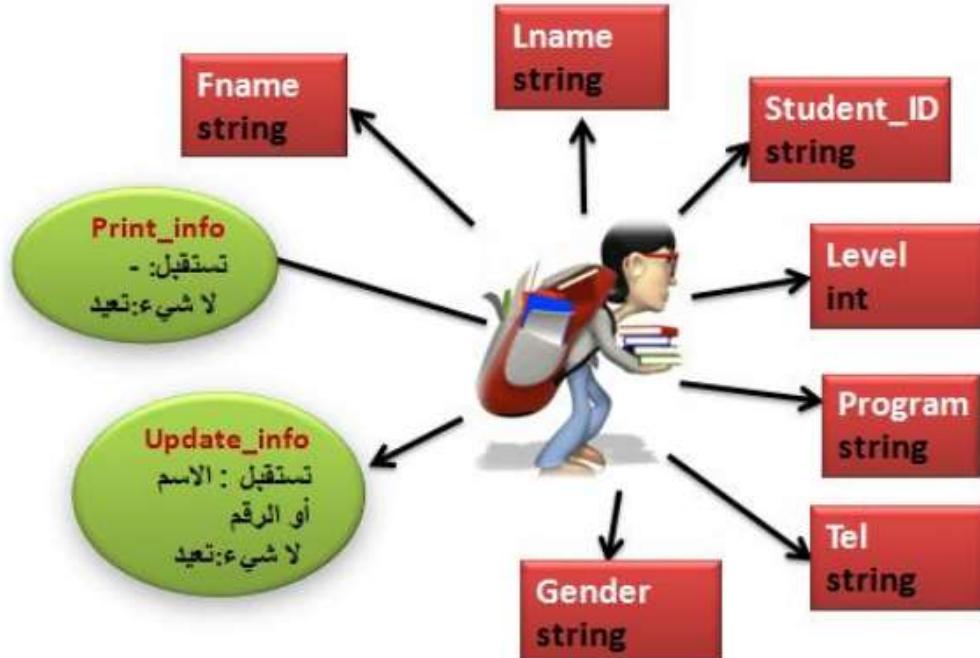
شكل 9.16: عدد من صفات وسلوك صنف الطالب

وهذه الصفات والسلوك يتم الحصول عليها من خلال التفكير المعمق في طبيعة العنصر الذي نريد تطوير صنف له، كما يمكننا السؤال هنا ما هي العمليات التي يقوم هذا العنصر بتنفيذها على بياناته أو التي يحتاج الآخرون تنفيذها على بياناته وهنا يصبح لدينا قائمة بالسلوك، أما عند الحديث عن الصفات، فنحتاج فقط لتحديد كل صفة يتضمن بها هذا العنصر من الصفات المميزة القابلة للتفاعل مثل الاسم وعنوان ورقم الهاتف والمستوى الجامعي والبرنامج الأكاديمي الذي يدرسها وهذا، ولا يتم الحديث عن صفاتيه النفسية مثل "عصبي"، "متقابل" وغير ذلك من الصفات.

### 9.9.1 مرحلة التخطيط

في هذه المرحلة يتم تحويل كل صفة من الصفات إلى متغير مع تحديد نوعه واسمها، كما يتم تحديد الأسماء المناسبة للدوال وما هي البيانات ونوعها التي تلزمها للعمل وكذلك نوع البيانات التي تعدها هذه الدوال.

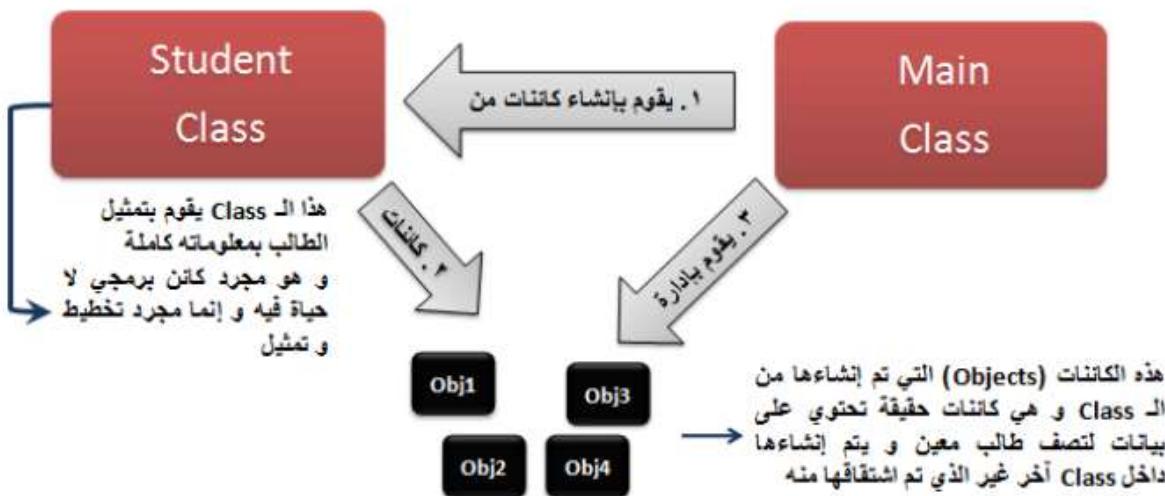
في الشكل 9.17 أخص لك هذه المرحلة للصنف "طالب".



شكل 9.17: بيانات المتغيرات والدوال الخاصة بالصنف "طالب"

تخطيط التعامل بين وحدات المشروع:

قبل البدء في مرحلة التنفيذ، دعنا نستوعب سوياً كيفية توزيع المشاريع التي تحتوي على بناء أصناف، حيث يصبح المشروع يحتوي أكثر من الصنف الرئيسي والذي تعاملنا معه خلال الأبواب السابقة والذي يحتوي على الدالة الرئيسية (**main**). الآن أصبح المشروع يتضمن ثلاثة وحدات، موضحة بالشكل 9.18.



شكل 9.18: بيانات المتغيرات والدوال الخاصة بالصنف "طالب"

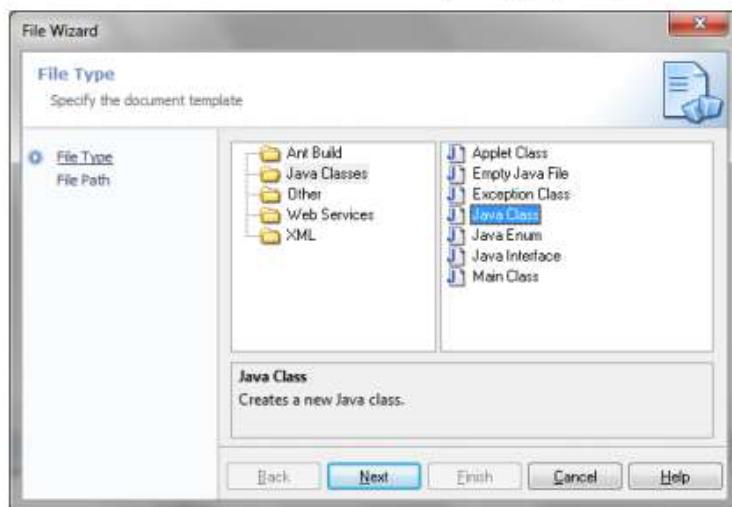
كما تلاحظ فإن الوحدات الثلاث وهي:

1. الصنف الرئيسي: وهو الصنف الذي من خلاله يتم بدء تنفيذ المشروع ومن خلاله يتم ربط وحدات المشروع المختلفة، وفي حالتنا هذه سيتم اشتقاق الكائنات من الصنف "طالب" فيه.
2. الصنف المستحدث وهو "الطالب" في حالتنا.
3. الكائنات التي يتم اشتقاقها من الصنف "طالب"، وهذه الكائنات يتم اشتقاقها داخل الصنف الرئيسي. هذا التخطيط يعتبر تخطيطاً عاماً وقد يختلف من حالة لحالة أخرى فقط من حيث إمكانية إنشاء أكثر من صنف مستحدث وإمكانية اشتقاق كائنات من أصناف في غير الصنف الرئيسي.

### 9.9.3 مرحلة التنفيذ

في هذه المرحلة سنقوم بإنشاء الصنف برمجياً من خلال لغة جافا، ولكي نضيف صنفاً جديداً للمشروع، يتم ذلك من خلال الخطوات التالية:

1. نقوم بإنشاء مشروع ونعطيه اسمًا مثل *studentProject*.
  2. ثم من القائمة (File) نختار الخيار (Java class) ثم (new) ثم (file) كما تشاهد في
- الشكل 9.19
3. نكتب اسم الصنف ونعرف أن يكون الحرف الأول كبيراً ولتكن (*Student*) انتبه لأن يكون هذا الصنف داخل المجلد (*src*) الخاص بلغة جافا على قرصك الصلب.
  4. سيظهر لنا صنفاً فارغاً لا يوجد به إلا التعريف وتعریف الدالة البنائية.
  5. نبدأ في إضافة المتغيرات والدوال اللازمة.



