



بسم الله الرحمن الرحيم

ملخص مقرر جافا





مقدمة عن لغات البرمجة

• لغات البرمجة تقسم إلى ثلاث مستويات:

1. **لغة الآلة**: هي اللغة الوحيدة التي يفهمها المعالج (CPU). وهي تعتمد على التشفير الثنائي (0 و 1). كل نوع معالج له لغة الآلة الخاصة به.

2. **لغة التجميع (Assembly)**: لغة أقرب للإنسان من لغة الآلة، وتستخدم رموز لتمثيل العمليات بدلاً من الأرقام الثنائية. مثال: بدلاً من كتابة "10110011", نستخدم "MV" لنقل البيانات من الذاكرة إلى المعالج.

3. **اللغات عالية المستوى**: مثل لغة جافا، التي تساعي المبرمجين على كتابة البرامج بشكل أسرع باستخدام تعايير رياضية وصيغة أبسط.

التعريفات:

- **البرنامج (Program)**: مجموعة من التعليمات التي ينفذها الحاسوب.
- **تنفيذ البرنامج (Program Execution)**: عملية تنفيذ التعليمات الموجودة في البرنامج عبر تغذيتها إلى وحدة المعالجة المركزية.
- **لغة البرمجة (Programming Language)**: مجموعة من القواعد المستخدمة لوصف العمليات الحسابية بطريقة مفهومة وقابلة للتتعديل من قبل الإنسان.



فيه لغات برمجة كثيرة عالية المستوى زي Java, C++, C, Basic, Fortran, Cobol, Lisp, Perl, Prolog, Eiffel, Python كل لغة تختلف عن الثانية في القدرات، لكن كلها لازم يكون فيها:

- **تصريحات**: عشان تكتب الأوامر اللي تبي الحاسب ينفذها.
- **الشروط**: عشان تنفذ أوامر معينة إذا تحقق شرط معين.
- **التكرار**: إذا تبي تكرر تنفيذ نفس الأوامر أكثر من مرة.

المترجم (Compiler) هو البرنامج اللي ياخذ الكود اللي كتبته بلغة زي جافا ويحوله إلى لغة الآلة اللي يقدر المعالج يفهمها وينفذها.

وش هي جافا؟

جافا هي لغة موجهة للકائنات، يعني تعتمد على فكرة "الكائنات" في البرمجة. حصلت على اهتمام كبير سواء في الصناعة أو الجامعات. اللغة جافا مبنية على لغات زي C و C++, وكانت بالبداية مخصصة لكتابة برامج تحكم في الأجهزة مثل الميكروويف والتلوستر. لكن الآن، جافا صارت تعتبر لغة برمجة للويب لأنها تستخدم كثير في كتابة تطبيقات وبرامج الويب.

مثال بسيط لبرنامج مكتوب بجافا

Hello.java

```
public class Hello {  
    public static void main (String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```



• **Hello**: هنا نعرف فئة (`Class`) جديدة اسمها `public class Hello`

الفئة (`class`) هي الأساس في البرمجة بلغة جافا، وتحتوي على مجموعة من الخصائص والوظائف التي يمكن تستخدامها.

• `public static void main(String[] args)` هذه هي النقطة التي يبدأ منها البرنامج. أي برنامج جافا لازم يحتوي على دالة `main` لأنها المدخل الأساسي التي ينفذ الأوامر التي داخله.

• `System.out.println("Hello World!");` هذه التعليمية وظيفتها طباعة النص التي بين القوسين على الشاشة. في هذه الحالة، رج يطبع `"Hello World!"` على الشاشة.

نصائح لكتابة الكود في جافا:

1. الحساسية لحالة الأحرف: (Case Sensitivity)

- لغة جافا حساسة لحالة الأحرف. هذا يعني إن الفرق بين الأحرف الكبيرة والصغيرة مهم.
- على سبيل المثال:

▪ `System` بحرف كبير ("S") هو شيء مختلف عن `system` (بحرف صغير ("s"))

◦ لازم تكون دقيق في كتابة الأسماء والمتغيرات بنفس الطريقة التي عرفتها.

2. المسافات: (Spaces)

◦ المسافات بين الأوامر والأقواس مو مهمة لجافا من ناحية التشغيل، لكنها تسهل عليك قراءة الكود وفهمه.

◦ مثال:

▪ تقدر تكتب: `System.out.println("Hello World!");`

▪ أو تكتبها كذا `System . out . println ("Hello World!");`

▪ لكن الخيار الأول هو الأفضل لأنه مرتب وأسهل للقراءة.



- دائم حافظ على تنسيق مرتب للكود باستخدام مسافات وتبويبات واضحة بين الأسطر.

3. التبويب:(Indentation)

- استخدم التبويب (عن طريق الضغط على زر Tab أو أربع مسافات) داخل الكود عشان تكون الأكواد المتداخلة واضحة.

4. التسمية:(Naming)

- اختيار أسماء واضحة ومعبرة للفئات(classes) ، المتغيرات، والدوال.

- مثلا، بدل ما تسمى المتغير x، سمه numberOfStudents عشان يكون واضح وش يقصد.

5. التعليقات:(Comments)

- استخدم التعليقات داخل الكود عشان تشرح الأجزاء المعقدة، وهذا يساعدك مستقبلاً إذا رجعت للكود، أو إذا أحد غيرك يقرأ الكود.

- التعليقات تبدأ ب // للسطر الواحد أو /* ... */ للتعليقات المتعددة الأسطر.

6. الأقواس:

- في جافا، كل تعليمة أو أمر داخل دالة لازم ينتهي بفاصلة منقوطة .;

- الأقواس {} تستخدم لاحتواء مجموعة من الأوامر. تأكد دائم إنك قفلت القوس اللي فتحته عشان تتجنب الأخطاء.

تعريفات إضافية:

• الكود أو الشيفرة (code or source code):

والتعليمات اللي تكتبها في البرنامج. مثلا، الكود اللي شفناه في المثال السابق يطلب من الكمبيوتر طباعة رسالة "Hello, world!" على الشاشة.

• الإخراج (output):

هو الرسائل أو النتائج اللي يطبعها البرنامج على الشاشة أو يعرضها للمستخدم.



- **الكونسول : (console)** هو النافذة أو المربع النصي اللي تطلع فيه النتائج أو الرسائل اللي يطبعها البرنامج.

الترجمة والتشغيل: (Compiling and Running)

- **المترجم (Compiler)** هو البرنامج اللي يحول الكود المكتوب بلغة برمجة عالية المستوى إلى لغة الآلة. مثلا، يحول الكود المكتوب بلغة جافا إلى bytecode.
- الـ **Bytecode**: هو لغة خاصة بمعالج وهمي (CPU imaginary CPU)، يعني ما يشتغل على معالجات حقيقية بدون ما يتم تحويله مرة ثانية.
- **المفسر (Interpreter)** هو البرنامج اللي يأخذ كل تعليمة أو سطر من الـ bytecode ويحولها إلى لغة الآلة اللي يقدر معالج جهازك بيشغلها، وبعدين ينفذها.
- لما تشغلي برنامج جافا، المترجم يحول الكود إلى bytecode، وبعدين المفسر يحول الـ bytecode إلى تعليمات يفهمها المعالج اللي عندك. في حالي، المعالج اللي أستخدمه هو "Pentium"، فيتم تحويل الكود إلى تعليمات تناسب هذا المعالج.

شرح بنية برامج جافا: (Structure of Java Programs)

فى أي برنامج مكتوب بلغة جافا:

- كل برنامج جافا يتكون من فئة . (class) الفئة تحتوى على:
 - دالة اسمها main. هذه الدالة هي النقطة اللي يبدأ منها البرنامج تنفيذ الأوامر.
 - الأوامر (statements) اللي تكتب داخل دالة الـ main هي اللي ينفذها البرنامج.
- فى البرنامج اللي شفناه قبل، الفئة اسمها Hello، ودالة main تحتوى على تعلیمة System.out.println التي تطبع رسالة على الشاشة.



الدوال الثابتة: (Static Methods)

- **الدوال الثابتة:** static methods هي مجموعة أوامر تكتب تحت اسم معين وتقدر تستدعيها في برنامجك. لما تكتب دالة ثابتة، تقدر تستدعيها في أي مكان داخل البرنامج.

• استخدام الدوال الثابتة يتطلب خطوتين:

1. تعلنها: declare it يعني تكتب الدالة وتعطيها اسم.
2. تستدعيها: call it يعني تستخدم الدالة اللي كتبتها في دالة الـ main أو أي مكان آخر في البرنامج.

• الدوال الثابتة مفيدة لأنها:

- تقسم البرنامج الكبير إلى أجزاء أصغر وأسهل لفهم.
- تقلل التكرار في الكود عن طريق استخدام نفس الدالة أكثر من مرة.

مثال:

```
public class Example {  
    public static void main(String[] args) {  
        printMessage(); // استدعاء الدالة  
    }  
  
    public static void printMessage() {  
        System.out.println("!" + " رسالة من دالة ثابتة");  
    }  
}
```

في المثال هذا، فيه دالة ثابتة اسمها printMessage اللي تطبع رسالة على الشاشة. لما تستدعي الدالة في دالة الـ main، يطبع البرنامج الرسالة.



شرح الدوال الثابتة: (Static Methods)

صيغة كتابة الدالة الثابتة: (Static Method Syntax):

لما تكتب دالة ثابتة في جافا، الصيغة تكون بالشكل التالي:

```
public class <Class Name> {
    public static void <Method Name>() {
        <statements>;
    }
}
```

يعنى:

اسم الفئة اللي تكتب فيها الدالة. • Class Name:

اسم الدالة اللي تبغى تسميهها. • Method Name:

هي الأوامر أو التعليمات اللي تبي الدالة تنفذها. • statements:

مثال:

هنا عندنا دالة ثابتة اسمها printCheer اللي تطبع جملتين على الشاشة:

```
public static void printCheer() {
    System.out.println("Three cheers for Pirates!");
    System.out.println("Huzzah!");
    System.out.println("Huzzah!");
    System.out.println("Huzzah!");
}
```

الدالة تطبع:

"Three cheers for Pirates!" •

"Huzzah!" ثلاث مرات. •



مثال على استخدام الدوال الثابتة:

هنا برنامج جافا يحتوي على دالة main، ودالة ثابتة printCheer اللي شفناها قبل:

```
public class TwoMessages {  
    public static void main(String[] args) {  
        printCheer();  
        System.out.println();  
        printCheer();  
    }  
  
    public static void printCheer() {  
        System.out.println("Three cheers for Pirates!");  
        System.out.println("Huzzah!");  
        System.out.println("Huzzah!");  
        System.out.println("Huzzah!");  
    }  
}
```

- دالة main تستدعي الدالة الثابتة printCheer مرتين. كل مرة تستدعي فيها الدالة، تطبع الرسائل الموجودة داخلها.

ناتج البرنامج:

```
Three cheers for Pirates!  
Huzzah!  
Huzzah!  
Huzzah!  
  
Three cheers for Pirates!  
Huzzah!  
Huzzah!  
Huzzah!
```

ملاحظات:

- استخدام الدوال الثابتة يسهل عليك إعادة استخدام نفس الأوامر أكثر من مرة بدون تكرار الكود.
- تقدر تستدعي الدالة داخل أي مكان في البرنامج طالما إنها موجودة في نفس الفئة أو الفئات المرتبطة.



شرح الاتصال بين الدوال : (Methods Calling Each Other)

في جافا، ممكن للدالة الثابتة (static method) أنها تستدعي دالة ثابتة أخرى. يعني تقدر تخلّي دالة معينة تنفذ أوامر موجودة في دالة ثانية.

مثال:

```
public class TwelveDays {
    public static void main(String[] args) {
        استدعاء الدالة الأولى // day1();
        day2(); // استدعاء الدالة الثانية
    }

    public static void day1() {
        System.out.println("A partridge in a pear tree.");
    }

    public static void day2() {
        System.out.println("Two turtle doves, and");
        day1(); // استدعاء الدالة الأولى داخل الدالة الثانية
    }
}
```

- الدالة `day2` استدعت الدالة `day1` ثم `main`.
- الدالة `day2` استدعت الدالة `day1` داخلها.

ناتج البرنامج:

```
A partridge in a pear tree.
Two turtle doves, and
A partridge in a pear tree.
```

زي ما شفت، لما استدعيت `day2`، هي بدورها استدعت `day1`، وهذا اللي خلى الرسالة "A partridge in a pear tree." تنطبع مرتين.



تدفق التحكم بين الدوال: (Control Flow of Methods)

لما تستدعي دالة في جافا، البرنامج "يقفز" إلى داخل الدالة وينفذ كل الأوامر اللي فيها، وبعدين يرجع لمكانه اللي بدأ منه.

مثال:

في البرنامج السابق، لما دالة day2 استدعت: day1

- البرنامج أول شي نفذ الأوامر في day2.
- وبعدين "قفز" إلى day1 ونفذ الأوامر اللي فيها.
- بعدها رجع وأكمل باقي الأوامر في day2.

ملاحظات:

تقدير تستفيد من الاتصال بين الدوال لتقسيم العمليات الكبيرة إلى أجزاء أصغر وأبسط.

كل دالة تقدير تستدعي دوال أخرى داخلها بدون تكرار الكود، وبهذا يكون الكود أكثر تنظيماً وسهل القراءة.

شرح الـ Identifiers المعرفات:

المعرف (identifier) هو الاسم اللي نعطيه لجزء من البيانات أو جزء من البرنامج.

- المعرفات مهمة لأنها تسمح لنا بالإشارة للبيانات أو الأوامر في البرنامج لاحقاً.
- المعرفات تعطي أسماء للأشياء التالية:

- الفئات (classes)
- الدوال (methods)
- المتغيرات (variables)

على سبيل المثال: الاسم اللي تعطيه للدالة الثابتة هو نوع من أنواع المعرفات.

تفاصيل حول المعرفات:

في جافا، الأسماء اللي تعطيها للمعرفات لازم تتبع قواعد معينة:

- أول حرف لازم يكون حرف، أو _، أو \$.
- باقي الأحرف ممكن تكون أي حرف أو رقم.
- المعرفات حساسة لحالة الأحرف: يعني name يختلف عن Name.



أمثلة على معرفات صحيحة وغير صحيحة:

- أمثلة على معرفات صحيحة:

olivia ○
second_place ○
_myName ○
\$variable ○
TheCure ○
ANSWER_IS_42 ○

- أمثلة على معرفات غير صحيحة:

me+u ○
:-) ○
side-swipe ○
2%milk ○
kelly@yahoo.com ○

مثال على المتغيرات (Variable Example):

في هذا المثال، نعلن عن متغيرين var1 و var2

