بسم اللـهِ الرَحْمَـٰنِ الرَحِيمِ

Introduction to Java programming language Æ

قال تعالى: "إِنْ أَحْسَنتُمْ أَحْسَنتُمْ لِأَنفُسِكُمْ ۖ وَإِنْ أَسَأْتُمْ فَلَهَا"

أصيل قدح | إيمان الغلبان

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

إِنْ أَحْسَنتُمْ أَحْسَنتُمْ لِأَنفُسِكُمْ وَإِنْ أَسَأْتُمْ فَلَهَا

أصيل قدح إيمان الغلبان

بسم الله الرحمن الرحيم

هذا التلخيص هو جهد فردي
ليس تابعً ا لأي جهة، نبتغي به رضى الله عز وجل
قد نصيب وقد نخطئ، فإذا أصبنا، فهو من
.توفيق الله، وإذا أخطأنا، فهو من عندنا

”بِسْمِ اللهِ نَبْدَأُ، وَعَلَى بَرَكَةِ اللهِ نَسِير“

فهرس المواضيع

1. **Programming basics:**
   - Logical operations.
   - comparison operations.
   - If statement.
   - Switch statement.

2. **Java classes:**
   - Math class and its methods.
   - Character class and its methods.
   - string class and its methods.

3. **loops:**
   - While - loops.
   - Do - while loops.
   - For loops.

4.methods.

5. Recursion.

6. Arrays:

- 1 dimensional arrays.
- 2 dimensional arrays.
- For-each loops

8. Stringbuilder, relations,and wrapper classes:

- Stringbuilder and its methods.
- Wrapper classes.
- Big integer.
- Big decimal.
- Association - relation.
- Aggregation - relation.
- Composition - relation.

7. OOP (Object oriented programming)

- Classes.
- Objects.
- Constructors.
- Local variables.
- This keyword.
- Reference equality.
- Arrays of objects.
- Static.
- Modifiers.
- getters/setters methods.
- Primitive and non primitive data types.
- Immutable classes.
- garbage collection.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Conditional Statements: If and Switch

قال تعالى: "وَقُلْ رَبِّ زِدْنِي عِلْماً"

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

أدوات المقارنة (Comparison Operators)

	الأداة	المعنى
1	==	يساوي
2	!=	لا يساوي
3	>	أكبر
4	<	أصغر
5	>=	أكبرأو يساوي
6	<=	أصغرأو يساوي

في لغات البرمجة تُستخدم لمقارنة القيم وتحديد العلاقة بينها.

هذه الأدوات تُعيد نتيجة من نوع boolean، إما True أو False، بناءً على نتيجة المقارنة.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

أدوات المقارنة (Comparison Operators)

Example:

	المثال	النتيجة
1	3==2	false
2	3!=2	true
3	7>7	false
4	1<9	true
5	7>=7	true
6	5<=4	false

في لغات البرمجة تُستخدم لمقارنة القيم وتحديد العلاقة بينها.

هذه الأدوات تُعيد نتيجة من نوع boolean، إما True أو False، بناءً على نتيجة المقارنة.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

العمليات المنطقية (Logical Operators)

	الأداة	المعنى
1	&&	and
2	||	or
3	!	not
4	^	xor

هي أدوات أساسية في البرمجة تُستخدم للتحقق من العلاقات بين القيم والشرطيات. تُستخدم هذه العوامل بشكل رئيسي في تعبيرات الشرطيات واتخاذ القرارات في البرامج.

العمليات المنطقية تُعيد نتيجة من نوع boolean، إما True أو False،

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

أدوات المقارنة (Comparison Operators)

p1	p2	p1&& p2	EX (palestine is free and أصيل is palestinian)
true	true	True	palestine is free and أصيل is palestinian = true
true	false	false	palestine is free and أصيل is not palestinian = false
false	true	false	palestine is not free and أصيل is palestinian = false
false	false	false	palestine is not free and أصيل is not palestinian = false

And (&&)

المعنى:

يُستخدم هذا العامل للتحقق من صحة جميع الشروط المرفقة به. يُعيد True فقط إذا كانت جميع الشروط التي يتم التحقق منها صحيحة.

استخدامه:

غالبًا ما يُستخدم في الجمل الشرطية عندما تحتاج إلى أن يكون أكثر من شرط صحيح لتحقيق حالة معينة.

مثال :

```
( (3>2) && (1==6) =
( true && false ) = false
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

أدوات المقارنة (Comparison Operators)

p1	p2	p1 || p2	EX: (palestine is free or أصيل is palestinian)
true	true	true	palestine is free or أصيل is palestinian = true
true	false	true	palestine is free or أصيل is not palestinian = true
false	true	true	palestine is not free or أصيل is palestinian = true
false	false	false	palestine is not free or أصيل is not palestinian = false

or (||)

المعنى:

يُستخدم هذا العامل للتحقق من صحة أي من الشروط المرفقة به. يُعيد True إذا كان أي من الشروط صحيحًا.

استخدامه:

يُستخدم عندما تكون لديك مجموعة من الشروط، وتريد تنفيذ عملية إذا كان أي من الشروط صحيحًا.

مثال عام:

((3>2) || (1==6) =
(true || false) = true

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

أدوات المقارنة (Comparison Operators)

p1	p2	p1 ^ p2	EX (palestine is free xor أصيل is palestinian)
true	true	false	palestine is free xot أصيل is palestinian = false
true	false	true	palestine is free xor أصيل is not palestinian = true
false	true	true	palestine is not free xor أصيل is palestinian = true
false	false	false	palestine is not free xor أصيل is not palestinian = false

XOR (Exclusive OR):

المعنى:

يُستخدم عامل XOR للتحقق مما إذا كان واحد فقط من الشرطين المرفقين به صحيحًا، وليس كلاهما. يُعيد True إذا كان أحد الشروط صحيحًا والآخر غير صحيح، ويُعيد False إذا كان كلا الشرطين صحيحين أو كلاهما غير صحيحين.

الاستخدام:

يُستخدم عندما تريد تنفيذ عملية فقط إذا كان واحد من الشروط صحيحًا، وليس كلاهما.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

أدوات المقارنة (Comparison Operators)

Example:

7 ^ 5 = ?

5 = 0101

7 = 0111

? = 0010 = 2

عندما تقوم بعمل x^y

مثـ

ـلا

فإنه يقوم بتحويل العدد إلى النظام الثنائي والقيام بعملية xor لكـ
تقابلها، وثم تحويل الناتج إلى
النظام العشري من جديد

XOR (Exclusive OR):

المعنى:

يُستخدم عامل XOR للتحقق مما إذا كان واحد فقط من الشرطين المرفقين به صحيحًا، وليس كلاهما. يُعيد True إذا كان أحد الشروط صحيحًا والآخر غير صحيح، ويُعيد False إذا كان كلا الشرطين صحيحين أو كلاهما غير صحيحين.

الاستخدام:

يُستخدم عندما تريد تنفيذ عملية فقط إذا كان واحد من الشروط صحيحًا، وليس كلاهما.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

If statement