شكل 9.19: الخطوة الثانية لإنشاء صنف جديد

```

1 import javax.swing.JOptionPane;
2 public class Student {
3     // في الجزء القادم سنقوم بتعريف كل متغير من المتغيرات التي تمثل صفات الطالب
4     private String fname;
5     private String lname;
6     private String tel;
7     private String stu_id;
8     private String Program;
9     private String Gender;
10    private int level;
11    // الدالة التالية عبارة عن الدالة البناءة وهي تضم إعطاء قيمة بداعية لكل متغير
12    public Student() {
13        fname = "First_Name";
14        lname = "Last_Name";
15        tel = "00000000";
16        stu_id = "120090000";
17        Program = "Diploma";
18        Gender = "male";
19        level = 1;
20    }
21    // الأسطر القادمة عبارة عن دالة مخصصة لطباعة بيانات الطالب في صندوق حوار واحد
22    public void print()
23    {
24        JOptionPane.showMessageDialog(null, "The information is \n"+
25        "Name: "+fname+" "+lname+"\n"+
26        "ID: "+stu_id+"\n"+

```

```

27 "Program: "+Program+"\n"+
28 "Level: "+level+"\n"+
29 "Gender: "+Gender+"\n"+
30 "Tel.: "+tel);
31 }

```

الأسطر القادمة تحتوي على تعديل للمتغير الخاص برقم الطالب، وهي تستقبل الرقم الجديد //

```

32
33 public void updateId( String ID )
34 {
35   stu_id = ID;
36   JOptionPane.showMessageDialog(null, "The ID is updated");
37 }
38 } // القوس الخاتم للصنف الخاص بالطالب

```

الصنف البرمجي الخاص بالطالب

```

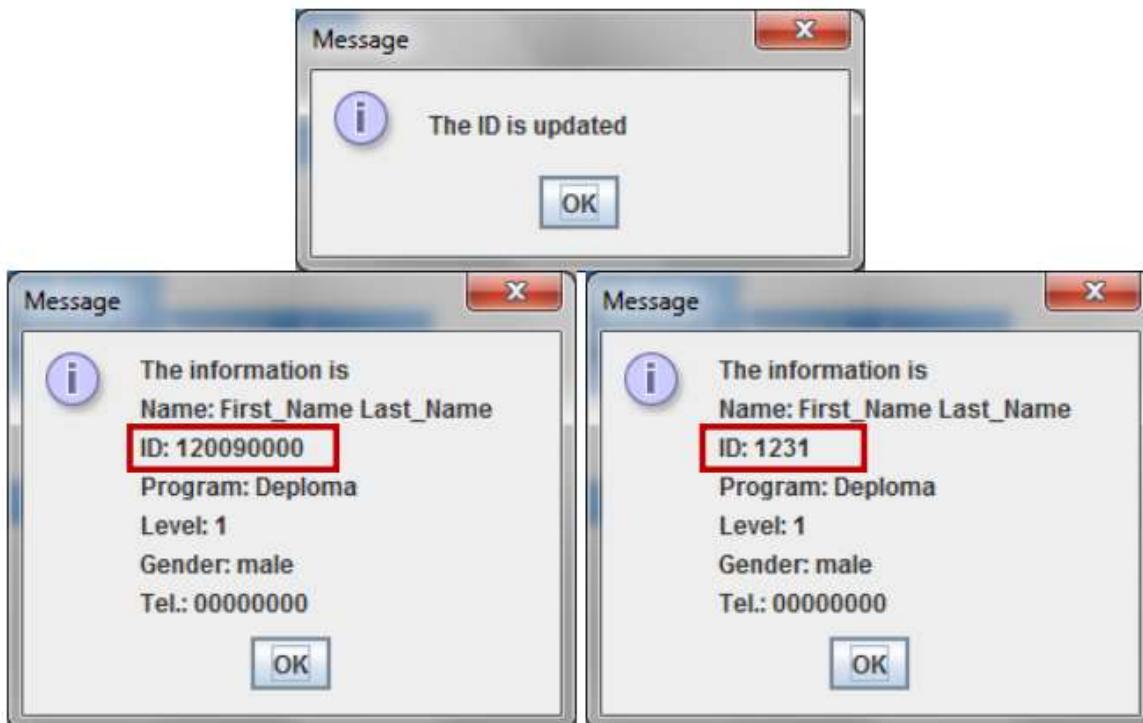
1 public class StudentProject {
2   public static void main(String[ ] args){
3     Student st1 = new Student();
4     st1.updateId("1231");
5     st1.print();
6     Student st2 =new Student();
7     st2.print();
8   }

```

الصنف الرئيسي للمشروع

في الشكل 9.20 تلاحظ نتائج تنفيذ هذا المشروع، عليك أن تلاحظ أننا في الصنف الرئيسي قمنا بإنشاء كائنين مختلفين هما `st1` و `st2` وعند طباعتهما ستجد أن بياناتهم متطابقة تماماً إلا في قيمة المتغير `ID` فالكائن الأول قيمة هذا المتغير له هي `1231` وهذه هي القيمة التي تختلف عن القيمة الموجودة في الدالة البنائية وقد قمنا بغيرها من خلال الدالة `updateId` كما يظهر في السطر 4 من الصنف الرئيسي. وهنا يظهر لك دور الدالة البنائية وهو إعطاء قيم للكائنات الجديدة إذا لم يعطهم المستخدم قيمًا من عنده.

في الأمثلة القادمة يمكننا تطوير دوال بنائية تستقبل قيم من المستخدم بالإضافة للدالة الافتراضية.



شكل 9.20: تنفيذ المشروع يظهر نتائج الكائن الأول بعدها الثاني يسألا

الآن لعلك أصبحت قادرًا على فهم الأهداف الرئيسية من هذا الباب حتى الآن والتي تتعلق في فهم المفهوم العام للبرمجة شيئاً فشيئاً التوجه ومفاهيمها، والأهم من ذلك قدرتك على تعريف وبناء أصنافاً برمجية لأي شيء من حولك في صورته البسيطة. ومع الأمثلة القادمة سيصبح لديك القدرة أكبر على تطوير الأصناف أسرع وأسهل.

**توجيه وإرشاد** ستصبح مهاراتك أعلى في تطوير الأصناف من خلال التمرين المتواصل عليها، حيث يمكنك الآن تحويل كل ما حولك إلى أصناف برمجية، ومع الوقت لن تحتاج إلى دقائق بسيطة لتطوير الصنف.



#### 9.7 مصفوفة الكائنات

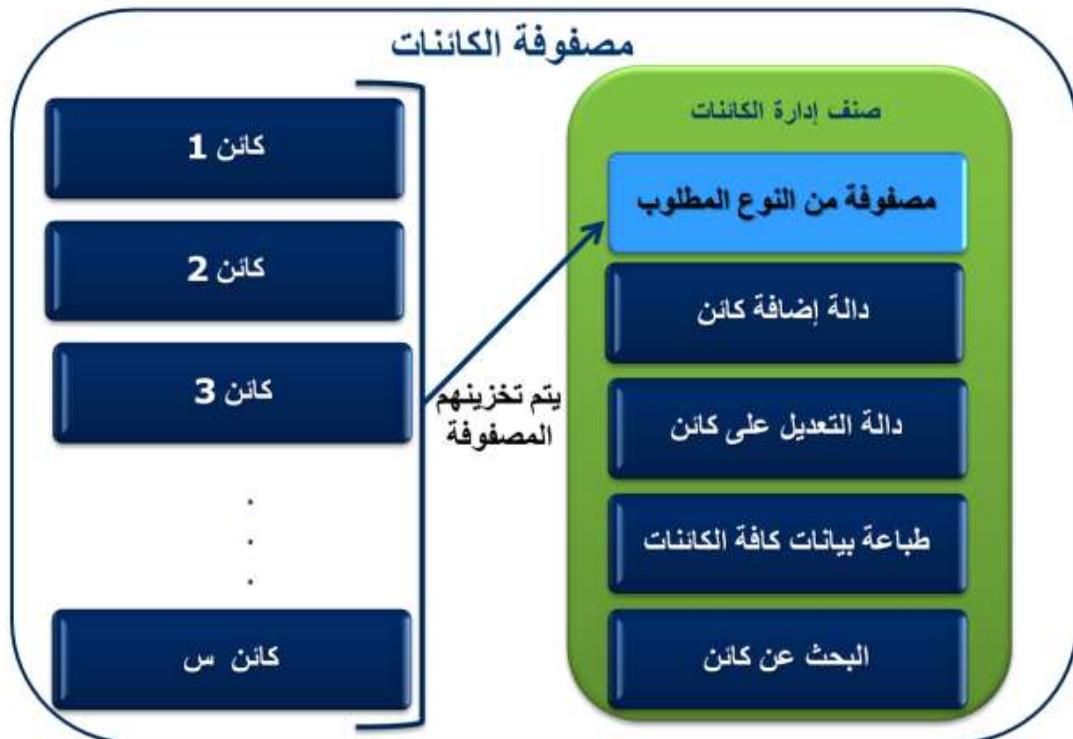
في الباب السادس من هذا الكتاب استعرضنا الحديث عن المصفوفات كونها إحدى تراكيب البيانات التي يمكنها التعامل مع مجموعة من البيانات، واستعرضنا أنها لا تضم في خلاياها سوى نوع واحد من البيانات، حيث تم تخزين نوع صحيح أو نص أو غير ذلك، ولكن ماذا لو أردنا تخزين (اسم الموظف، رقمه، راتبه، مسماه الوظيفي) في خلية واحدة من المصفوفة؟ هل يمكن؟ بالطبع حسب ما تعلمناه سابقاً ستكون الإجابة، لا.