```
class Example2 {  
    public static void main(String[] args) {  
        int var1; // تعرف المتغير var1  
        int var2; // تعرف المتغير var2  
  
        var1 = 1024; // إعطاء قيمة 1024 للمتغير var1  
        System.out.println("var1 contains " + var1);  
  
        var2 = var1 / 2;  
        System.out.print("var2 contains var1 / 2: ");  
        System.out.println(var2);  
    }  
}
```



- هنا نستخدم المتغير `var1` ونعطيه القيمة 1024.
 - بعدين نقسم `var1` على 2 ونخزن النتيجة في `var2`.
- النتيجة المتوقعة:

```
var2 contains var1 / 2: 512
```

ملاحظات:

- المتغيرات هي أجزاء من البيانات نستخدمها لتخزين القيم في البرنامج.
 - النوع `int` هو اختصار لـ "integer" ويستخدم لتخزين الأرقام الصحيحة فقط.
- التعليقات في جافا:

- **التعليقات (Comments)**: هي نصوص نكتبها داخل الكود عشان نشرح أجزاء معينة من البرنامج، مثل شرح الأهداف أو شرح معنى بعض الأوامر. التعليقات ما ينفذها الحاسوب، لكنها مفيدة للمبرمجين عشان يفهموا الكود.

أنواع التعليقات في جافا:

1. تعليقات متعددة الأسطر:

- تستخدم لكتابة تعليقات على عدة أسطر.
- تبدأ ب `/*` وتنتهي ب `*/`.
- مثال:

```
/*
  هذا تعليق مكون من
  عدة أسطر
*/
```

تعليقات سطر واحد:

- تستخدم للتعليق على سطر واحد.
- تبدأ ب `//`.
- مثال:

- هذا تعليق مكون من سطر واحد `//`



مثال برمجي يحتوى على تعليقات:

```
/*
برنامج مثال من الفصل 2: عرض نافذة
الملف : Ch2Sample1.java
*/
import javax.swing.*;

class Ch2Sample1 {
    public static void main(String[] args) {
        JFrame myWindow; // تعریف نافذة جديدة

        myWindow = new JFrame(); // إنشاء نافذة جديدة
        myWindow.setSize(300, 200); // ضبط حجم النافذة
        myWindow.setTitle("My First Java Program"); // تعيین عنوان النافذة
        myWindow.setVisible(true); // إظهار النافذة
    }
}
```

في هذا البرنامج:

- تم استخدام **التعليقات متعددة الأسطر** في بداية الكود لشرح اسم الملف وما يقوم به البرنامج.
- التعليقات الموجودة بجانب الأكواد هي **تعليقات سطر واحد** لشرح ما تقوم به كل تعليمية.

ملاحظات:

- استخدام التعليقات في الكود يسهل عليك وعلى غيرك فهمه، خاصة إذا رجعت لل kod بعد فترة.
- دائم استخدام التعليقات لشرح الأجزاء المعقدة أو الغامضة من الكود.



أنواع المتغيرات في جافا: (Java - Variable Types)

- المتغير في جافا يوفر لنا وسيلة لتخزين البيانات التي يستطيع البرنامج التعامل معها. كل متغير في جافا يجب أن يكون له نوع معين.
- صيغة تعريف المتغير:

```
dataType variable [= value][, variable [= value] ...];
```

- dataType هو نوع البيانات التي سيخزنها المتغير) مثل int للأعداد الصحيحة، أو double للأرقام العشرية.
- variable هو اسم المتغير.
- value هي القيمة التي يمكن إسنادها إلى المتغير عند تعريفه (اختياري).
- مثال:

```
int a, b, c;           // int
int a = 10, b = 10;    // و a إسناد قيمة 10 للمتغيرين b
byte B = 22;          // و إعطائه القيمة 22 byte
double pi = 3.14159;  // و إعطائه قيمة double تعريف متغير من النوع pi
char a = 'a';         // و إعطائه القيمة 'a' تعريف متغير من النوع char
```

العوامل في جافا: (Operators in Java)

- هناك أنواع مختلفة من العوامل التي يمكن استخدامها في جافا:
 - العوامل الحسابية: (The Arithmetic Operators) تستخدم لإجراء العمليات الحسابية مثل الجمع والطرح والضرب والقسمة.
 - العوامل العلاقة: (The Relational Operators) تستخدم لمقارنة القيم (مثل أكبر من أو يساوي).
 - العوامل المنطقية: (The Logical Operators) تستخدم لتنفيذ العمليات المنطقية (Mثل AND, OR).
 - عوامل الإسناد: (The Assignment Operators) تستخدم لإعطاء قيم للمتغيرات (مثل =).



العوامل الحسابية: (The Arithmetic Operators)

في جافا، العوامل الحسابية هي اللي نستخدمها عشان نسوى العمليات الرياضية مثل الجمع والطرح والقسمة. خلينا نفترض إنه عندنا متغيرين، الأول A قيمته 10، والثاني B قيمته 20. ونشوف كيف تشتغل العوامل الحسابية:

1. **الجمع: (+)** يضيف القيمتين على يمين ويسار العامل.

$A + B$ يعطي 30.

2. **الطرح: (-)** يطرح القيمة اللي على اليمين من اللي على اليسار.

$B - A$ يعطي -10.

3. **الضرب: (*)** يضرب القيمتين اللي على يمين ويسار العامل.

$A * B$ يعطي 200.

4. **القسمة: (/)** يقسم القيمة اللي على اليسار على اللي على اليمين.

B / A يعطي 2.

5. **الباقي: (%)** يقسم القيمة اللي على اليسار على اليمين ويرجع الباقي.

$B \% A$ يعطي 0 (لأن 20 يقبل القسمة على 10 بدون باقي).

6. **زيادة واحد: (A++)** يزيد قيمة المتغير بمقدار واحد.

$A++$ يعطي 21 يعني A كانت 20 وزادتها وحدة

7. **نقص واحد: (A--)** ينقص قيمة المتغير بمقدار واحد.

$B--$ يعطي 19 يعني B كانت 20 ونقصتها وحدة

العوامل العلاقة: (The Relational Operators)

العوامل العلاقة هي اللي نستخدمها عشان نقارن بين قيمتين. نفس المثال، عندنا $A = 10$ و $B = 20$.

1. **يساوي: (==)** يقارن إذا كانت القيمتين متساويتين. إذا كان صحيح، يرجع true، وإنما خطأ يرجع false.

$A == B$ يرجع false لأن 10 لا تساوي 20

2. **لا يساوي: (!=)** يقارن إذا كانت القيمتين غير متساويتين. إذا صحيح يرجع true.

$A != B$ يرجع true لأن 10 لا تساوي 20

3. **أكبر من: (>)** يقارن إذا كانت القيمة على اليسار أكبر من القيمة على اليمين.

$A > B$ يرجع false لأن 10 أكبر من 20

4. **أصغر من: (<)** يقارن إذا كانت القيمة على اليسار أصغر من القيمة على اليمين.

$A < B$ يرجع true لأن 10 أصغر من 20.



5. أكبر من أو يساوي : ($=>$) يقارن إذا كانت القيمة على اليسار أكبر أو تساوى القيمة على اليمين.

يرجع $A >= B$ لآن 10 ما تساوى ولا أكبر من 20.

6. أصغر من أو يساوي : ($=<$) يقارن إذا كانت القيمة على اليسار أصغر أو تساوى القيمة على اليمين.

يرجع $A <= B$ لآن 10 أصغر من 20

ملاحظات:

- هذه العوامل تستخدم بشكل كبير في جافا لما نحتاج نقارن بين الأرقام أو نستخدم العمليات الرياضية.
- العوامل العلاقة مفيدة في الشروط، مثل تقدر تستخدمها مع جمل if عشان تحدد مسار البرنامج بناءً على المقارنات.

المدخلات والمخرجات القياسية في جافا

• المخرجات القياسية:(Standard Output)

- لما يشتغل برنامج جافا وينتج نتيجة، نحتاج نعرض هذى النتيجة للمستخدم. واحد من أكثر الطرق شيوعاً في جافا هي استخدام نافذة الكونسول.
- نافذة الكونسول هي نفسها اللي نطلق عليها اسم نافذة المخرجات القياسية.
- نقدر نعرض بيانات مثل نتائج العمليات الحسابية أو الرسائل عبر `System.out` اللي تعرض البيانات على نافذة الكونسول.
- فئة `System` تحتوى على مجموعة من الدوال المفيدة اللي نستخدمها للتعامل مع المخرجات.

مثال:

- نستخدم دالة `print` عشان نعرض أي قيمة على الكونسول.