```java
class Main {
   public static void main(String[] args) {
       if (condition) {
           // anything
       }
   }
}
```

في لغة البرمجة JAVA تُستخدم عبارة if كأداة شرطية تُتيح للمبرمجين تنفيذ كود معين فقط إذا تم استيفاء شرط معين.
تعتبر عبارة if من أهم الأدوات في البرمجة لاتخاذ القرارات بناءً على تقييم تعبير منطقي، والذي يُعيد إما true أو false.

فإذا كان ما بداخل الـ if → true , فإن ما بداخل أداة if سيعمل,
وإن كان false , فإنه سيقوم بالمرور عنها

بنية أداة if :

الشرط

الفعل

اذا تحقق الشرط وكانت نتيجته true, فإن ما بداخل الأقواس سيعمل

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

If statement

```java
class Main {
   public static void main(String[] args) {
       int num = 12 ;
       if (num > 10) {
           System.out.println(num+" is greater than 10");
       }
   }
}
```

أمثلة توضيحية :

اكتب لي برنامج يقوم بفحص متغير, اذا كانت قيمته اكبر من 10
يقوم بطباعة: "هذا الرقم أكبر من 10"

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

If statement

```java
public class Main {
   public static void main(String[] args) {
       int num = 12 ;
       if (num >= 10) {
           System.out.println(num + " is greater than 10");
       }
       if (num <= 10) {
           System.out.println(num+" is less than 10" );
       }
   }

}
```

أمثلة توضيحية :

اكتب لي برنامج يقوم بفحص متغير, اذا كانت قيمته اكبر من 10 يقوم بطباعة: "هذا الرقم أكبر من 10" , وإذا لا, يقوم بطباعة "هذا الرقم أقل من 10"

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

If statement

```
class Main {
   public static void main(String[] args) {
       if (condition) {
           // anything
       } else{
           // anything else
       }
   }
}
```

لكن بالمثال السابق نلاحظ أن الشرط الثاني هو عكس الشرط الأول , إذا يمكننا إستخدام أداة if مصحوبة مع else, ففي حالة عدم تحقق الشرط الذي بداخل if, سيتحقق الشرط الذي بداخل else نظرا أنها نظيرة ل if

وهذا الشكل يمثل بنية if مصحوبة ب else

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

If statement

```java
class Main {
   public static void main(String[] args) {
       int num = 2 ;
       if (num > 10) {
           System.out.println(num+" is greater than 10");
       } else{
           System.out.println(num+" is less than 10");
       }
   }
}
```

أمثلة توضيحية :

اكتب لي برنامج يقوم بفحص متغير, اذا كانت قيمته اكبر من 10 يقوم بطباعة: "هذا الرقم أكبر من 10" , وإذا لا, يقوم بطباعة "هذا الرقم أقل من 10"

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

If statement

```
class Main {
   public static void main(String[] args) {
       if (condition1) {
      // الكود الذي سيتم تنفيذه إذا تحقق الشرط الأول
       } else if (condition2) {
      // الكود الذي سيتم تنفيذه إذا تحقق الشرط الثاني
       } else {
      // الكود الذي سيتم تنفيذه إذا لم يتحقق أي من الشروط السابقة
       }
   }
}
```

لكن فرضا أنني أريد أن أضع الكثير من الشروط, هل من المنطق أن أضع أداة if لكل شرط أريد فحصه؟.

الحل هو إستخدام أداة if مصحوبة مع else if, وهذا يجعل البرنامج يتحقق من الشرط الأول, فإن لم يتحقق أريد أن يفحص شرط اخر في نفس الأداة.

وهذه بنية if مصحوبة ب else if

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

If statement

```java
class Main {
   public static void main(String[] args) {
       int num = 12 ;

       if (num > 10){
           System.out.println("Number is greater than 12");
       } else if (num < 10){
           System.out.println("Number is less than 12");
       }else if (num == 10){
           System.out.println("Number is equal to 12");
       }
   }
}
```

أمثلة توضيحية :

اكتب لي برنامج لفحص قيمة رقم وتحليل هل هو أكبر أو أصغر أو يساوي الرقم 10, إطبع لي النتيجة: (أكبر , أصغر , يساوي)

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

If statement

```java
import java.util.Scanner;

class Main {
   public static void main(String[] args) {
       Scanner scan = new Scanner(System.in);
       int mark ;
       mark = scan.nextInt();
       if (mark > 100 && mark < 0){
           System.out.println("Invalid Mark") ;
       }
       else if(mark >= 90 && mark <= 100){
           System.out.println("Aseel Qadah");
       } else if(mark >= 80 && mark < 90){
           System.out.println("very good");
       }else if(mark >= 70 && mark < 80){
           System.out.println("Good");
       } else if(mark >= 60 && mark < 70){
           System.out.println("not good");
       }else {
           System.out.println("ساقط");
       }
   }
}
```

أمثلة توضيحية :

اكتب لي برنامج لإدخال علامة طالب, ثم إعطائه التقدير الخاص بالعلامة بالتنسيق التالي

(Aseel Qadah) 90-100

(very good) 80-89

(good) 70-79

(not good) 60-69

(ساقط) 60>

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

If statement

```java
class Main {
   public static void main(String[] args) {
           int num = 1 ;
           if (num == 2)
               System.out.println("Number is 2");
       System.out.println("the program ends here");
   }
}
```

ملاحظة مهمة! :

عند عدم وضع أقواس معقوفة { } بعد أداة الشرط, فإن الحدث الخاص بالشرط سيكون فقط أول سطر تحت أداة الشرط.

مثال:

مثلا هنا, لم يتم وضع أقواس معقوفة, إذا يعتبر هذا السطر هو الحدث الخاص بالشرط, أما السطر الذي يليه سيتم تنفيذه سواء تحقق الشرط أم لا.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

switch statement

في لغة البرمجة JAVA، تُستخدم أداة switch كوسيلة لاتخاذ القرار عند الحاجة إلى مقارنة قيمة متغير أو تعبير بعدة حالات (قيمة معينة لكل حالة)، بدلاً من استخدام عدة جمل if-else. تعتبر switch مفيدة عندما يكون لديك مجموعة من القيم المحددة مسبقًا وتريد تنفيذ كود مختلف بناءً على تلك القيم.

بنية ال switch :

```
switch (expression) {
    case value1:
        // الكود الذي سيتم تنفيذه إذا كان التعبير يساوي value1
        break;
    case value2:
        // الكود الذي سيتم تنفيذه إذا كان التعبير يساوي value2
        break;
    default:
        // الكود الذي سيتم تنفيذه إذا لم يتطابق التعبير مع أي حالة
}
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

switch statement

مكونات switch:

1. `expression`:
   - هذا هو التعبير الذي يتم تقييمه (عادةً ما يكون متغيرًا من نوع int أو char).
   - يمكن أن يكون نتيجة عملية رياضية أو متغير بسيط.
   - يتم مقارنة هذا التعبير مع كل حالة case.
2. `case value`:
   - يُمثل كل case قيمة محتملة يمكن أن يساويها التعبير.
   - إذا كانت قيمة التعبير تساوي value في case معين، يتم تنفيذ الكود الموجود أسفل هذا case.
3. `break`:
   - يُستخدم لإنهاء تنفيذ الحالة الحالية والخروج من عبارة switch.
   - إذا لم يتم وضع break، فسيستمر تنفيذ الحالات التالية (وهو ما يُعرف بـ "fall-through").
4. `default`:
   - اختياري، ويُستخدم لتحديد الكود الذي سيتم تنفيذه إذا لم تتطابق أي من القيم في الحالات case مع التعبير.
   - يشبه else في جمل if-else.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

switch statement

```java
class Main {
  public static void main(String[] args) {
    int day = 3;
    switch (day) {
      case 1:
        System.out.println("Sunday\n");
        break;
      case 2:
        System.out.println("Monday\n");
        break;
      case 3:
        System.out.println("Tuesday\n");
        break;
      case 4:
        System.out.println("Wednesday\n");
        break;
      case 5:
        System.out.println("Thursday\n");
        break;
      case 6:
        System.out.println("Friday\n");
        break;
      case 7:
        System.out.println("Saturday\n");
        break;
      default:
        System.out.println("Invalid day\n");
    }
  }
}
```

مثال توضيحي:

أكتب لي برنامج لإدخال رقم وطباعة ما يقابله بأيام الأسبوع

- المتغير `day` يحتوي على القيمة 3.
- في جملة `switch`، يتم مقارنة القيمة 3 مع كل حالة `case`.
- يتم تنفيذ الأمر `case 3:` عندما يصل البرنامج إلى الحالة printf("Tuesday\n"); وطباعة "Tuesday".
- بعد ذلك، يتم تنفيذ `break`، مما يخرج من جملة `switch`.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

switch statement

```java
class Main {
   public static void main(String[] args) {
       int day = 3 ;

       switch (day) {
           case 1:
               System.out.println("Sunday\n");
           case 2:
               System.out.println("Monday\n");
           case 3:
               System.out.println("Tuesday\n");
           case 4:
               System.out.println("Wednesday\n");
           default:
               System.out.println("Invalid day\n");
       }
   }
}
```

مفهوم `break` في `switch`:

- إذا لم يتم استخدام `break` بعد حالة معينة، فسيتم تنفيذ جميع الحالات التي تليها بغض النظر عن مطابقة الشروط. يُسمى هذا السلوك بـ "fall-through".

- الشرح:
  - في هذا المثال، إذا كانت قيمة `day` تساوي 3، سيتم طباعة "Tuesday"، ثم تستمر الطباعة حتى يتم تنفيذ "Invalid day" لأننا لم نستخدم `break` بعد كل حالة.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

switch statement

متى نستخدم `switch` ومتى نستخدم `if-else`:

- استخدم `switch` إذا كنت تتعامل مع متغير يمكن أن يأخذ مجموعة محدودة من القيم الثابتة.
- استخدم `if-else` إذا كنت تحتاج إلى التعامل مع تعبيرات شرطية أكثر تعقيدًا (مثل المقارنات الرياضية).

ملاحظات مهمة:

1. `switch` يدعم المقارنة بين أنواع مثل **int** و **char**، لكنه لا يدعم أنواعًا مثل **float** أو **double**.
2. كل حالة `case` يجب أن تكون فريدة ولا يُسمح بتكرار القيم.
3. إذا لم يكن هناك `break` بعد كل حالة، سيتم تنفيذ جميع الحالات التي تليها.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Java Classes

Math, character, and strings

قال تعالى: "إنّا لا نُضيّعُ أجرَ من أحسنَ عَمَلاً"

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Math Class

كلاس Math هو كلاس يوفر مجموعة من العمليات الرياضية الثابتة التي يمكن استخدامها لأداء حسابات رياضية متنوعة. هي جزء من الحزمة java.lang، لذا لا تحتاج إلى استيرادها بشكل صريح في معظم الحالات.

ميزات كلاس Math

- الثوابت الرياضية
- الدوال المثلثية
- الدوال اللوغاريتمية
- دوال التقريب
- دوال أكبر وأصغر

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Math Class

ميزات كلاس Math

- الثوابت الرياضية

- `Math.PI` - قيمة عدد باي (π) (3.141592653589793) بدقة عالية.
- `Math.E` - قيمة العدد النيبيري (e) (2.718281828459045) بدقة عالية.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Math Class

```java
double result1 = Math.sin(Math.PI / 2); // result = 1.0
double result2 = Math.cos(Math.PI); // result = -1.0
double result3 = Math.tan(Math.PI / 4); // result = 1.0
double result4 = Math.asin(1);
double result5 = Math.acos(1);
double result6 = Math.atan(1);
```

ميزات كلاس Math

- الدوال المثلثية

```java
Math.sin(angle)
Math.cos(angle)
Math.tan(angle)
Math.asin(angle)
Math.acos(angle)
Math.atan(angle)
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Math Class

```java
double result1 = Math.log(Math.E); // result = 1.0

double result2 = Math.log10(100); // result = 2.0

double result3 = Math.exp(1); // result = 2.71828182845
```

ميزات كلاس Math

- الدوال اللوغاريتمية

```
Math.log(x)

Math.log10(x)

Math.exp(x)
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Math Class

```java
long result = Math.round(7.5); // result = 8

double result = Math.floor(7.8); // result = 7.0

double result = Math.ceil(7.2); // result = 8.0

double result = Math.rint(7.2); // result = 7.0
```

ميزات كلاس Math

- دوال التقريب

`Math.round(x)`

تعيد العدد الصحيح الأقرب إلى x :

`Math.floor(x)`

تعيد أكبر عدد صحيح أقل من أو يساوي x

`Math.ceil(x)`

تعيد أصغر عدد صحيح أكبر من أو يساوي x

`Math.rint(x)`

تعيد قيمة double القريبة إلى x

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Math Class

```java
int x = Math.max(3,7); // returns 7

int y = Math.min(3,7); // returns 3

int z = Math.abs(-10); // returns 10
```

ميزات كلاس Math

- دوال أكبر وأصغر

```
Math.max(a,b)

Math.min(a,b)

Math.abs(a)
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Question :

Which of the following is not a math class ?

A - Math.pow

B - Math.max

C - Math.sum

D - Math.sin

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Question :

Which of the following is not a math class ?

A - Math.pow

B - Math.max

C - Math.sum

D - Math.sin

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Character and string:

يجب أن نعلم أننا عندما نتعامل مع `char` فإننا نتعامل مع قيمته بالـ `ASCII Code`، وهذه القيم هي الصيغة للحرف التي يتم التعامل معها بواسطة الكمبيوتر، فمثلا عندما نتعامل مع حرف `A` في الواقع نحن نتعامل مع قيمة الحرف A بالـ `ASCII Code`، وهي `65`.

وأمامكم الجدول التالي الذي يعرض قيم الأحرف والرموز بالـ `ASCII`، بصيغة `hex Number`، أي النظام السادي العشري،

مثلا: `A=(41)hex`، وكما تعلمنا سابقا عندما نحول من `hex->Dic`، فإن القيمة تكون `65`

الصف يمثل العشرات،
والعمود يمثل الآحاد فتكون القيمة: hex (41)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	space	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Character and string:

```
(char)  x // return character

(int)  x   // return integar
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Character and string:

Example:

`System.out.println( (char) 'b' );`	`b`
`System.out.println( (int) 'b' );`	`98`
`System.out.println( (int) 65 );`	`65`
`System.out.println( (char) 65 );`	`A`

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Character and string:

Example:

```
System.out.println( (char)('b'+5) );    g

System.out.println( (int)('b'-8) );
90

System.out.println( (int)(65+5);
70

System.out.println( (char)(65+5) );     F
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Character and string:

Example:

```
(int) ( '5' + '2' ) = 7 ?

(int) '2' = 50 // by the ASCII

(int) '5' = 53 // by the ASCII

(int) ( '5' + '2' ) = 103
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Character Class

```java
boolean result = Character.isLetter('a'); // true

boolean result = Character.isLetter('1'); // false

boolean result = Character.isDigit('9'); // true

boolean result = Character.isDigit('a'); // false

boolean result = Character.isLetterOrDigit('a'); // true

boolean result = Character.isLetterOrDigit('9'); // true

boolean result = Character.isLetterOrDigit(' '); // false

boolean result = Character.isUpperCase('A'); // true

boolean result = Character.isUpperCase('a'); // false
```

كلاس Character هو كلاس تغليف (wrapper class) لـ char، ويوفر طرقًا متعددة للتعامل مع أحرف النصوص. يمكن استخدامها للحصول على معلومات حول الأحرف، مثل تحديد ما إذا كان الحرف حرفًا كبيرًا، صغيرًا، رقميًا، أو مسافة.

- `Character.isLetter(char ch)`:

  تعيد `true` إذا كان `ch` هو حرف أبجدي (a-z, A-Z).

- `Character.isDigit(char ch)`:

  تعيد `true` إذا كان `ch` هو رقم (0-9).

- `Character.isUpperCase(char ch)`:

  تعيد `true` إذا كان `ch` هو حرف كبير (A-Z).

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Character Class

```java
boolean result =
Character.isLowerCase('a'); // true

boolean result =
Character.isLowerCase('A'); // false

char result =
Character.toUpperCase('a'); // 'A'

char result =
Character.toUpperCase('A'); // 'A'

char result =
Character.toLowerCase('A'); // 'a'

char result =
Character.toLowerCase('a'); // 'a'
```

كلاس Character هو كلاس تغليف (wrapper class) لـ char، ويوفر طرقًا متعددة للتعامل مع أحرف النصوص. يمكن استخدامها للحصول على معلومات حول الأحرف، مثل تحديد ما إذا كان الحرف حرفًا كبيرًا، صغيرًا، رقميًا، أو مسافة.

- `Character.isLower(char ch)`:

  تعيد `true` إذا كان `ch` هو حرف أبجدي صغير(a-z).

- `Character.toUpperCase(char ch)`:

  تعيد `ch` كحرف كبير.

- `Character.toLowerCase(char ch)`:

  تعيد `ch` كحرف صغير.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String Class

كلاس String في جافا هو واحد من أهم الكلاس في اللغة، حيث يستخدم لتمثيل سلسلة (array of char) من الأحرف . String تُعتبر غير قابلة للتغيير (immutable)، مما يعني أن أي عملية تعديل على سلسلة نصية تُنتج سلسلة جديدة بدلاً من تعديل السلسلة الأصلية.

خصائص كلاس String

1- **غير قابلة للتغيير** (Immutable): بمجرد إنشاء كائن String، لا يمكن تغييره. بدلاً من ذلك، تُنتج عمليات التعديل سلسلة جديدة.
2- **التحجيم**: سلسلة النصوص في جافا يمكن أن تكون بأي طول، من الصفر إلى الطول الأقصى المسموح به في الذاكرة.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String Class
Simple method for String object

	Methods	الشرح
1	`charAt(int index)`	إرجاع الحرف الموجود في الموضع المحدد ضمن السلسلة.
2	`compareTo(String anotherString)`	مقارنة هذه السلسلة مع سلسلة أخرى قاموسيًا.
3	`compareToIgnoreCase(String str)`	مقارنة هذه السلسلة مع سلسلة أخرى قاموسيًا بدون التمييز بين الأحرف الكبيرة والصغيرة.
4	`concat(String str)`	دمج السلسلة المحددة مع هذه السلسلة.
5	`contains(CharSequence s)`	التحقق مما إذا كانت هذه السلسلة تحتوي على التسلسل المحدد من الأحرف.
6	`equals(Object anObject)`	مقارنة هذه السلسلة مع كائن آخر للتحقق من المساواة.
7	`equalsIgnoreCase(String anotherString)`	مقارنة هذه السلسلة مع سلسلة أخرى للتحقق من المساواة بدون التمييز بين الأحرف الكبيرة والصغيرة.

String Class
Simple method for String object

	Methods	الشرح
8	`indexOf(int ch)`	إرجاع فهرس أول ظهور للحرف المحدد في السلسلة.
9	`indexOf(String str)`	إرجاع فهرس أول ظهور للسلسلة الفرعية المحددة في السلسلة.
10	`isEmpty()`	التحقق مما إذا كانت السلسلة فارغة.
11	`lastIndexOf(int ch)`	إرجاع فهرس آخر ظهور للحرف المحدد في السلسلة.
12	`length()`	إرجاع طول السلسلة.
13	`startsWith(String prefix)`	التحقق مما إذا كانت السلسلة تبدأ بالبداية المحددة.
14	`endsWith(String prefix)`	التحقق مما إذا كانت السلسلة تنتهي بالنهاية المحددة.

String Class
Simple method for String object

	Methods	الشرح
15	`substring(int beginIndex)`	إرجاع سلسلة فرعية تبدأ من الفهرس المحدد إلى نهاية السلسلة.
16	`substring(int beginIndex, int endIndex)`	إرجاع سلسلة فرعية من الفهرس المحدد إلى الفهرس الآخر المحدد.
17	`trim()`	إرجاع نسخة من السلسلة بدون المسافات البيضاء في بداية ونهاية السلسلة.
18	`toLowerCase()`	تحويل جميع أحرف السلسلة إلى حروف صغيرة.
19	`toUpperCase()`	تحويل جميع أحرف السلسلة إلى حروف كبيرة.
20	`toCharArray()`	تحويل السلسلة إلى مصفوفة من الأحرف.
21	`valueOf(int i)`	إرجاع تمثيل نصي للقيمة المحددة.