حسناً، بعد دراستنا لهذا الفصل ستصبح الإجابة نعم، حيث بإمكاننا الآن تعريف مصفوفة من نوع صنف جديد نحن نطوره، وبالتالي سيصبح بمقدورنا تخزين كائن في كل خلية، وكما تابعنا في الفصل السابق، فإن الكائن مكون من مجموعة من البيانات، وبالتالي سيصبح من السهل تخزين أكثر من نوع من البيانات في المصفوفة ولكن جميعهم ضمن اسم جديد من نوع الصنف الذي قمنا بتعريفه.

**إضافة:** إنشاء مصفوفة من نوع صنف محدد سيتيح لنا الفرصة لتخزين كائنات هذا الصنف في مكان محدد، وبالتالي سيصبح متاحاً لنا تنفيذ كافة العمليات التي تحتاجها على مجموعة الموظفين مثل البحث، الترتيب، طباعة الجميع، التعديل، وغير ذلك من العمليات.



ويختصر المنظر العام لهذا المفهوم في الشكل 9.21.



شكل 9.21: تنفيذ المشروع يظهر نتائج الكائن الأول يميناً والثاني يساراً

في المثال 9.2 سنستعرض إنشاء مصفوفة من الموظفين (*Array of Employees*) وسنحتاج لإنشائها ما يلي:

1. إنشاء صنف للموظف اسمه (*Employee*) يمثل النوع الذي سنقوم بتعريف المصفوفة منه.
2. إنشاء صنف جديد اسمه (*ArrayOfEmployee*) وفيه سيتم تعريف المصفوفة التي ستصنف كائنات الصنف (*Employee*) كما سيضم كافة العمليات التي يمكننا تنفيذها على مصفوفة الموظفين.

3. إنشاء كائن من مصفوفة الموظفين في الصنف الرئيسي ومعالجة بياناته مثل (إضافة موظف، حذف موظف، البحث عن موظف،....)

### مثال توضيحي 9.2:

تحتاج إحدى المستشفيات تطوير تطبيق يمكنها من تخزين بيانات كافة المرضى لديها ومعالجة بياناتهم بتتوفر العمليات التالية:

- إضافة مريض
- البحث عن مريض بالاسم
- طباعة تقرير يضم بيانات كافة المرضى.

كيف يمكنك مساعدتهم ؟

```

1 public class Patient {
2     private String name;
3     private String status;
4     private String priority;
5     private double cost;
6     public Patient(){
7         name = "Omar";
8         status = "R";    // مريض مقيد
9         priority = "high";
10        cost = 30; }
11    public Patient(String name, String priority) {
12        this.name = name;
13        this.priority = priority;
14        WhatStatus();
15        if (status.equalsIgnoreCase("R")){

```

```

16 int days = Integer.parseInt(JOptionPane.showInputDialog("Enter number of
days"));
17 calcCost(days); }
18 }
19 public void WhatStatus(){
20 if(priority.equalsIgnoreCase("high"))
21 this.status = "R";
22 else
23 this.status = "O";}
24 public void calcCost(int days){
25 if (status.equalsIgnoreCase("R") && (days >=1 ) )
26 this.cost = 10 + (days*20);
27 else
28 this.cost = 10; }
29 public void info (){
30 System.out.println("Name: "+name+"\nStatus: "+ status +"\nPriority:
"+priority+"\nCost: "+cost);
31 System.out.println("=====");
32 }
33 //Accessor Methods
34 public void setPriority(String priority){
35 this.priority = priority;
36 }
37 public String getPriority(){
38 return priority;
39 }

```

صنف المريض حيث تم فيه توضيح كافة الجزئيات التي تم شرحها سابقاً

```

1 public class PatientRecord {
2     private int Length;

```

```

3   private int counter =0;
4   private Patient [] Record ; //= new Patient[100];
5   public PatientRecord(int length){
6       this.Length = length; // validate
7       Record = new Patient[length];
8   }
9   public void addPatient( Patient p){ // المصفوفة
10    if(counter < Record.length){
11        Record[counter] = p;
12        counter++; }
13    else
14        System.out.println("Record is full"); }
15   public void report(){// كافة المرضى
16    if (counter> 0){
17        for (int i=0; i<counter ; i++)
18            Record[i].info(); }
19    else
20        System.out.println("Record is empty !!");
21    }
22 مستخدماً أيًّ من خوارزميات البحث التي درستها في الباب الثامن يمكنك البحث عن مريض //
23 }

```

صنف مصفوفة (سجل) المرضى حيث تم فيه توضيح كيفية إنشاء مصفوفة من الكائنات

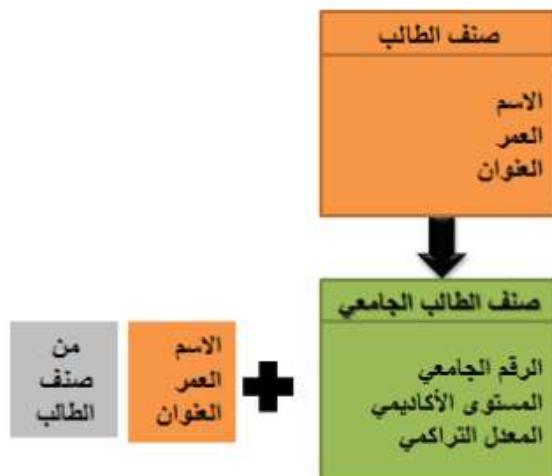
#### 9.8 الوراثة

قلنا في بداية الباب أنَّ لغة جافا تسمح لنا بإنشاء أصناف جديدة من أصناف تم إنشائها سابقاً وهذا ما يُعرف بالوراثة (*Inheritance*) وهذا وإن كان متاحاً بشكل عام إلا أنه يرتبط منطقياً بنوع الأصناف، فمثلاً ليس منطقياً أن يتم إنشاء صنف للطالب من صنف الطاولة ! ولكن يمكن إنشاء صنف الطالب من صنف آخر للإنسان وذلك لأنَّ ثمة علاقة وصفات مشتركة موجودة بين الصنفين.

في لغة جافا يسمح لنا بإنشاء صنف جديد أكثر تخصصاً يُعرف باسم الابن (*subclasses*) من صنف آخر أكثر عمومية يُعرف باسم الأب (*superclass*)، الصنف الأب هو صنف عام، يشمل صفات عامة بما يضمن وجود أصناف أخرى يمكنها الاستناد إليه، فعلى سبيل المثال:

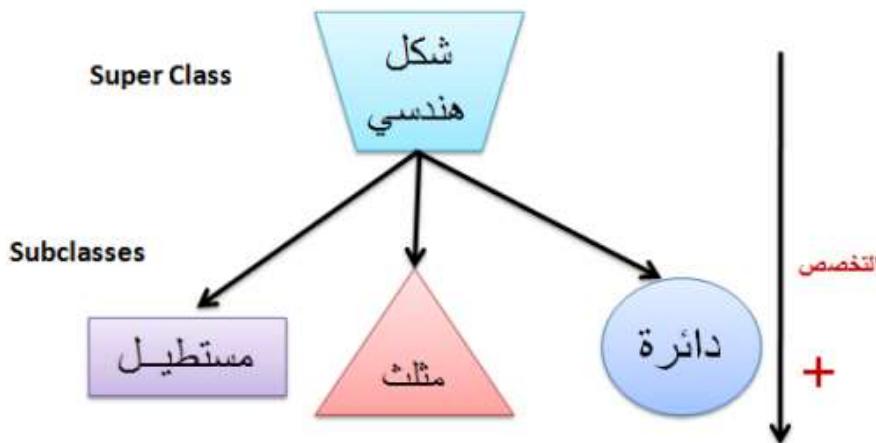
- الصنف "موظف ثابت" (صنف ابن) يمكن إنشاءه بالاعتماد على الصنف "موظف" (صنف أب).
- الصنف "طالب جامعي" (صنف ابن) يمكن إنشاءه بالاعتماد على الصنف "طالب" (صنف أب).
- الصنف "دائرة" (صنف ابن) يمكن إنشاءه بالاعتماد على الصنف "شكل هندسي" (صنف أب).