```
System.out.print("King Khalid University");
```

النتيجة:

```
King Khalid University
```



الفرق بين `println` و `print`

- `print` يعرض النص أو القيمة بدون الانتقال إلى سطر جديد. يعني، لما تستخدم `print`، أي شيء تطبعه بعده يجي في نفس السطر.
 - `println` يعرض النص أو القيمة ثم ينتقل مباشرة إلى سطر جديد بعد الطباعة.
أي شيء تطبعه بعده يبدأ من سطر جديد.
- مثال:
- باستخدام `print`:

```
System.out.print("How do you do? ");
System.out.print("My name is ");
System.out.print("your name.");
```

الفرق بين `println` و `print`

- `print` يعرض النص أو القيمة بدون الانتقال إلى سطر جديد. يعني، لما تستخدم `print`، أي شيء تطبعه بعده يجي في نفس السطر.
 - `println` يعرض النص أو القيمة ثم ينتقل مباشرة إلى سطر جديد بعد الطباعة.
أي شيء تطبعه بعده يبدأ من سطر جديد.
- مثال:
- باستخدام `print`:

```
System.out.print("How do you do? ");
System.out.print("My name is ");
System.out.print("Seattle Slew.");
```

الناتج:

```
How do you do? My name is Seattle Slew.
```

- باستخدام `println`:

```
System.out.println("How do you do? ");
System.out.println("My name is ");
System.out.println("Seattle Slew.");
```



الناتج:

```
How do you do?  
My name is  
Seattle Slew.
```

ملاحظات:

- استخدم `println` إذا كنت تبغى تضمن إن البيانات المطبوعة تبدأ من سطر جديد.
- استخدم `print` إذا كنت تبغى تطبع بيانات وراء بعضها في نفس السطر.

درس: إدخال وإخراج البيانات في جافا باستخدام Scanner

في هالدرس، بنعلمك كيف تستخد Scanner في جافا عشان تدخل البرنامج يأخذ بيانات من المستخدم، زي الاسم أو الرقم، ويعرضها على الشاشة باستخدام أوامر `print` و `println` هالطريقة بتدخل البرنامج تفاعلي أكثر ويطلب المدخلات من المستخدم.

Scanner هو ما؟

الـ Scanner هو كلاس (Class) في جافا نستخدمه عشان نقرأ المدخلات من المستخدم، مثل ما لو تبي تأخذ منه اسمه أو رقمه أو أي شي ثاني.

كيف تسوي كائن (Object) من Scanner:

```
Scanner scanner = new Scanner(System.in);
```

- هذا السطر يسوي كائن جديد من Scanner ويذلية يقرأ من لوحة المفاتيح (الكيبورد).



2. كيف تقرأ النصوص من المستخدم؟

عشان تقرأ نص من المستخدم، مثل اسمه الأول واسم العائلة، بنستخدم الدالة `(next())` اللي تقرأ كلمة وحدة من المدخل.

مثال:

```
Scanner scanner = new Scanner(System.in); // إنشاء كائن Scanner

String firstName, lastName; // تعريف متغيرين لتخزين الاسم الأول واسم العائلة

System.out.print("Enter your first name: "); // طباعة رسالة لإدخال الاسم الأول
firstName = scanner.next(); // قراءة الاسم الأول

System.out.print("Enter your last name: "); // طباعة رسالة لإدخال اسم العائلة
lastName = scanner.next(); // قراءة اسم العائلة

System.out.println("Your name is " + firstName + " " + lastName +
"."); // عرض الاسم الكامل
```

شرح:

1. `Scanner scanner = new Scanner(System.in);`

- هنا تسمى كائن من `Scanner` عشان يقرأ البيانات من الكيبورد.

2. `String firstName, lastName;`

- هنا تعرف متغيرين من النوع `String` عشان نخزن فيهم اسم المستخدم واسم العائلة.

3. `System.out.print("Enter your first name: ");`

- نطبع للمستخدم رسالة على الشاشة تطلب منه إدخال اسمه.

4. `firstName = scanner.next();`

- هنا نقرأ الكلمة اللي دخلها المستخدم ونخزنها في المتغير `firstName`.

5. `System.out.print("Enter your last name: ");`

- نطبع للمستخدم رسالة ثانية عشان يدخل اسم العائلة.

6. `lastName = scanner.next();`

- هنا نقرأ اسم العائلة اللي دخلها المستخدم ونخزنها في المتغير `lastName`.



```
System.out.println("Your name is " + firstName + " " + lastName + ".7  
    ");
```

- نطبع للمستخدم اسمه الكامل باستخدام المتغيرات اللي دخلها.

3. الفرق بين `print` و `println`.

- `print`: يعرض النص أو الرقم بدون ما يروح لسطر جديد. كل اللي يطبع بعده يطلع في نفس السطر.
 - `println`: يعرض النص أو الرقم وبعدها ينقل لسطر جديد، وكل اللي بعده يطلع في سطر ثاني.
- مثال:

```
System.out.print("Enter your first name: ");  
System.out.print("John");  
الناتج بيكون:
```

Enter your first name: John
لكن لو استخدمنا
مثال :

```
System.out.println("Enter your first name: ");  
System.out.println("John");  
الناتج بيكون:
```

Enter your first name:
John

4. وش تطلع النتيجة للبرنامج؟

لما تشغل البرنامج اللي فوق، بيطلب منك تدخل اسمك الأول واسم العائلة، ويعرض لك اسمك الكامل بالشكل هذا:

Enter your first name: George

Enter your last name: Washington

Your name is George Washington.



ملخص سريع:

- نستخدم `Scanner` عشان نقرأ بيانات من المستخدم.
- نستخدم `println` و `print` عشان نعرض رسائل على الشاشة، والفرق بينهم إن `println` ينقل لسطر جديد بعد الطباعة.
- لما تشغّل البرنامج، المستخدم يدخل بياناته والبرنامج يعرضها له بعد كذا.

شرح الدرس: قراءة الأعداد الصحيحة والحسابات باستخدام `Scanner` في جافا

1. قراءة الأعداد الصحيحة باستخدام `int`:

في جافا، نقدر نستخدم `Scanner` عشان نطلب من المستخدم يدخل أرقام أو نصوص ونتعامل معها في البرنامج. هنا مثال بسيط لكيفية قراءة عددين صحيحين (`int`). من المستخدم باستخدام النوع `int`.

الكود:

```
Scanner scanner = new Scanner(System.in); // إنشاء كائن Scanner

int num1, num2; // تعریف متغيرین لتخزين الأعداد

System.out.print("Enter two integers: "); // طباعة رسالة للمستخدم لإدخال عددين
num1 = scanner.nextInt(); // قراءة العدد الأول من المستخدم
num2 = scanner.nextInt(); // قراءة العدد الثاني من المستخدم

System.out.println("num1 = " + num1 + " and num2 = " + num2); // طباعة الأعداد المدخلة
```

الشرح:

- `Scanner scanner = new Scanner(System.in);`: هذا السطر ينشئ كائناً من `Scanner` الذي يسمح لنا نقرأ المدخلات من المستخدم عبر الكيبورد.
- `int num1, num2;`: هنا عرفنا متغيرين من نوع `int` عشان نخزن فيهم الأعداد التي يدخلها المستخدم.
- `System.out.print("Enter two integers: ");`: هنا نطبع رسالة تطلب من المستخدم إدخال عددين.
- `num1 = scanner.nextInt();`: القراءة العدد الأول الذي يدخله المستخدم.
- `num2 = scanner.nextInt();`: القراءة العدد الثاني.
- `System.out.println("num1 = " + num1 + " and num2 = " + num2);`: نطبع النتيجة، ونظهر العددين الذي دخلهم المستخدم.



النتيجة عند التنفيذ:

Enter two integers: 12 87

num1 = 12 and num2 = 87

المستخدم يقدر يدخل الأعداد في نفس السطر أو كل عدد في سطر منفصل، والبرنامج يقرأهم.

2. حساب مساحة ومحيط دائرة باستخدام double.

في هذا المثال، بنطلب من المستخدم إدخال نصف قطر دائرة، ونحسب المحيط والمساحة بناءً على القيمة اللي يدخلها باستخدام نوع البيانات double.

الكود:

```
import java.util.*;  
  
class Ch3Sample1 {  
    public static void main(String[] args) {  
        double radius, circumference, area; //  
        //تعريف المتغيرات للمساحة والمحيط  
  
        ChCircle circle; //  
        //تعريف كائن الدائرة  
  
        Scanner scanner = new Scanner(System.in);  
        //طلب إدخال نصف القطر  
        System.out.print("Enter radius: ");  
        radius = scanner.nextDouble(); //  
        //قراءة نصف القطر من المستخدم  
  
        circle = new ChCircle(radius); //  
        //إنشاء كائن الدائرة مع نصف القطر  
        circumference = circle.getCircumference(); //  
        //حساب المحيط  
        area = circle.getArea(); //  
        //حساب المساحة  
  
        System.out.println("Input radius: " + radius); //  
        //عرض نصف القطر الذي دخله المستخدم  
        System.out.println("Circumference: " + circumference); //  
        //عرض المحيط  
        System.out.println("Area: " + area); //  
        //عرض المساحة  
    }  
}
```

الشرح:

double radius, circumference, area;: .1



- عرفنا ثلاثة متغيرات من نوع `double` لتخزين نصف القطر، المحيط، والمساحة.

`ChCircle circle; .2`

- عرفنا كائن `ChCircle` اللي يمثل دائرة. في هذا المثال، نستخدم هذا الكائن عشان نحسب المحيط والمساحة.

`radius = scanner.nextDouble(); .3`

- نطلب من المستخدم إدخال نصف القطر، ونستخدم `(nextDouble()` عشان نقرأها ونعطيها للمتغير `radius`.

`circle = new ChCircle(radius); .4`

- ننشئ كائن الدائرة مع نصف القطر اللي دخله المستخدم.

`circumference = circle.getCircumference(); .5`

- نستخدم دالة `(getCircumference()` اللي موجودة في الكائن `circle` لحساب محيط الدائرة.

`area = circle.getArea(); .6`

- نستخدم دالة `(getArea()` اللي موجودة في الكائن `circle` لحساب مساحة الدائرة.

`System.out.println(); .7`

- نطبع للمستخدم النتائج مثل نصف القطر، المحيط، والمساحة.

الخلاصة:

- تعلمنا في هذا الدرس كيف نستخدم `Scanner` لقراءة أعداد صحيحة (`integers`) باستخدام `nextInt()`.
- شفنا مثال ثانى لكيفية قراءة أعداد عشرية (`double`) لحساب محيط ومساحة دائرة باستخدام `(nextDouble()`.
- عرفنا الفرق بين `int` و `double` واستخدامهم في البرمجة.



شرح هيكلة التحكم (Control Structure) في جافا: الجملة الشرطية if.

الجملة الشرطية **if** تستخدم عشان تنفذ كود معين إذا تحقق شرط معين. إذا كان الشرط صحيح، ينفذ البرنامج الأوامر داخل الكود.

الصيغة العامة للجملة الشرطية if:

```
if (condition) {  
    الكود اللي ينفذ إذا كان الشرط صحيح //  
}
```

مثال بسيط:

```
if (wish == true) {  
    System.out.println("Hi!");  
}
```

هنا إذا كانت قيمة المتغير **wish** صحيحة(true)، البرنامج يطبع "Hi!"

2. الجملة الشرطية if-else.

الجملة if-else تستخدم عشان تنفذ كود معين إذا تحقق الشرط، وإذا ما تحقق الشرط تنفذ كود آخر موجود في **else**.

الصيغة العامة للجملة if-else:

```
if (condition) {  
    الكود اللي ينفذ إذا كان الشرط صحيح //  
} else {  
    الكود اللي ينفذ إذا كان الشرط غير صحيح //  
}  
مثال على العدد الفردي والعدد الزوجي  
java  
نسخ الكود  
if (num % 2 == 0) {  
    System.out.println("Number is even"); //  
} else {  
    System.out.println("Number is odd"); //  
}
```



هذا البرنامج يحدد إذا كان العدد المدخل زوجي أو فردي باستخدام عملية القسمة على 2.

3. الجملة الشرطية if-else مع مثال علامة الطالب:

هذا المثال يوضح كيف نستخدم if-else للتحقق إذا كانت علامة الطالب أقل من 70 أو أكبر:

الكود:

```
Scanner scanner = new Scanner(System.in);
System.out.print("Enter test score: ");
int testScore = scanner.nextInt();

if (testScore < 70) {
    System.out.println("You did not pass");
} else {
    System.out.println("You did pass");
}
```

شرح الكود:

1. نطلب من المستخدم إدخال العلامة.
2. إذا كانت العلامة أقل من 70، يطبع "You did not pass" لم تنجح.
3. إذا كانت العلامة 70 أو أكثر، يطبع "You did pass" نجحت.

4. العمليات المنطقية:

الجملة الشرطية تستخدم العمليات المنطقية للتحقق من عدة شروط. هذه العمليات تشمل:

- < أقل من
- > أكبر من
- <= أقل من أو يساوي
- >= أكبر من أو يساوي
- == يساوي
- != إلا يساوي



مثال:

```
if (a + b <= c) {  
    System.out.println("The sum of a and b is less than or equal to  
c");  
}
```

5. جمل الشرط المركبة:

يمكننا استخدام **if-else** مع جمل شرطية مركبة للتحقق من عدة شروط وتقديم ردود مختلفة لكل شرط.

مثال:

```
if (testScore < 70) {  
    System.out.println("You did not pass");  
} else if (testScore >= 70 && testScore < 90) {  
    System.out.println("You did pass");  
    System.out.println("Good job!");  
} else {  
    System.out.println("Excellent score!");  
}
```

شرح الكود:

1. إذا كانت العلامة أقل من 70، يطبع البرنامج "You did not pass".
2. إذا كانت العلامة بين 70 و 89، يطبع "You did pass" و "Good job!".
3. إذا كانت العلامة 90 أو أكثر، يطبع "Excellent score!".

ملخص:

- الجمل الشرطية تساعدنا في تنفيذ كود معين بناءً على شروط محددة.
- **if-else** تساعد في اتخاذ قرارات متعددة بناءً على تحقق الشروط.
- يمكننا استخدام العمليات المنطقية لتحسين التحقق من الشروط.



هيكلة التحكم (Control Structures) وحلقات التكرار (Loops) في جافا:

1. الجمل الشرطية (Conditional Statements):

الجمل الشرطية هي نوع من هيكلة التحكم المستخدمة في البرمجة لاتخاذ قرارات بناءً على شروط معينة. عندما تواجه شرطًا، يمكن أن يكون هناك مسار مختلف يتبع بناءً على ما إذا كان هذا الشرط صحيحاً أو خاطئاً. الجمل الشرطية تسمح لنا بكتابة برامج تتفاعل مع مدخلات المستخدم أو البيانات الممتدة بطريقة مرنّة وذكية.

الجملة الشرطية if:

الجملة الشرطية if هي أبسط أنواع الجمل الشرطية. هيكل الجملة الشرطية if يقوم بتنفيذ الكود فقط إذا كان الشرط صحيحاً. إذا لم يتحقق الشرط، يتخطى البرنامج الكود داخل الجملة.

النظرية هنا تتعلق بفكرة "القرار الشرطي". بمعنى، يتم اتخاذ القرار بناءً على "شرط" يقيم على أنه صحيح أو خاطئ. إذا كان صحيحاً، ينفذ القرار، وإنما كان خاطئاً، يتجاهل.

الجملة الشرطية if-else:

الجملة if-else تعطى خياراً بديلاً. إذا كان الشرط الأول غير صحيح، يتم تنفيذ الكود في الجملة else. من هنا يمكننا فهم أنه في البرمجة لا نحتاج دائمًا إلى مسار واحد؛ يمكننا بناء سيناريوهات معقدة تعامل مع عدة حالات مختلفة بناءً على المدخلات أو المتغيرات.

الداخل في الجمل الشرطية (Nested if):

في بعض الحالات، قد نحتاج إلى شروط إضافية تعتمد على نتيجة شرط آخر. هنا يأتي مفهوم التداخل. النظرية هنا هي أن لكل جملة شرطية يمكن أن يكون هناك شرط آخر يعتمد عليها، مما يعطي البرنامج القدرة على التعامل مع قرارات معقدة تعتمد على مجموعة من الشروط.

2. استخدام جمل if المتعددة (Multiple if-else statements):

النظرية هنا تتعلق بفكرة "التفرع" في القرارات. في الحياة الواقعية، نواجه مواقف تحتاج إلى أكثر من مجرد قرار بسيط نعم أو لا. نحتاج إلى التفرع إلى عدة قرارات بناءً على شروط مختلفة. البرمجة ليست مختلفة؛ نحتاج إلى تقديم عدة خيارات بناءً على القيم المختلفة للمدخلات. هذا ما تقدمه لنا جمل if-else المتعددة.

الجمل الشرطية المتعددة تساعد على جعل البرمجة أكثر واقعية. على سبيل المثال، عند التعامل مع درجات الطالب، لا يكفي فقط التحقق إذا ما كان الطالب ناجحاً أو راسباً؛



نحتاج أيضاً إلى تصنيف أدائه (ممتاز، جيد جداً، جيد، مقبول، إلخ). هنا نستخدم التفرع بناءً على القيم المختلفة لتقديم النتيجة الصحيحة.

3. الجملة الشرطية :switch

الجملة الشرطية switch تقدم لنا طريقة أخرى لاتخاذ القرارات، ولكنها أكثر تنظيماً عندما يكون لدينا العديد من الخيارات الممكنة بناءً على قيمة معينة. النظرية هنا هي أن switch تعتبر بديلاً أفضل في بعض الأحيان عن if-else المتعددة لأنها تسمح بالتركيز على قيمة واحدة فقط وتحليلها بناءً على عدة حالات محتملة.

عندما نتحدث عن switch، نحن نتحدث عن طريقة "تنظيمية" لاتخاذ القرارات. بدلاً من كتابة العديد من جمل if-else، نستخدم switch التي تتيح لنا مقارنة قيمة معينة مع عدة حالات دفعة واحدة.

من الأمثلة الجيدة على استخدام switch هو تحديد أيام الأسبوع بناءً على قيمة عدديّة. كل يوم يمثل حالة، وعندها نحدد الكود الذي سيتم تنفيذه بناءً على تلك الحالة.

4. حلقات التكرار (Looping Statements) :

حلقات التكرار تمثل جزءاً حيوياً من البرمجة، حيث تسمح لنا باعادة تنفيذ الكود عدة مرات بدون الحاجة إلى إعادة كتابته. النظرية هنا هي فكرة "التكرار". في الحياة الواقعية، نقوم بأشياء متكررة مثل قيادة السيارة كل يوم أو تنظيف البيت أسبوعياً. البرمجة تعامل مع هذه العمليات المتكررة بنفس الطريقة باستخدام الحلقات.

حلقة :for

حلقة for تعتبر من الحلقات البسيطة والمباشرة. النظرية الأساسية خلف هذه الحلقة هي التكرار لعدد محدد مسبقاً من المرات. إذاً كنا نعرف مسبقاً عدد المرات التي نريد فيها تكرار كود معين، فإن for هي الخيار الأفضل.

حلقة :while

حلقة while تقدم مفهوماً أكثر مرونة. النظرية هنا تعتمد على "التكرار المشروط". أي، يتم تكرار الكود طالما الشرط صحيح. هذه الحلقة لا تعتمد على عدد المرات المحددة مسبقاً، بل تعتمد على شرط معين قد يتغير خلال التنفيذ.

حلقة :do-while

حلقة do-while تقدم لنا طريقة مختلفة قليلاً للتعامل مع التكرار. النظرية هنا هي أنها تضمن أن يتم تنفيذ الكود مرة واحدة على الأقل حتى إذا كان الشرط خاطئاً من البداية. هذا يجعلها مفيدة في الحالات التي تحتاج فيها إلى تشغيل الكود مرة واحدة قبل التحقق من الشرط.



أمثلة عملية مختصرة:
الجملة الشرطية1.

```
int temperature = 35;
if (temperature > 30) {
    System.out.println("الجو حار");
}
```

الجملة الشرطية2. if-else:

```
int age = 16;
if (age >= 18) {
    System.out.println(" تستطيع التصويت ");
} else {
    System.out.println(" لا يمكنك التصويت ");
}
```

الجملة الشرطية المتداخلة3. (Nested if).

```
int score = 85;
if (score > 90) {
    System.out.println("ممتاز");
} else if (score >= 70) {
    System.out.println("جيد جداً");
} else {
    System.out.println("لم تنجح");
}
```



switch: جملة 4.

```
int day = 3;
switch(day) {
    case 1: System.out.println("الأحد"); break;
    case 2: System.out.println("الاثنين"); break;
    case 3: System.out.println("الثلاثاء"); break;
    default: System.out.println("رقم غير صحيح");
}
5. حلقة for:
java
نسخ الكود
for (int i = 1; i <= 5; i++) {
    System.out.println(i);
}
```

while: حلقة 6.