String Class
Simple method for String object

	Methods	الشرح
22	`replace(char oldChar, char newChar)`	استبدال جميع ظهورات الحرف القديم في السلسلة بالحرف الجديد.
23	`replace(CharSequence target, CharSequence replacement)`	استبدال جميع ظهورات التسلسل المستهدف بالتسلسل البديل.
24	`replaceAll(String regex, String replacement)`	استبدال كل ظهورات التعبير العادي بالتسلسل البديل.
25	`replaceFirst(String regex, String replacement)`	استبدال أول ظهور للتعبير العادي بالتسلسل البديل.
26	`split(String regex)`	تقسيم السلسلة إلى مصفوفة من السلاسل بناءً على الفاصل المحدد.
27	`split(String regex, int limit)`	تقسيم السلسلة إلى مصفوفة من السلاسل بناءً على الفاصل المحدد، مع تحديد الحد الأقصى لعدد العناصر.

String methods

1. charAt(int index)

يعيد الحرف الموجود في الموقع المحدد من السلسلة.

```
String str = "Hello";

char ch = str.charAt(1);

// 'e'
```

2. compareTo(String anotherString)

يقارن السلسلة مع سلسلة أخرى حسب الترتيب القاموسي

```
String str1 = "apple";

String str2 = "banana";

int result = str1.compareTo(str2);

      // ينتج قيمة سالبة لأن "apple" أقل من "banana"
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```
String str1 = "Apple";

String str2 = "apple";

int result = str1.compareToIgnoreCase(str2);

// 0 لأنهما متساويتان مع تجاهل الأحرف الكبيرة والصغيرة
```

3.compareToIgnoreCase(String str)

يقارن سلسلتين مع تجاهل الفرق بين الأحرف الكبيرة والصغيرة

```
String str1 = "Hello";

String str2 = str1.concat(" World");

// "Hello World"
```

4. concat(String str)

يلحق السلسلة المحددة إلى نهاية السلسلة الحالية

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```java
String str = "Hello World";

boolean result = str.contains("World");

// true
```

5. contains(CharSequence s)

يتحقق مما إذا كانت السلسلة تحتوي على التسلسل المحدد من الأحرف

```java
String str1 = "Hello";

String str2 = "Hello";

boolean result = str1.equals(str2);

// true
```

6. equals(Object anObject)

يتحقق مما إذا كانت السلسلة تساوي السلسلة الأخرى

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```
String str1 = "HELLO";

String str2 = "hello";

boolean result = str1.equalsIgnoreCase(str2);

// true
```

7. equalsIgnoreCase(String anotherString)

يتحقق مما إذا كانت السلسلتان متساويتين مع تجاهل الفرق بين الأحرف الكبيرة والصغيرة

```
String str = "Hello";

int index = str.indexOf('e');

// 1
```

8. indexOf(int ch)

يعيد موقع أول ظهور للحرف المحدد

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```java
String str = "Hello World";

int index = str.indexOf("World");

// 6
```

9. indexOf(String str)

يعيد موقع أول ظهور للسلسلة المحددة

```java
String str = "";

boolean result = str.isEmpty();

// true
```

10. isEmpty()

يتحقق مما إذا كانت السلسلة فارغة

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```java
String str = "Hello";

int index = str.lastIndexOf('l');

// 3
```

11. lastIndexOf(int ch)

يعيد موقع آخر ظهور للحرف المحدد.

```java
String str = "Hello";

int len = str.length();

// 5
```

12. length()

يعيد طول السلسلة

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```
String str = "Hello World";

boolean result = str.startsWith("Hello");

// true
```

13. startsWith(String prefix)

يتحقق مما إذا كانت السلسلة تبدأ بالمقطع المحدد.

```
String str = "Hello World";

boolean result = str.endsWith("World");

// true
```

14. endsWith(String prefix)

يتحقق مما إذا كانت السلسلة تنتهي بالمقطع المحدد

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```java
String str = "Hello World";

String sub = str.substring(6);

// "World"
```

15. substring(int beginIndex)

يعيد مقطعًا من السلسلة يبدأ من الموقع المحدد إلى النهاية

```java
String str = "Hello World";

String sub = str.substring(0, 5);

// "Hello"
```

16. substring(int beginIndex, int endIndex)

يعيد مقطعًا من السلسلة من الموقع المحدد إلى الموقع الآخر (غير شامل)

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```
String str = "  Hello World  ";

String trimmed = str.trim();

// "Hello World"
```

17. trim()

يزيل المسافات الفارغة من بداية ونهاية السلسلة

```
String str = "HELLO";

String lower = str.toLowerCase();

// "hello"
```

18. toLowerCase()

يحول جميع الأحرف في السلسلة إلى أحرف صغيرة

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```java
String str = "hello";

String upper = str.toUpperCase();

// "HELLO"
```

19. toUpperCase()

يحول جميع الأحرف في السلسلة إلى أحرف كبيرة

```java
String str = "Hello";

char[] chars = str.toCharArray();

// ['H', 'e', 'l', 'l', 'o']
```

20. toCharArray()

يحول السلسلة إلى مصفوفة من الأحرف

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```
String str = "Hello";

int index = str.lastIndexOf('l');

// 3
```

21. valueOf(int i)

يحول القيمة العددية إلى سلسلة

```
String str = "Hello";

String newStr = str.replace('l', 'p');

// "Heppo"
```

22. replace(char oldChar, char newChar)

يستبدل كل ظهور للحرف القديم بالحرف الجديد

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```
String str = "ahmad mohammed hamada";

String newStr = str.replace("h", "7");

// "a7mad mo7ammed 7amada"
```

23. replace(CharSequence target, CharSequence replacement)

يستبدل جميع ظهورات التسلسل المستهدف بالتسلسل البديل

```
String str = "ahmad mohammed hamada";

String newStr = str.replaceAll("h", "7");

// "a7mad mo7ammed 7amada"
```

24. replaceAll(String regex, String replacement)

يستبدل كل أجزاء السلسلة التي تطابق التعبير النمطي (regex) بالسلسلة البديلة

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```
String str = "ahmad mohammed hamada";

String newStr = str.replaceFirst("h", "7");

// a7mad mohammed hamada
```

25. replaceFirst(String regex, String replacement)

يستبدل أول ظهور للجزء المطابق للتعبير النمطي (regex)
بالسلسلة البديلة

```
String str = "one,two,three";

String[] parts = str.split(",");

// ["one", "two", "three"]
```

26. split(String regex)

يقسم السلسلة بناءً على التعبير النمطي (regex)

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String methods

```
String str = "one,two,three,four";

String[] parts = str.split(",", 3);

// ["one", "two", "three,four"]
```

27. split(String regex, int limit)

يقسم السلسلة بناءً على التعبير النمطي مع تحديد الحد الأقصى لعدد الأجزاء

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String Class