وبالنظر إلى هذه الأمثلة وغيرها، يمكن إدراك أنَّ الصنف الابن يتم استناده من الصنف الأب وذلك من خلال الاستفادة من عدد من صفات الأب وسلوكه ثم بإمكان الصنف الابن زيادة المناسب من الصفات والسلوك كما يمكِنه التعديل على سلوك لدى الأب.



شكل 9.22: توفير الوقت والجهد أحد مميزات تطبيق مفهوم التوارث

والتوارث في لغات البرمجة يمكن تطبيقه بأشكال مختلفة مثل (التوارث الفردي)، (التوارث المتعدد)، (التوارث المتعدد المستويات) (التوارث الهجين) وغير ذلك؛ في جافا يمكننا تطبيق الوارثة الفردية (*Single Inheritance*) ويقصد به أنَّ الصنف لا يرث إلا من صنف واحد فقط، وهذا بخلاف الوراثة في الحياة البشرية حيث أنَّ الابن قد يرث من أبييه. ومثال ذلك الشكل 9.23.

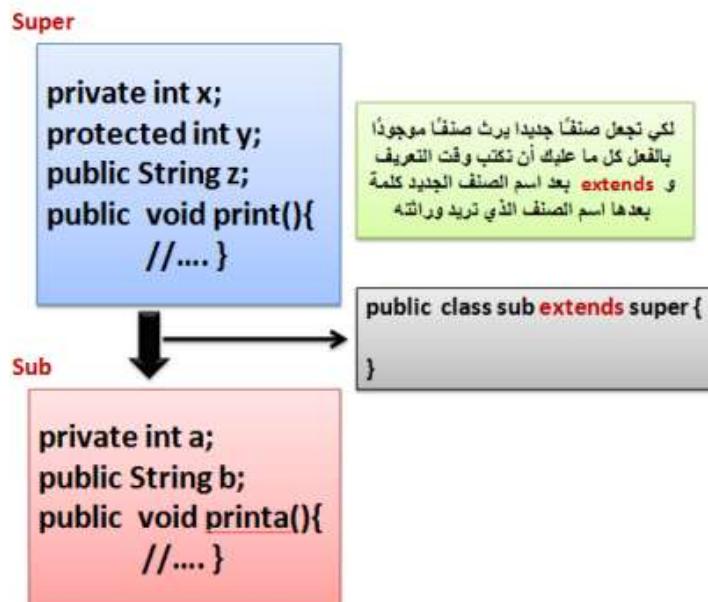


شكل 9.23: الوراثة الفردية

ولتطبيق هذا المفهوم نستخدم الكلمة المحوّزة (*extends*) وذلك لتوضيح العلاقة بين الابن (*Uni\_Student*) والأب (*Student*) كما في الجملة التالية:

```
public class Uni_Student extends Student { .... }
```

هذه الكلمة (*extends*) وهذه الجملة التعريفية تجعل الصنف الابن يرث كافة الصفات (المتغيرات) والسلوك (الدوال) من الصنف الأب ما لم يمنع محمد الوصول لها وذلك كما في الشكل 9.24.



شكل 9.24: تأثير محددات الوصول على العناصر التي يرثها الصنف الابن

محاولة التفاعل مع عنصر *private* من الاب في الابن يعطي خطأ برمجي.



فيما يلي، نستعرض مثلاً توضيحاً لمفهوم الوراثة وذلك بتطبيقه على الطالب كأب وطالب البحث العلمي كابن.

في هذا المثال سنحتاج إلى إنشاء ما يلي:

1. إنشاء صنف الأب أولاً باسم (*Student*) مع الانتباه إلى اختيار محددات الوصول للأب بشكل مناسب.
2. إنشاء صنف الابن وذلك من خلال اشتقاقه من الابن باستخدام الكلمة (*extends*).
3. استكمال الصفات والسلوك للصنف الابن.

### مثال توضيحي 9.3:

مستخدماً مفهوم الوراثة، اكتب برنامج يضم صنف الطالب، وطالب البحث العلمي مع اشتقاق كائنات من كلاهما وطباعة النتائج.

```

1 class Student {
2     protected long id;
3     protected String name;
4     protected double gpa;
5     public Student(long id, String name, double gpa) {
6         this.id = id;
7         this.name = name;
8         this.gpa = gpa;
9     }
10    public Student() {
11        this(999999, "No name", 0.0);
12    }
13    public void changeGPA(double newGPA) {
14        gpa = newGPA;
15    }
16    public double getGPA() {
17        return gpa;

```

```

18 }
19 public void print() {
20   System.out.print(id+"\t"+name+ "\t"+gpa);
21 }
22 }

```

الجزء الأول: صنف الطالب

```

1 class ResearchAssistant extends Student {
2   private int workLoad; // in hours
3   ResearchAssistant(long id, String name, double gpa, int workLoad){
4     super(id, name, gpa);
5     this.workLoad = workLoad;
6   }
7   ResearchAssistant() {
8     workLoad = 0;
9   }
10  public void print() {
11    super.print();
12    System.out.print("\t" + workLoad);
13  }

```

الجزء الثاني: صنف طالب البحث العلمي

```

1 public static void main (String[] args) {
2   ResearchAssistant s1;
3   s1 = new ResearchAssistant();
4   s1.print();
5   ResearchAssistant s2;
6   s2 = new ResearchAssistant(991234, "Mahmoud Alfarra", 3.45, 15);
7   s2.changeGPA(3.75);
8   System.out.println();

```

```
9 s2.print();
```

```
10 }
```

### الجزء الثالث: استدعاء الأصناف وطباعة البيانات في الصنف الأساسي

من المعلوم أنَّ طالب البحث العلمي يحصل على ساعات عمل بجانب دراسته، وبالتالي يحصل على مقابل مالي، وهذه الصفة غير موجودة في الطالب بشكل عام، لذلك قمنا في هذا المثال بتطوير صنف طالب يحتوي على البيانات الأساسية مثل الاسم، المعدل والرقم الجامعي في الأسطر 3,4,5 من الجزء الأول، ثم قمنا بإعطائها قيمًا بدائية من خلال الدالة البنائية في الأسطر من 5 إلى 9 من الجزء ذاته، مع بناء دوال أساسية مثل الطباعة وتغيير القيم لبعض المتغيرات باستخدام مفهوم دوال الوصول التي ناقشناها في بداية الباب.

بينما في الجزء الثاني، قمنا بتوريث الصنف (*Student*) للصنف (*ResearchAssistant*) في السطر الأول من هذا الجزء، وذلك باستخدام جملة التوريث الموضحة، ثم أضفنا المتغير الذي تحتاجه وهو (*workload*) والخاص بعدد ساعات العمل.

في الأسطر من 3 إلى 6 تم إنشاء الدالة البنائية المتخصصة والتي تُعطي قيمًا للمتغيرات الخاصة بالصنف، وكما تشاهد فإنَّ الصنف (*Student*) به دالة تعمل بالمبدأ ذاته حيث تستقبل قيمًا وتمررها للمتغيرات. في هذه الحالة احتجنا إلى استدعاء الدالة البنائية من الصنف (*Student*) كما في السطر رقم 4 بشكل خاص من خلال الجملة (*super*) ومُررنا له القيم التي نريدها، وبدون هذا الاستدعاء الخاص فلن يتم إعطاء قيم مخصصة للمتغيرات، بل سيتم العمل بالقيم الافتراضية الموجودة في الصنف الاب.

في السطر رقم 9 من الجزء الثاني قمنا بتعريف دالة للطباعة خاصة بالصنف (*ResearchAssistant*), ولكننا بدلاً من طباعة كافة البيانات (الجديدة، المورثة) قمنا فقط باستدعاء الدالة (*print*) الخاصة بالأب من خلال الجملة في السطر رقم 10 ثم أضفنا عليها طباعة قيمة المتغير (*workload*) ليتم بذلك طباعة كافة بيانات الكائنات المشتقة من الصنف (*ResearchAssistant*) وهذه الدالة مثال عمل على مفهوم إعادة تطبيق الدوال.