```
int count = 1;
while (count <= 3) {
    System.out.println(count);
    count++;
}
```

do-while: حلقة 7.

```
int count = 1;
do {
    System.out.println("مرحباً");
    count++;
} while (count <= 3);
```

8. جمع الأرقام باستخدام while:

```
int sum = 0, num = 1;
while (num <= 100) {
    sum += num;
    num++;
}
System.out.println("المجموع: " + sum);
```



مفهوم المصفوفات (Arrays) والمجموعات (Collections) في جافا: المصفوفات: (Arrays)

المصفوفة في جافا، هي ذي صندوق ترتيب فيه مجموعة عناصر من نفس النوع، وكل عنصر له رقم ترتيبى (index). يعني لو عندك مثل 10 درجات لطلاب، تقدر تحطthem فى مصفوفة وتستخدم الرقم الترتيبى عشان تجيب الدرجة المطلوبة. بس تذكر، دجم المصفوفة ثابت يعني من أول ما تحدد الحجم ما تقدر تزيد أو تنقص منه.

النظرية بشكل بسيط:

المصفوفات تسهل عليك تخزين البيانات بشكل مرتب ومريج، بدل ما تعرف 10 متغيرات لكل درجة، تحطthem كلهم فى مصفوفة وتعامل معهم بنفس الطريقة، وتقدر تسوى عمليات مثل الجمع والتكرار بسهولة.

كيف تعرّف مصفوفة:

في جافا، تعرّف المصفوفة كالتالى:

```
int[] grades = new int[10];
```

هنا حددنا مصفوفة من 10 عناصر، كل عنصر منها من نوع int، يعني أرقام صحيحة.

المصفوفات متعددة الأبعاد: (Multidimensional Arrays)

المصفوفة العادية هي بعد واحد، يعني صف واحد من البيانات. بس لو تيغى تخزن بيانات بشكل جدولى، مثل درجات الطلاب فى 3 مواد، هنا تحتاج مصفوفة ثنائية الأبعاد. مصفوفة ثنائية الأبعاد عبارة عن مجموعة صفوف وأعمدة، تخزن فيها البيانات بشكل منظم.

مثال على مصفوفة ثنائية الأبعاد:

```
int[][][] grades = new int[3][4]; // 3 صفوف و4 أعمدة
```

جملة foreach:

جملة **foreach** في جافا تسهلك عملية المرور على كل العناصر في المصفوفة أو المجموعة بدون ما تحتاج تستخدم الفهارس (الأرقام الترتيبية). يعني تقدر تمر على كل عنصر في المصفوفة وتعامل معه مباشرة.

النظرية بشكل عام:



توفر لك أسلوب `foreach` سريع للمرور على العناصر، وخصوصاً إذا كنت ما تبغي تعامل مع الفهارس أو هو مهم بترتيب العناصر.

(Passing Arrays to Methods): تمرير المصفوفات للدوال

زي ما تقدر ترسل قيم عادية للدوال (مثل الأرقام أو النصوص)، تقدر برضه ترسل مصفوفات. يعني لو عندك مصفوفة تبغى تسوى عليها حسابات أو تعديلات معينة، تقدر ترسلها لدالة وتتولى الدالة العمليات المطلوبة.

النظريّة:

تمرير المصفوفات للدوال يخليك تقدر تسوى عمليات معينة على بياناتك بشكل مرتب ومنظم بدون ما تكرر نفس الكود في أكثر من مكان.

(Collections): المجموعات

(Java Collections Framework): المجموعات في جافا

جافا توفر لنا حاجة اسمها المجموعات (Collections)، وهي بني بيانات متقدمة تخليك تعامل مع البيانات بشكل من أكثر من المصفوفات. عندك `ArrayList`، `LinkedList`، `HashMap` وغيرها. هذه المجموعات تعطيك مرونة في إضافة وحذف وتعديل البيانات بدون ما تشيل هم حجم ثابت زي المصفوفة.

النظريّة بشكل عام:

المجموعات تعطيك حل من أكثر من المصفوفات. مثلًا تقدر تضيف عناصر جديدة بسهولة، وتقدر تزحفها بدون ما تحتاج تعيد تعريف الحجم. هنا شيء مره مفيد لما تعامل مع بيانات متغيرة أو متزايدة.

`ArrayList`:

عبارة عن قائمة مرنّة، يعني تقدر تضيف عناصر لها بدون ما تحدد الحجم من البداية. كل ما أضفت عنصر، القائمة تتسع تلقائياً.

النظريّة:

الخيار `ArrayList` لما تكون تشتعل على بيانات متغيرة وتحتاج تضيف أو تحذف عناصر بشكل مستمر. برضه، هي أسهل في الاستخدام من المصفوفة لما تجي للإضافة والحذف.

`LinkedList`:

تشبه `ArrayList` بس الفرق إنها تعتمد على الروابط بين العناصر. هذا الشيء يخليها أسرع في الإضافة والحذف من بداية أو نهاية القائمة، لكن شوي أبطأ في الوصول للعناصر مقارنة بـ `ArrayList`.

النظريّة:



LinkedList مفيدة إذا كنت تشتغل على قائمة وتحتاج تضييف أو تحذف عناصر بشكل متكرر من البداية أو النهاية، بدون ما تتشيل هم الأداء لما تجي للوصول للعناصر بشكل عشوائي.

HashMap:

HashMap تستخدم لتخزين البيانات على شكل مفتاح وقيمة. كل مفتاح له قيمة مرتبطة فيه، يعني تقدر تبحث عن قيمة معينة باستخدام المفتاح.

النظرية:

HashMap مثالية لما تحتاج تخزن بيانات على شكل أزواج (مفتاح-قيمة) وتحتاج سرعة في البحث والوصول للقيمة باستخدام المفتاح. زي دفتر عناوين، تبحث باسم الشخص وتلقي رقمه.

الملخص العام باللهجة العامية:

- **المصفوفات** هي زي صندوق مرتب تحط فيه مجموعة من البيانات من نفس النوع، لكن حجمها ثابت.
- **المجموعات** زي **ArrayList** و **LinkedList** تعطيك مرونة أكبر من المصفوفات لأنك تقدر تضييف وتحذف فيها براحتك.
- **HashMap** تخليلك تخزن بيانات على شكل مفتاح وقيمة، زي ما تسجل اسم شخص ورقمه في دفتر العناوين.

الحمد لله الذي وفقنا لإكمال هذه الملزمة الخاصة بأساسيات لغة البرمجة جافا. من خلال هذه الملزمة، استعرضنا مجموعة من المفاهيم الأساسية التي تمثل حجر الأساس في عالم البرمجة باستخدام جافا.



معظم التعريفات التي وردت في الملزمة:

1. لغة الآلة: (Machine Language)

- هي اللغة الوحيدة التي يفهمها المعالج .(CPU) تعتمد على الشيفرة الثنائية (0 و 1). كل نوع معالج له لغة آلة خاصة به.

2. لغة التجميع: (Assembly Language)

- لغة أقرب للإنسان من لغة الآلة، تستخدم رموز لتمثيل العمليات بدلاً من الأرقام الثنائية. مثال: بدلاً من كتابة "10110011"، نستخدم "MV" لنقل البيانات من الذاكرة إلى المعالج.

3. اللغات عالية المستوى: (High-Level Languages)

- مثل لغة Java، تتيح للمبرمجين كتابة البرامج بشكل أسرع باستخدام تعاير رياضية وصيغ أقرب للبشر.

4. البرنامج: (Program)

- مجموعة من التعليمات التي ينفذها الحاسوب.

5. تنفيذ البرنامج: (Program Execution)

- عملية تنفيذ التعليمات الموجودة في البرنامج عبر تغذيتها إلى وحدة المعالجة المركزية.

6. لغة البرمجة: (Programming Language)

- مجموعة من القواعد المستخدمة لوصف العمليات الحسابية بطريقة مفهومة وقابلة للتتعديل من قبل الإنسان.

7. المصفوفة: (Array)

- هي بنية بيانات (data structure) قيستخدم لتخزين مجموعة من العناصر من نفس النوع في موقع ذاكرة متتابع. حجم المصفوفة ثابت ولا يمكن تغييره بعد إنشائها.

8. المصفوفة متعددة الأبعاد: (Multidimensional Array)

- هي مصفوفة تحتوي على مصفوفات أخرى بداخلها، وقىستخدم لتنظيم البيانات بشكل جدولي (مثل الجداول).

9. جملة foreach:

- هي جملة تستخدم للمرور على جميع عناصر المصفوفة أو المجموعة دون الحاجة لاستخدام الفهارس. تساعده في تبسيط الكود.

ArrayList: .10

- هي قائمة مرنة يمكن إضافة أو حذف عناصر منها بسهولة. حجمها يتغير تلقائياً حسب الحاجة، على عكس المصفوفة التي يكون حجمها ثابتاً.

LinkedList: .11



- تشبه `ArrayList` ولكنها تعتمد على الروابط بين العناصر، مما يجعلها أكثر كفاءة عند إضافة أو حذف العناصر من البداية أو النهاية.

HashMap: .12

- هي بنية بيانات قيستخدم لتخزين البيانات على شكل أزواج (مفتاح-قيمة). كل مفتاح مرتب بقيمة معينة ويمكن البحث عنها بسهولة باستخدام المفتاح.

13. الجملة الشرطية: if

- هيكل يستخدم للتحقق من شرط معين. إذا تحقق الشرط يتم تنفيذ الكود الموجود داخل الجملة.

14. الجملة الشرطية: if-else

- توفر خيارين بناءً على تحقق شرط معين. إذا كان الشرط صحيحاً، ينفذ الكود داخل جملة `if` ، وإذا كان خاطئاً، ينفذ الكود داخل جملة `else`.

15. الجملة الشرطية المتداخلة: (Nested if)

- هي جملة شرطية تحتوي بداخلها على جمل شرطية أخرى، وقىستخدم فى السيناريوهات التي تتطلب اتخاذ قرارات معقدة بناءً على عدة شروط.

16. جملة switch:

- تستخد لاتخاذ قرار بناءً على قيمة معينة ولديها عدة حالات محتملة. كل حالة تمثل قيمة معينة يتم التعامل معها بناءً على تطابق القيمة المدخلة.

17. حلقة for:

- هي حلقة تكرارية تستخد لتنفيذ كود معين عدد محدد مسبقاً من المرات.

18. حلقة while:

- هي حلقة تكرارية تستمر طالما الشرط صحيح، ولا تعتمد على عدد مرات محددة مسبقاً.

19. حلقة do-while:

- مشابهة لحلقة `while` ، ولكنها تضمن تنفيذ الكود على الأقل مرة واحدة حتى لو كان الشرط خاطئاً من البداية.

20. المجموعات: (Collections)

- هي إطار عمل في جافا يوفر مجموعة من البنى المتقدمة للتعامل مع البيانات مثل القوائم (`Lists`) ، الخرائط (`Maps`) ، والمجموعات (`Sets`).

21. المصفوفات كمعاملات: (Passing Arrays to Methods)

- هي عملية إرسال مصفوفة كمعامل إلى دالة في جافا لتتم معالجتها أو تنفيذ عمليات عليها.



Definitions:

1. Machine Language:

- The only language the CPU understands. It operates using binary code (0 and 1). Each type of CPU has its own machine language.

2. Assembly Language:

- A language closer to human understanding than machine language. It uses symbols to represent operations instead of binary numbers. Example: Instead of writing "10110011", we use "MV" to move data from memory to the processor.

3. High-Level Languages:

- Languages like Java that allow programmers to write code faster using mathematical expressions and simpler syntax.

4. Program:

- A set of instructions that are executed by a computer.

5. Program Execution:

- The process of running the instructions contained in a program by feeding them to the CPU.

6. Programming Language:

- A system of rules used to describe computations in a format that humans can edit and understand.

7. Array:

- A data structure used to store a fixed-size collection of elements of the same type, stored in contiguous memory locations. The size of an array is fixed once it is defined.

8. Multidimensional Array:

- An array containing arrays within it, used to organize data in a table-like structure (e.g., two-dimensional arrays for rows and columns).

9. foreach Loop:

- A loop that allows you to iterate over all elements in an array or collection without needing to manage indices. It simplifies code readability.

10. ArrayList:

- A flexible, resizable list that allows elements to be added or removed dynamically, unlike arrays which have a fixed size.



11. **LinkedList:**

- Similar to an ArrayList, but it relies on links between elements, making it more efficient for inserting or removing elements at the beginning or end of the list.

12. **HashMap:**

- A data structure that stores data as key-value pairs. Each key is associated with a value, and you can retrieve the value by looking up its key.

13. **if Statement:**

- A conditional structure that checks if a certain condition is true. If it is, the code inside the statement is executed.

14. **if-else Statement:**

- A conditional structure that provides two outcomes based on whether a condition is true or false. If true, the code inside the if block is executed; if false, the code inside the else block is executed.

15. **Nested if Statement:**

- An if statement within another if statement. It is used for more complex decision-making with multiple conditions.

16. **switch Statement:**

- A conditional structure that selects one of many possible blocks of code to be executed based on the value of a variable.

17. **for Loop:**

- A loop that repeats code a specific number of times, based on a starting point, a condition, and an increment or decrement.

18. **while Loop:**

- A loop that repeats code as long as a given condition is true, without a predefined number of iterations.

19. **do-while Loop:**

- Similar to a while loop, but it guarantees that the loop is executed at least once, even if the condition is false initially.

20. **Collections in Java:**

- An advanced framework in Java that provides flexible and efficient ways to manage data. Examples include lists (Lists), maps (Maps), and sets (Sets).



21. Passing Arrays to Methods:

- A process of sending an array as a parameter to a method in Java, allowing the method to process or manipulate the array's elements.

أرجو من الجميع التكرم بعدم نشر هذه الملزمة بأي شكل من الأشكال أو مشاركتها مع أي شخص آخر بدون إذني، وذلك لأنني بذلت جهداً كبيراً في إعدادها، وأرغب في الحصول على مقابل عادل مقابل هذا العمل.

إذا كان لديكم أي استفسار أو ترغبون في الحصول على الملزمة بشكل رسمي، يمكنكم التواصل معى على رقم الجوال [0558147903].

كما يمكنكم الانضمام إلى قروب الواتساب الخاص بنا للحصول على التحديثات والمحتويات الأخرى المتعلقة بالملزمة:

<https://chat.whatsapp.com/LSJV3NplRla6ErafSBEuls>

أشكركم على تفهمكم ودعمكم

تحياتي،

رابط لدعمى

<https://buymeacoffee.com/fullmark>