```
String s = "AEprogram" ;
```

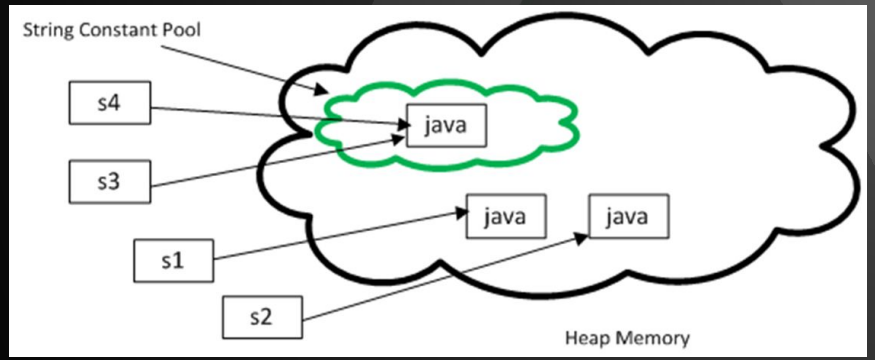
0								8
A	E	p	r	o	g	r	a	m

`s.length()`	9
`s.charAt(4)`	o
`s.indexOf("E")`	1
`s.toUpperCase()`	AEPROGRAM
`s.lastIndexOf("r")`	6

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String Class

```
String s1 = "aseel" ;
String s2 = new String("aseel") ;
String s3 = "eman" ;
String s4 = "eman" ;
```



إدارة تخزين (String) تعتمد على مفهوم String Pool أو . إلى جانب الذاكرة الديناميكية التي تُستخدم لتخزين الكائنات التي يتم إنشاؤها باستخدام البنية new,

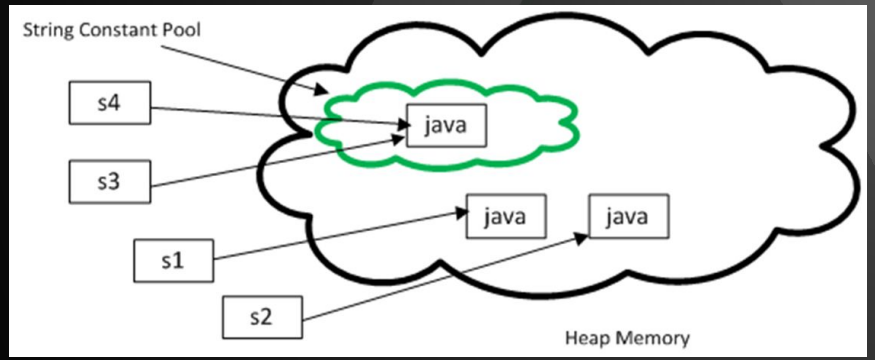
1. **مفهوم** String Pool: هو منطقة خاصة في الذاكرة تُستخدم لتخزين السلاسل النصية الثابتة (literal strings) التي يتم استخدامها بشكل متكرر في البرنامج. يتم استخدام هذا الميكانيزم لتحسين الأداء وكفاءة استخدام الذاكرة.

تخزين النصوص الثابتة: عندما تقوم بإنشاء سلسلة نصية باستخدام النص الثابت مباشرة، مثل "example"، فإن جافا **تتحقق** أولاً مما إذا كانت هذه السلسلة موجودة بالفعل في الـ String Pool.

إعادة استخدام النصوص: إذا كانت السلسلة النصية موجودة في الـ Pool، فإن المتغير الجديد **سيشير** إلى نفس النسخة في الـ Pool بدلاً من إنشاء نسخة جديدة. وهذا يوفر الذاكرة ويعزز الأداء.

String Class

```
String s1 = "aseel" ;

String s2 = new String("aseel") ;

String s3 = "eman" ;

String s4 = "eman" ;
```



آلية التخزين :

1- جافا تتحقق من وجود "aseel" في الـ String Pool. إذا كانت "aseel" غير موجودة، يتم إنشاؤها وتخزينها في الـ Pool. s1 يشيرإلى "aseel" في الـ Pool.

2- "aseel" يتم العثورعليه في الـ String Pool، لكن new String("aseel") ينشئ نسخة جديدة من "aseel" في Heap. Heap: هو جزء من الذاكرة المخصصة لتخزين الكائنات التي يتم إنشاؤها ديناميكيًا. s2 يشيرإلى النسخة الجديدة التي تم إنشاؤها في الـ Heap، بينما s1 يشيرإلى النسخة الأصلية في الـ String Pool.

3- "eman" يتم تخزينه في الـ String Pool إذا لم يكن موجودًا بالفعل. كل من s3 و s4 سيشيران إلى نفس النسخة من "eman" في الـ Pool، لأن جافا تعيد استخدام النسخ الموجودة في الـ Pool بدلاً من إنشاء نسخ جديدة.

String Class

```
String s1 = "aseel" ;
String s2 = new String("aseel") ;
String s3 = "eman" ;
String s4 = "eman" ;
String s5 = "ĖMAN" ;
```

آلية التخزين :

s1 == s2	false
s3 == s4	true
s3 == s5	false
s1.equal(s2)	true

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String Class

```java
int iNum = Integer.parseInt("1111");
```

- في جافا، تحويل النصوص (String) إلى أعداد صحيحة (int) يتم عادة باستخدام الميثود parseInt من كلاس Integer. هذا النوع من التحويل مفيد عندما تحتاج إلى التعامل مع البيانات العددية التي تأتي على شكل نصوص، مثل مدخلات المستخدم أو قراءات من ملفات.

```java
double dNum = Double.parseDouble("7.10");
```

- لتحويل النصوص إلى أعداد من نوع double. يمكنك استخدام الميثود parseDouble من كلاس Double. هذه الميثود تعمل بشكل مشابه لـ parseInt ولكنها تتعامل مع الأعداد العشرية.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

String Class

```
String num1 = "800" ;

String num2 = "311" ;

int sinwar = Integer.parseInt(num1)
 + Integer.parseInt(num2);
```

الكود يقوم بتحويل النصوص "800" و "311" إلى أعداد صحيحة باستخدام Integer.parseInt. ثم يجمع هذين العددين (800 + 311). وأخيرًا، يخزن النتيجة (1111) في المتغير sinwar.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Loops

وَإِذَا سَأَلَكَ عِبَادِي عَنِّي فَإِنِّي قَرِيبٌ أُجِيبُ دَعْوَةَ الدَّاعِ إِذَا دَعَانِ
فَلْيَسْتَجِيبُوا لِي وَلْيُؤْمِنُوا بِي لَعَلَّهُمْ يَرْشُدُونَ

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