أخيرًا في الدالة الأساسية، تم اشتراق كائنين من الصنف (*ResearchAssistant*) مع الاعتماد على القيم الافتراضية في السطر رقم 3 بينما تم إعطاءهم قيمًا خاصة في السطر 6، كما تم استخدام الدالة (*print*) لهذه الكائنات وهي موجودة في الصنفين (الأب، الأبن): لكنه اختارها من الكائن الأبن. وفي حالتي الطباعة، طبعت في الأولى القيم الافتراضية، وفي الثانية القيم المخصصة.

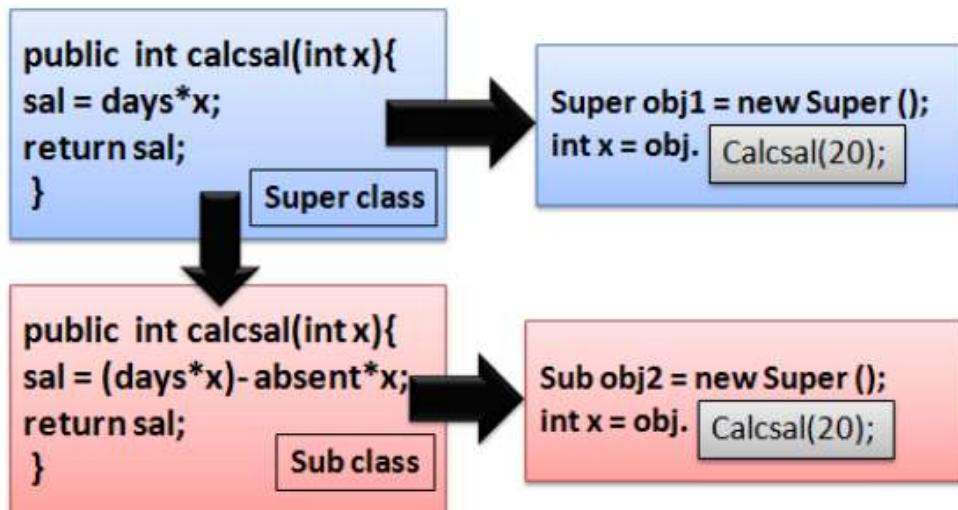
في السطر رقم 7 تم استدعاء الدالة (*changeGPA*) وهي دالة تم توريثها للصنف الأبن من الصنف الأب.

```
General Output
-----Configuration: Example10
999999 No name 0.0 0
991234 Mahmoud Alfarra 3.75 15
Process completed.
```

شكل 9.25: نتائج المثال 9.3

ويبيقى مجموعة من الملاحظات المهمة:

- الصنف الابن لا يمكن له التفاعل مع عناصر الصنف الاب وهي (*private*) إلا من خلال دوال (*set&get*) والمعرفة بمحدد وصول يسمح بذلك، وهي موضحة في الجزء الأول السطر 16، حيث أن الدالة (*get*) تعيد قيم ولا تستقبل بينما الدالة (*set*) تستقبل قيم ولا تعيد.
- عندما يحتاج الصنف الابن لاستخدام دالة موجودة في الصنف الأب، يتم استدعاءها بطريقة طبيعية كأنها لديه.
- يمكن للصنف الابن التعديل على سلوك للأب فيما يعرف بمفهوم (*override Method*) ويقصد بها إعادة تطبيق دالة تم تطبيقها في الصنف الأب بطريقة ثم في الصنف الابن بطريقة أخرى، وكلاهما له توقيع مماثل بينما الاختلاف في التطبيق، مثال ذلك: دالة الراتب للموظف "الصنف الأب" هي دالة تحسب الراتب الثابت مطروحا منه ضريبة الدخل، بينما في الصنف الابن للموظف المثبت بعمولة هي دالة تحسب الراتب الثابت مضافا له العمولة التي يحصل عليها كنسبة من مبيعاته ويطرح من المجموع ضريبة الدخل، هنا كلا الدالتين تحسب الراتب وتتوقع الدالتين مماثل؛ إلا أن التطبيق اختلف. (ما الفارق بينها وبين التحميل الزائد؟)
- إن قام الصنف الابن بإعادة تطبيق دالة متوفرة عند الأب، واحتاج الابن استدعاء دالة الأب لإنجاز مهمة معينة، يتم استدعاءها بالطريقة التالية، على اعتبار أن (*method*) هي دالة متوفرة في الأب والابن سوية.  
*super.method();*
- عند توفر الدوال المعاد تطبيقها في الأب والابن ونحتاج لاستدعائهما من خلال الكائن، فإن كل دالة تتطلب من الكائن الخاص بالصنف، بمعنى إن كان الكائن للأب يتم استدعاء دالة الأب وإن كان الكائن للابن يتم استدعاء دالة الابن، كما بالشكل 9.26.



شكل 9.26: استدعاء الدوال المعدتعريفها من كائنات مختلفة

### 9.9 تعدد الأشكال

المفهوم الرابع من مفاهيم البرمجة شيئاً فشيئاً التوجه هو تعدد الأشكال ويقصد به قدرة كائنات الابن أن تشير لكتائات الأب، أي أن كائنات الابن وهي الأكثر تخصصاً يمكن أن تحل محل كائنات الأب وهي الأقل تخصصاً والأكثر عموماً، وهذا يمنح التطبيق أكثر مرونة وقدرة على الانتاج.

تعدد الأشكال هو مفهوم يساعد المبرمجين على تطوير برمجيات أكثر لدونة و مرونة



من خلال تعدد الأشكال سيصبح من حق المبرمج التحكم في كيفية تطبيق سلوك الابن.



باختصار شديد؛ فإن تطبيق مفهوم تعدد الأشكال مرتبط بثلاثة مفاهيم هي:

1. الوراثة: والتي تم مناقشتها في الفصل السابق، حيث أن تعدد الأشكال يُعد تطويراً للوراثة من حيث تطبيق الدوال، وهذا يعني أن تعدد الأشكال لا يمكن تطبيقه دون وجود الوراثة.

2. إعادة تطبيق الدوال: وقد تم شرحها في الفصل السابق، وهنا سيتم تطويرها لنصل لمرحلة تعريف الدالة في الأب دون تطبيق ثم يتم تطبيق الدالة في الأبناء وذلك لإعطاء فرصة أعلى للمرونة ولتطبيق الحلول.

3. الدوال والأصناف المجردة (*Abstract Methods and Classes*): هذا المفهوم ظهر كنتيجة طبيعية للرغبة في تطوير البرمجة في محاكاة الواقع، ويقصد بالدوال المجردة أي دالة بلا تطبيق، حيث يتم الاكتفاء بتعريف الدالة فقط دون تطبيق ولذلك يتم في توقيع الدالة كتابة اللفظ (*abstract*) كإشارة إلى أنها بلا تطبيق كما بالشكل 9.27:



شكل 9.27: كيفية تعريف الدالة المجردة من التطبيق

وهذه الدوال يتم تطبيقها في الابن من خلال إعادة تعريفها مع توفير الأوامر البرمجية اللازمة للتطبيق وبالتالي لا يتم ذكر اللفظ (*abstract*).

ومن القواعد أن أيُّ صنف يحتوي على دالة واحدة مجردة (*Abstract Method*) فمباشرة يصبح الصنف الخاص بها هو مجرد (*Abstract Class*) ويتم كتابة اللفظ (*abstract*) في تعريف الصنف كإشارة إلى أنه يضم دالة واحدة على الأقل بلا تطبيق كما بالشكل 9.28.

```
public abstract class Employee{  
    ---  
    public abstract void calcSal();  
}
```

شكل 9.28: تعريف الصنف المجرد

الصنف المجرد لا يمكن اشتراق كائن منه، بل يمكن اشتراق ابن له ليطبق الدوال المجردة.



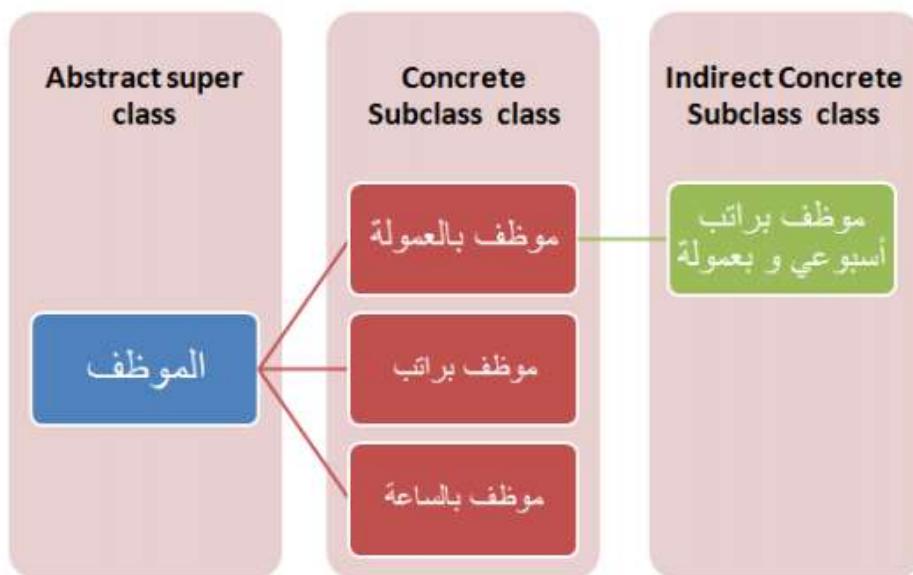
من أنواع الأصناف (*Concrete Class*) وهو ذو دوال لها تطبيق، والأصناف المجردة (*Abstract Class*) وهي التي تضم على الأقل دالة واحدة بلا تطبيق.



ولإيصال مفهوم تعدد الأشكال بشكل أكثر وضوحاً، نستعرض سوياً المثال 9.4 حيث نحتاج لإنشاء أصناف متعددة لمحاكاة الموظف، وكما تعلم فإن طريقة حساب الراتب للموظفين تختلف باختلاف نوع الموظف، فمثلاً الموظف ذو الراتب الثابت ليس كالموظف العقد وليس كالموظف الذي يعمل بعمولة مع راتب ثابت، وبالتالي سنحتاج إلى تعريف صنف مجرد يضم كافة الصفات والسلوك المشترك، بينما يضم دالة حساب الراتب كدالة مجردة من التطبيق، ويتم لاحقاً في الأصناف الأبناء (موظف مثبت، موظف عقد، موظف عمولة) تطبيق هذه الدالة بطرق تناسب مع نوع الموظف. عند الحاجة لاشتقاق كائنات، يتم اشتقاق كائن من الصنف المطلوب وحساب الراتب له بالطريقة التي تناسبه.

#### مثال توضيحي 9.4

قم بإنشاء تطبيق يسمح باحتساب راتب الموظف لإحدى الشركات، علمًا بأن الشركة تتعامل مع أنواع مختلفة من الموظفين كما بالشكل 9.29.



شكل 9.29: توضيح أنواع الموظفين للمثال 9.24

```

1 public abstract class Employee{
2     private String firstName;
  
```

```

3   private String lastName;
4   private String ID;
5   public Employee( String firstName, String lastName, String ID ) {
6       this.firstName = firstName;
7       this.lastName = lastName;
8       this.ID = ID; }
9       هنا يمكن تطوير دوال الوصول للمتغيرات الخاصة // 
10  public String info() {
11      return "The information is: " + getfirstName() + getLastname() + getID() ;
12  }
13  دالة مجردة من التطبيق، س يقوم الأبناء بتطبيقاتها // 
14  public abstract double earnings();
15 }
```

صنف الموظف وهو الصنف الرئيسي العام الذي يضم دوال مجردة من التطبيق

```

24 public class SalariedEmployee extends Employee {
25     private double weeklySalary;
26     public SalariedEmployee( String firstName, String lastName, String ID,
27         double weeklySalary ) {
28         super( firstName, lastName, ID ); // تمرير قيم الدالة البنائية الخاصة بالأب
29         this.weeklySalary = weeklySalary ;
30     }
31     هنا يمكن تطوير دوال الوصول للتفاعل مع المتغيرات الخاصة // 
32     الدالة التالية كانت مجردة في الأب، بينما الابن يوفر لها تطبيق كما شاهد // 
33     public double earnings() {
34         return getWeeklySalary(); }
35     هذه الدالة في الأب تم تطبيقها بطريقة، بينما تم التعديل على التطبيق هنا // 
36     public String info()
37 { }
```

```

38 return super.info() + getWeeklySalary();
39 }
40 }
```

صنف الموظف (براتب ثابت) وهو الصنف الابن حيث يختلف تطبيقه لدالة احتساب الراتب

```

1 public class CommissionEmployee extends Employee {
2     private double grossSales; // قيمة المبيعات الأسبوعية
3     private double commissionRate; // نسبة العمولة للموظف
4     public CommissionEmployee( String firstName, String lastName,
5         String ID, double grossSales, double commissionRate ){
6         super( firstName, lastName, ID );
7         this.grossSales= grossSales;
8         this.commissionRate= commissionRate;
9     }
10    public double earnings() {
11        return getCommissionRate() * getGrossSales();
12    }
13    public String info()
14    {
15        return super.info() + getGrossSales() + getCommissionRate();
16    }
17 }
```

صنف الموظف (عمولة) وهو الصنف الابن حيث يختلف تطبيقه لدالة احتساب الراتب يمكنك بالطريقة ذاتها تطوير صنف الموظف بالساعة، والذي يختلف عن الأب في كيفية حساب الراتب، حسب معادلة تقوم على حاصل ضرب عدد ساعات العمل مع القيمة المالية للساعة الواحدة.

```

1 public class SalariedEmployee extends Employee {
2     private double weeklySalary;
3     public SalariedEmployee( String firstName, String lastName, String ID,
```

```

double weeklySalary ) {
4   super( firstName, lastName, ID ); // تمرير قيم للدالة البناءة الخاصة بالأب
5   this.weeklySalary = weeklySalary ;
6 }
7 هنا يمكن تطوير دوال الوصول للتفاعل مع المتغيرات الخاصة //
8 الدالة التالية كانت مجرد في الأب، بينما الابن يوفر لها تطبيق كما تشاهد //
9 public double earnings()
10      return getWeeklySalary();
11 }
12 هذه الدالة في الأب تم تطبيقها بطريقة، بينما تم التعديل على التطبيق هنا //
13 public String info()
14 {
15   return super.info() + getWeeklySalary();
16 }
17 }
```

صنف الموظف (براتب ثابت) وهو الصنف الابن حيث يختلف تطبيقه لدالة احتساب الراتب

```

1 public class BasePlusCommissionEmployee extends CommissionEmployee
2 {
3   private double baseSalary; // الراتب الأسبوعي الثابت
4   public BasePlusCommissionEmployee(String firstName, String lastName,
      String ID, double grossSales, double commissionRate, double baseSalary)
5   {
6     super( firstName, lastName, ID, grossSales, commissionRate );
7     this.baseSalary = baseSalary ;
8   }
9   تعتمد على هذه الدالة تحتسب طريقة الراتب بطريقة مختلفة عن الأب المباشر وغير المباشر لك /*/
10  /* كلها في التطبيق من خلال استدعاء دالة الأب ثم الإضافة عليها.
11  public double earnings() {
12    return getBaseSalary() + super.earnings();
```

```

12 }
13     public String info() {
14         return super.info() + getBaseSalary() ;
15 }

```

صنف الموظف (براتب وعمولة) وهو الصنف اين غير مباشر يختلف تطبيقه لدالة احتساب الراتب

**إضافة:** الصنف الابن إذا لم يطبق دوال الأب المجردة، يبقى مجرد ويحتاج لابن جديد يطبق دواله ووقتها يمكن اشتقاق كائنات من الابن الجديد.



**إضافة:** الصنف الذي يتم إعطاءه محدد وصول private يعتبر تقليديا وبالتالي لا يسمح بإعادة تطبيق دواله في الأبناء.



**خطأ برمجي:** محاولة اشتقاق كائن من صنف مجرد.



**تنكير:** الصنف الابن يمكن أن يكون مباشر للأب أو غير مباشر أي ابن للابن.



**إضافة:** يتعلق بمفهوم تعدد الأشكال مفاهيم أخرى مثل static class و final class وهي أصناف لا يمكن إعادة تطبيق دوالها في الأبناء.



**توضيح:** هذا الباب يهتم بإعطاء مقدمة مساعدة لأي طالب يرغب في إتقان مفاهيم البرمجة شيئاً فشيئاً التوجه، وهو أسلوب برمجة يحتاج لمساق منفصل لتدريسه. ولذلك أنصح من يرغب باكتساب المزيد من أن يتبع سلسلة محاضرات البرمجة الهدفة على قناة المؤلف على يوتيوب [www.youtube.com/mralfarra1](http://www.youtube.com/mralfarra1) أو من خلال موقعه الإلكتروني.