```
while (condition) {
    // الكود الذي سيتم تنفيذه
}
```

حلقة while في لغة java هي واحدة من أهم الحلقات التكرارية التي تستخدم لتنفيذ مجموعة من التعليمات بشكل متكرر طالما أن شرطًا معينًا يتحقق. إذا كان الشرط true (صحيح)، يتم تنفيذ الكود داخل الحلقة، وإذا كان الشرط false (خاطئ)، تتوقف الحلقة عن التكرار.

كيفية عمل حلقة while:

1. يتم التحقق من الشرط في بداية الحلقة.
2. إذا كان الشرط true، يتم تنفيذ الكود داخل الحلقة.
3. بعد تنفيذ الكود، يتم الرجوع إلى الشرط مرة أخرى.
4. إذا ظل الشرط true، يتم تكرار الخطوة 2.
5. إذا أصبح الشرط false، تتوقف الحلقة، وينتقل التنفيذ إلى الكود الذي يلي الحلقة.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

سؤال:

اكتب لي كود لإدخال 5 أرقام ثم حساب الوسط الحسابي لها

بإستخدام `while Loop`

كالشكل التالي

```
Enter a number:4
 Enter a number:2
 Enter a number:8
 Enter a number:10
 Enter a number:1
 Sum of the 5 numbers: 25
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

```java
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
       Scanner scan = new Scanner(System.in);
           int counter = 0;
           int sum = 0;
           int num;
           while(counter <5){
               System.out.println("Enter a number: ");
               num = scan.nextInt();
               sum += num;
               counter++;
           }
       System.out.println("Sum of the 5 numbers: "+ sum);
   }
}
```

```java
int counter = 0;
```

في هذا السطر يتم تعريف متغير يسمى `counter` من نوع `int` (عدد صحيح)، ويبدأ بقيمة صفر. هذا المتغير يُستخدم كعداد لعدد المدخلات التي سيتم إدخالها.

```java
int sum = 0;
```

يتم تعريف متغير آخر يسمى `sum` من نوع `int` ويبدأ أيضًا بقيمة صفر. هذا المتغير سيقوم بتخزين مجموع الأرقام المدخلة.

```java
int num;
```

يتم تعريف متغير `num` من نوع `int` والذي سيتم استخدامه لتخزين كل رقم يدخله المستخدم.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

```java
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
       Scanner scan = new Scanner(System.in);
           int counter = 0;
           int sum = 0;
           int num;
           while(counter <5){
               System.out.println("Enter a number: ");
               num = scan.nextInt();
               sum += num;
               counter++;
           }
       System.out.println("Sum of the 5 numbers: "+ sum);
   }
}
```

```java
while(counter <5){
```

هنا نبدأ جملة تكرارية (while loop). هذه الحلقة ستستمر بالتكرار طالما أن قيمة المتغير counter أقل من 5، أي ستتكرر 5 مرات.

```java
sum += num;
```

في هذا السطر، يتم إضافة الرقم الذي أدخله المستخدم (num) إلى مجموع الأرقام المدخلة (sum). هذه العملية تكافئ sum = sum + num.

```java
counter++;
```

هنا يتم زيادة قيمة العداد (counter) بمقدار 1 بعد كل إدخال، حتى يعرف البرنامج كم مرة تمت العملية.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

```
Enter a number:4
 Enter a number:3
 Enter a number:2
 Enter a number:8
 Enter a number:0
 Sum of all numbers entered: 17
```

سؤال: أريد أن تكتب لي برنامج يقوم بجعل المستخدم يدخل أرقام ويطبع مجموعها بالنهاية, ينتهي البرنامج عندما يدخل المستخدم 0

كالشكل التالي :

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

```java
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
       Scanner scan = new Scanner(System.in);
       int sum = 0;
       int num = 1;
       while (num != 0) {
       System.out.println("Enter a number: ");
           num = scan.nextInt();
           sum += num;
       }
       System.out.println("Sum : "+ sum);
   }
}
```

سؤال: أريد أن تكتب لي برنامج يقوم بجعل المستخدم يدخل أرقام ويطبع مجموعها بالنهاية, ينتهي البرنامج عندما يدخل المستخدم 0

قمنا بتعريف متغير sum لحساب المجموع وإعطائه قيمة بداية 0

قمنا بتعريف المتغير num الذي سوف يستقبل بيانات الإدخال, وقمنا بإعطائه قيمة بداية 1 لأنه يجب إعطاء المتغير قيمة بدائية قبل إستعماله بالعمليات الحسابية والمقارنة والخ.. , فـ هدفها الأساسي هو دخول الحلقة

قراءة بيانات من المستخدم

اضافة البيانات المدخلة للمجموع

وهكذا سوف تستمر الحلقة حتى ندخل الرقم 0

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

1- while Loop

سؤال: أريد أن تكتب لي برنامج يقوم بقراءة رقم يدخله المستخدم ثم طباعة كل منزلة منه في سطر منفرد.

كالشكل التالي

```
Enter a number:710
 0
1
7
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

```java
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
       Scanner scan = new Scanner(System.in);
       int num ;
       System.out.println("Enter a number: ");
       num = scan.nextInt();
       while (num > 0) {
           System.out.println(num%10);
           num /= 10;
       }
   }
}
```

سؤال: أريد أن تكتب لي برنامج يقوم بقراءة رقم يدخله المستخدم ثم طباعة كل منزلة منه في سطر منفرد.

وضعنا الشرط بأن يكون الرقم أكبر من 0 , فما الذي حدث بالضبط؟

أولا نقوم بإستخراج المنزلة الأصغر عن طريق إيجاد باقي قسمة الرقم على 10, فنستخرج المنزلة الأصغر ونقوم بطباعتها,

ثم نقوم بقسمة الرقم على 10 لكي يتم محي هذه المنزلة,

وتتكرر الخطوات حتى يصبح الرقم مساوي لصفر ولا يوجد منازل لاستخراجها

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

ملاحظات مهمة !

1- عدم تحديث الشرط يؤدي إلى حدوث حلقة لا نهائية

2- وضع رقم بداخل الشرط يؤدي إلى حدوث حلقة لا نهائية

3- استخدام عبارة صحيحة أساسا يؤدي إلى حدوث حلقة لا نهائية

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

```java
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
       Scanner scan = new Scanner(System.in);
       int num ;
       System.out.println("Enter a number: ");
       num = scan.nextInt();
       while (num > 0) {
           System.out.println(num%10);
             // num /= 10;
       }
   }
}
```

ملاحظات مهمة !

1- عدم تحديث الشرط يؤدي إلى حدوث حلقة لا نهائية

مثلا عندما قمنا بإزالة تحديث الشرط في هذه الحلقة دخلنا في حلقة لا نهائية

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1- while Loop

```c
#include <stdio.h>
int main() {
    while (1) {
        // inf Loop
    }
    while (2==2) {
        // inf Loop
    }
    return 0;
}
```

ملاحظات مهمة !

2- وضع رقم بداخل الشرط يؤدي إلى حدوث حلقة لا نهائية

3- استخدام عبارة صحيحة أساسا يؤدي إلى حدوث حلقة لا نهائية

لكن ماذا لو حدث ووقعنا في حلقة لا نهائية , ما الطريقة لكي نخرج منها ؟

break & continue

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

break & continue