### 9.1 أمثلة وتدريبات عامة

ملحوظة: جزء من التمارين في هذا الفصل ستجد له إجابات بعد نهاية التمارين ذاتية الحل والجزء الآخر تم تركه لك للمحاولة.

#### 1. استخدم الكلمات المناسبة لتعبئة الفراغات في الجمل التالية:

- أ- محاكاة الواقع على أنه مجموعة من الأشياء وأن كل شيء مكون من صفات وسلوك يتم محاكتهم في البرمجة هو مفهوم \_\_\_\_\_.
- ب- عملية إخفاء طريقة تطبيق السلوك أو العمل داخل الصنف عن الآخرين هو مفهوم \_\_\_\_\_.
- ت- تسعى البرمجة شديدة التوجه إلى تحقيق أربعة مقاهيم هي \_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_.
- ث- استخدام الصنف *JOptionPane* مثلاً على \_\_\_\_\_ و\_\_\_\_\_.
- ج- تجزئة المشكلة الكبيرة إلى مشاكل صغيرة يمكن حلها بشكل منفصل وبسهولة أكبر هو مفهوم \_\_\_\_\_.
- ح- مخطط البيت هو مثال على \_\_\_\_\_ بينما البيت ذاته مثال على \_\_\_\_\_.
- خ- قدرة الأصناف على الاستفادة من عناصر الأصناف الأخرى كأنها جزء منها تسمى \_\_\_\_\_.
- د- من مميزات الوراثة كلا من \_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_.
- ذ- يمكن الاحتياج لتطبيق الوراثة في مجالات مختلفة مثل \_\_\_\_\_ و\_\_\_\_\_.
- ر- الوراثة في لغات البرمجة لها نوعين هما \_\_\_\_\_ و\_\_\_\_\_.
- ز- من الأمثلة على الصنف الأب \_\_\_\_\_ و\_\_\_\_\_.
- س- من الأمثلة على الصنف الابن \_\_\_\_\_ و\_\_\_\_\_.
- ش- تستخدم الكلمة المحجوزة \_\_\_\_\_ لتوريث صنف آخر.

#### 2. حدد صحة أو خطأ كلام من العبارات التالية مع التعطيل:

- ( ) أ- الأصناف في البرمجة تخفي بياناتها عن مستخدميها وهذا ما يعرف باسم التجريد
- ب- *Composability* هي قدرة المبرمج من خلال لغات البرمجة التي تدعم شديدة التوجه أن يُنتج عدد من الدوال التي يتم تطبيقها بشكل مختلف يُعرف باسم التحميل الزائد للدوال. ( )

- ت - مفهوم *Composability* هو القدرة على استخدام الأصناف التي تم بناءها سابقاً في إنتاج تطبيقات جديدة مختلفة من خلال إعادة تركيبها وتشكيلها حسب حاجة التطبيق الجديد ( )
- ث - البرمجة شيئاً فشيئاً تتوجه للمشاكل البرمجية على أنها مكونة من مجموعة من الأشياء وأن كل شيء هو مجموعة من الدوال. ( )
- ج - اختلاف اسم الدالة البنائية عن اسم الصنف يعتبر خطأً منطقياً أحياناً وبرمجياً أحياناً أخرى. ( )
- ح - تدعم لغة جافا الوراثة الفردية فقط بينما الثانية تدعمها بطريق غير مباشر. ( )
- خ - الصنف الذي يورث صفاتة لصنف آخر يعرف باسم *super class* ( )
- د - الصنف الذي يرث صفات صنف آخر يعرف باسم الصنف الأساسي ( )
- ذ - عنصر *private* موجود في الصنف الاب، محاولة التفاعل معه في الابن يعطي خطأً منطقياً. ( )
- ر - كافة العناصر المحددة *public* أو *protected* في الاب يمكن لابن التفاعل معها. ( )
- ز - الدالة البنائية للأب يتم استدعاءها ضمنياً في الصنف الابن باستخدام المبرمج. ( )
- س - تستخدم الدوال *set&get* للتفاعل مع دوال الاب من النوع *private*. ( )

3. حول الجمل العربية الآتية إلى جمل يفهمها مترجم جافا:

أ - الصنف *employee* يورث صفاتة للعناصر *part*

ب - الدالة البنائية *Car* تستدعي الدالة البنائية للصنف الأب بشكل صريح.

ت - اكتب دالة *set* لتغيير قيمة مكان السكن.

ث - اكتب دالة *get* لإعادة قيمة المرتب.

4. اكتب صنف برمجي جديد يحاكي فئة السيارات.

5. اكتب صنف برمجي جديد يحاكي فئة الجامعات مع إنشاء كائنات منه وطباعة بياناتها.

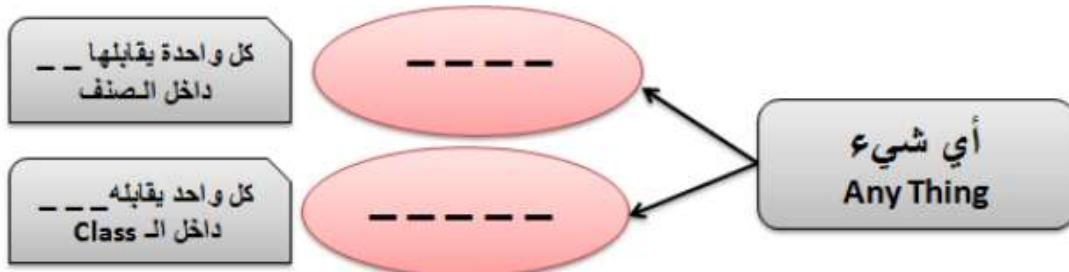
6. أذكر خمسة من الأصناف تصلح لأن تكون (أب) وأذكر أمثلة على أصناف تصلح لتكون (ابن) لها.

7. اكتب برنامجاً باستخدام مفاهيم الوراثة، لمحاكاة صنف الموظفين وموظفي العمل بالساعة، على أن يتم احتساب مرتب موظفي الساعة من خلال معادلة ( المرتب = ساعات العمل \* 15).

#### 9.8 تمارين ذاتية الحل

- استخدم الكلمات المناسبة لتعبئة الفراغات في الجمل التالية:
  - الهدف من تعدد الأشكال يتضح في \_\_\_\_\_ و\_\_\_\_\_ و\_\_\_\_\_.
  - يمكن الاحتياج لتطبيق الوراثة في مجالات مختلفة مثل \_\_\_\_\_ و\_\_\_\_\_.
  - من الأمثلة على تعدد الأشكال \_\_\_\_\_ و\_\_\_\_\_.
  - الدوال التي يمكن عمل استبدال لتطبيقها في الابن محددة بالمحدد \_\_\_\_\_.
  - تطبيق مفهوم Overridden يشترط مماثلة \_\_\_\_\_ في دالة الأب لمثيله في دالة الابن.
  - تستخدم الكلمة المحجوزة \_\_\_\_\_ لمنع الأصناف من اشتقاق كائنات من صنف ما.
- حدد صحة أو خطأ كلاً من العبارات التالية مع التعليل:
  - الدالة البنائية للصنف الأب لا يتم توريثها للصنف الابن.
  - العلاقة "a - is" يتم تطبيقها من خلال التوريث بين الكائن والصنف.
  - الصنف سيارة لها علاقة "has-a" مع الأبواب والزجاج.
  - مفهوم التوارث يشجع على عدم التعاون بين المبرمجين وعدم الثقة في برمجيات الآخرين.
  - تعدد الأشكال يتيح لنا الفرصة لتطوير برمجيات غير قابلة للتتوسيع وإنما للتطوير.
  - يعتبر تعدد الأشكال أكثر ليونة من التوارث.
  - من خلال تعدد الأشكال سيصبح من حق المبرمج التحكم في كيفية تطبيق سلوك الابن.
  - مفهوم تعدد الأشكال والوراثة متعارضين، فلا يجتمعان في برنامج واحد.
  - مفهوم التحميل الزائد للدوال واستبدال تطبيق الدوال كلاهما مرتبط بـ تعدد الأشكال.
  - يشترط للصنف الواحد لكي يصبح مجرد أن يحتوي ثلاثة دوال مجردة.
  - محاولة اشتقاق كائن من الصنف مجرد ينتهي خطأ برمجي.
- ناقش أنماط البرمجة الثلاثة مع ذكر الفارق بينها.
- أنكر مع ضرب الأمثلة مميزات البرمجة شيئاً فشيئاً التوجه.

5. أذكر مع التوضيح مثالين لمفهوم الأصناف مع ذكر ثلاثة كائنات لكل واحد منهم.
6. نقش محددات الوصول مع ذكر الهدف منها.
7. اختر 5 من الأشياء التي تراها حولك الآن وطبق عليها خطوات المثال التوضيحي 9.1
8. قارن بين أنواع الدوال المختلفة في النصف الابن عند تطبيق تعدد الأشكال وعند تطبيق الوراثة.
9. أذكر مثالين مختلفين عن الأمثلة المذكورة في الباب يتم فيهم تطبيق تعدد الأشكال.
10. اقترح ثلاثة دوال يتم تجريدهم ثم عمل استبدال لتطبيقهم **overridden**
11. اكتب صنفًا برمجيًا لثلاث فئات موجودة في حياتك، مع إنشاء كائنات لها وطباعة بياناتها.
12. أكمل الشكل التالي:

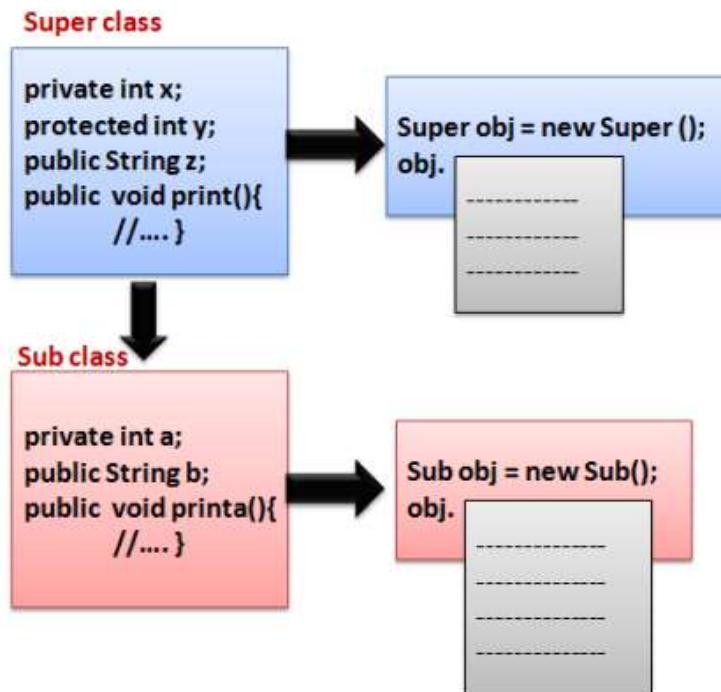


شكل 9.30: تمرين ذاتي 12

13. اكتب برنامج باستخدام مفاهيم الوراثة، تحاكي من خلاله أنواع المركبات المختلفة.
14. اكتب برنامج باستخدام مفاهيم الوراثة، تحاكي من خلاله أنواع الحسابات البنكية.
15. اكتب برنامج لمحاكاة أجهزة الحاسوب المختلفة الأصناف
16. أذكر كيف تتأثر عملية الوراثة بأنواع محددات الوصول مع ضرب أمثلة على كل نوع.
17. بالاستفادة من السؤال رقم 4 من الأمثلة والتدريبات العامة، اختر ثلاثة من الأمثلة على **super class** وابتكر برنامج لكل واحد منها مع توريثه لصنف آخر.

18. اكتب برنامج لتمثيل الأشكال الهندسية مع استخدام تعدد الأشكال في حساب مساحة ومحيط كل شكل من الأشكال. (اختر ثلاثة من الأشكال على الأقل، مع ثلاثة مستويات من الهرمية)

19. أكمل الشكل التالي، بالعناصر التي يمكن أن يتفاعل معها كل من الكائنات الموضحة في الشكل 9.31.



شكل 9.31: تمرين ذاتي 19

#### اجابات التدريبات العامة :

1. (أ) البرمجة شيئاً فشيئاً التوجّه (ب) التجريد، التجريد، التوارث، تعدد الأشكال (ث) التجريد، (ج) التجزئة (ح) الصنف، الكائن (خ) الوراثة (د) إعادة استخدام البرمجيات، توفير الوقت، تسريع إنجاز المهام (ذ) أنظمة الشؤون الأكاديمية، أنظمة الشؤون الإدارية (ر) التوارث الفردي، التوارث المتعدد (ز) الشكل الهندسي، الموظف (س) المربع، الموظف المتفرع (ش) *extends*

2. حدد صحة أو خطأ كلًّا من العبارات التالية مع التعليق:

أ- (خاطئة) يعرف باسم التغليف

ب- (صحيحة)

- ت - (خاطئة) *Composability* هو القدرة على استخدام الأصناف التي تم بناءها سابقاً في إنتاج تطبيقات جديدة مختلفة من خلال إعادة تركيبها وتشكيلها حسب حاجة التطبيق الجديد
- ث - (خاطئة) البرمجة شبيهة التوجه تنظر للتطبيقات أنها مجموعة من الأشياء وأن كل شيء هو مجموعة من الصفات والسلوك.
- ج - (خاطئة) خطأ برمجي
- ح - (صحيحة)
- خ - (صحيحة)
- د - (خاطئة) يُعرف باسم الصنف الآبن.
- ذ - (خاطئة) يُعطي خطأ برمجي لأنّه لا يمكنه رؤيته.
- ر - (صحيحة)
- ز - (خاطئة) الاستدعاء الضمني يكون من خلال المترجم أما الصريح فيكون من خلال المبرمج
- س - (صحيحة)

.4

```
import javax.swing.JOptionPane;
public class car{
    private String name;
    private String Id;
    private int price;
    public car()
    {
        name = "Honda";
        Id = "2000KH";
        price = 16000;
    }
    public void printdata()
```

```
{
    JOptionPane.showMessageDialog(null,"Name "+name+ "Id "+Id+"Price "+price); }
public void update(int pr) {
    price = pr;
    JOptionPane.showMessageDialog(null,"Price is updated"); } }
```

شكل 9.32: حل تمرين 4

```
public class University{
    private String name;
    private int count;
    private String ID;
    private String Tel;
    public University () {
        name = "CST";
        count = 2000;
        ID = "KH-gaza4";
        Tel = "2051786"; }
    public void info() {
        System.out.print("Name: "+name+
            "Count: "+count+ " ID: "+ID+
            "Tel: "+Tel); }}
```

شكل 9.33: حل تمرين 5 (الصنف جامعات)

```
public class UniverProject {
    public static void main(String[] args){
        University u1 = new University();
        u1.info(); }}
```

شكل 9.34: حل تمرين 5 (الصنف الرئيسي)

انتهى الباب

## المراجع

### أولاً، المراجع العربية

نظرًا لندرة المراجع العربية الجيدة، لم يتم الاستعانة من المراجع العربية سوى من كتاب واحد.

- برمجة الحاسب، الإدارية العامة لتصميم وتطوير المناهج، المملكة العربية السعودية

### ثانياً، المراجع الإنجليزية

- Introduction to Java Programming, Y. Daniel Liang, 10'th edition, 2014.
- Java How to Program, Deitel and Deitel, 9'th Edition, 2012.
- Learning Java, By Jonathan Knudsen, Patrick Niemeyer, O'Reilly, 2005

### ثالثاً، المراجع الإلكترونية

- (web site), JDK 0.5, Java Documentation, last visit, 22 October 2015.
- (web site), Stack Overflow, last visit, 4 November 2015.
- (web site), Evan Jones, Adam Marcus, and Eugene Wu. 6.092 Introduction to Programming in Java, January IAP 2010. (Massachusetts Institute of Technology: MIT OpenCourseWare), <http://ocw.mit.edu> (Accessed 4 Nov, 2015). License: Creative Commons BY-NC-SA

## المواد الإلكترونية المساعدة

سيتمكن -بإذن الله- الدارس من هذا الكتاب بتوفّر عدد من المواد الإلكترونية المساعدة والتي تم الإشارة لها في صفحة الأهداف لكل باب من أبواب الكتاب، وهذه المواد تم إعدادها وتوفيرها من مؤلف الكتاب عبر منصات وموقع إلكتروني، حيث يتم توفير المواد التالية:

1. العروض التقديمية للأبواب من الثاني حتى التاسع. [40 ملف إلكتروني]
2. محاضرات بالصوت والصورة للمؤلف للأبواب من الثاني حتى التاسع. [60 ملف مرنى]
3. نماذج من امتحانات سابقة. [10 نماذج إلكترونية]

**كافة هذه المواد متوفّرة عبر الروابط التالية:**

- الموقع الأكاديمي للمؤلف <http://mfarra.cst.ps> – صفحة الكتاب.
- المحاضرات المرنى عبر قناة يوتيوب الخاصة بالمؤلف [www.youtube.com/mralfarra1](https://www.youtube.com/mralfarra1)
- ✓ سلسلة محاضرات الخوارزميات ومبادئ البرمجة.
- ✓ سلسلة محاضرات البرمجة الهدفية.