```java
class Main {
   public static void main(String[] args) {
       int i = 0;
       while (i < 10) {
           if (i == 5) {
               break;
            // يخرج من الحلقة عندما تكون i تساوي 5
           }
           System.out.println("Iteration "+ i);
           i++;
       }
   }
}
```

```
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
```

المخرجات

استخدام `break` و `continue` مع `while`:

1. `break`: يُستخدم للخروج من الحلقة فورًا حتى لو كان الشرط لا يزال يتحقق.
   - على سبيل المثال، إذا كنت ترغب في إيقاف الحلقة بناءً على شرط معين داخل الحلقة.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

break & continue

```java
class Main {
   public static void main(String[] args) {
       int i = 0;
       while (i < 10) {
           if (i == 5) {
               continue;
            // يخرج من التكرار الحالي عندما تكون i تساوي 5
           }
           System.out.println("Iteration "+ i);
           i++;
       }
   }
}
```

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
```

المخرجات

استخدام `break` و `continue` مع `while`:

1. `continue` : يُستخدم لتخطي التكرار الحالي للحلقة والانتقال إلى التكرار التالي.
   - يتم تخطي التعليمات المتبقية في التكرار الحالي والانتقال إلى الشرط مرة أخرى.

كما تلاحظون , قام بتخطي التكرار رقم 5

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2- do-while Loop

```
do {
    // الكود الذي سيتم تنفيذه
}while(condition);
```

حلقة `do-while` في لغة java تشبه إلى حد كبير حلقة `while`، ولكن الفرق الرئيسي هو أن حلقة `do-while` تضمن تنفيذ الكود داخل الحلقة على الأقل مرة واحدة، بغض النظر عن الشرط. السبب في ذلك هو أن الشرط يتم التحقق منه بعد تنفيذ الكود، وليس قبله كما هو الحال في حلقة `while`.

كيفية عمل حلقة `do-while`:

1. يتم تنفيذ الكود داخل كتلة `do` مرة واحدة على الأقل.
2. بعد تنفيذ الكود، يتم التحقق من الشرط الموجود بعد `while`.
3. إذا كان الشرط `true`، يتم تكرار تنفيذ الكود.
4. إذا كان الشرط `false`، تتوقف الحلقة عن التكرار وينتقل التنفيذ إلى الكود الذي يليها.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2- do-while Loop

سؤال: أريد أن تكتب لي برنامج يقوم بجعل المستخدم يدخل أرقام
ويطبع مجموعها بالنهاية, ينتهي البرنامج عندما يدخل المستخدم 0

باستخدام do-while Loop

كالشكل التالي

```
Enter a number:4
 Enter a number:2
 Enter a number:8
 Enter a number:10
 Enter a number:1
 Sum of the 5 numbers: 25
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2- do-while Loop

```java
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
       Scanner scan = new Scanner(System.in);
       int sum = 0;
       int num ;
       do {
           System.out.println("Enter a number: ");
           num = scan.nextInt() ;
           sum += num;
       }while (num != 0);
       System.out.println("Sum : "+ sum);
   }
}
```

```java
int num ;
```

هنا يتم تعريف متغير آخر يسمى num من نوع int، والذي سيتم استخدامه لتخزين الرقم الذي يدخله المستخدم في كل مرة, ولم نقم هذه المرة لأن المتغير يحصل على قيمة من المستخدم قبل أن يدخل بعملية مقارنة أو عملية رياضية

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

3- for loop

1. حلقة `for` في لغة java تُستخدم لتكرار تنفيذ مجموعة من التعليمات عددًا محددًا من المرات. تُعتبر حلقة `for` مثالية عندما تعرف مسبقًا عدد التكرارات التي تريد تنفيذها. تتيح لك حلقة `for` تحديد عدد التكرارات وتحديث قيمة المتغيرات المرتبطة بالتكرار في تعليمة واحدة.

كيفية عمل حلقة `for`:

1. تهيئة المتغير: يتم تنفيذ التعليمة `initialization` مرة واحدة في بداية الحلقة.
2. التحقق من الشرط: يتم التحقق من `condition`. إذا كان `true`، يتم تنفيذ الكود داخل الحلقة. إذا كان `false`، تتوقف الحلقة.
3. تنفيذ الكود: يتم تنفيذ الكود داخل الحلقة.
4. تحديث المتغير: يتم تنفيذ التعليمة `update`.
5. العودة إلى الشرط: يتم التحقق من الشرط مرة أخرى، وتستمر العملية حتى يصبح الشرط `false`.

```
for(initialization; condition; update) {
    // الكود الذي سيتم تنفيذه
}
```

- `initialization`: يُستخدم لتهيئة المتغير الذي يتحكم في الحلقة. يتم تنفيذ هذه التعليمة مرة واحدة فقط في بداية الحلقة.
- `condition`: هو الشرط الذي يتم التحقق منه قبل كل تكرار. إذا كان الشرط **true** (صحيحًا)، يتم تنفيذ الكود داخل الحلقة. إذا كان **false** (خاطئًا)، تتوقف الحلقة.
- `update`: يُستخدم لتحديث قيمة المتغير الذي يتحكم في الحلقة بعد تنفيذ الكود داخل الحلقة. يتم تنفيذ هذه التعليمة في نهاية كل تكرار.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

3- for loop

سؤال:

اكتب لي كود لإدخال 5 أرقام ثم حساب الوسط الحسابي لها

بإستخدام `for Loop`

كالشكل التالي

```
Enter a number:4
 Enter a number:2
 Enter a number:8
 Enter a number:10
 Enter a number:1
 Sum of the 5 numbers: 25
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

3- for loop

```java
import java.util.Scanner;
class Main {
   public static void main(String[] args) {
       Scanner scan = new Scanner(System.in);
       int sum = 0;
       int num;
       for (int i = 0; i < 5 ; i++){
           System.out.println("Enter a number: ");
           num = scan.nextInt();
           sum += num;
       }
       System.out.println("Sum of the 5 numbers: "+sum);
   }
}
```

```java
for (int i = 0; i < 5 ; i++){
```

هنا تبدأ حلقة `for` هذه الحلقة تتكرر 5 مرات بفضل تعيين `i` إلى 0 وشرط التكرار `i < 5`.
بعد كل تكرار، يتم زيادة قيمة `i` بمقدار 1 `(++i)`.

- **الجزء الأول** `(int i = 0;)`:

يتم تعيين المتغير i إلى 0 في بداية الحلقة.

- **الجزء الثاني** `(i < 5)`:

يتم التحقق من أن قيمة i أقل من 5. إذا كان الشرط صحيحًا، يتم تنفيذ الكود داخل الحلقة.

- **الجزء الثالث** `(i++)`:

يتم زيادة قيمة `i` بمقدار 1 بعد كل تكرار للحلقة.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Nested Loops

```
for (initialization1; condition1; update1) {
// الكود الذي يتم تنفيذه في كل تكرار من الحلقة الخارجية

for (initialization2; condition2; update2) {
// الكود الذي يتم تنفيذه في كل تكرار من الحلقة الداخلية
}
}
```

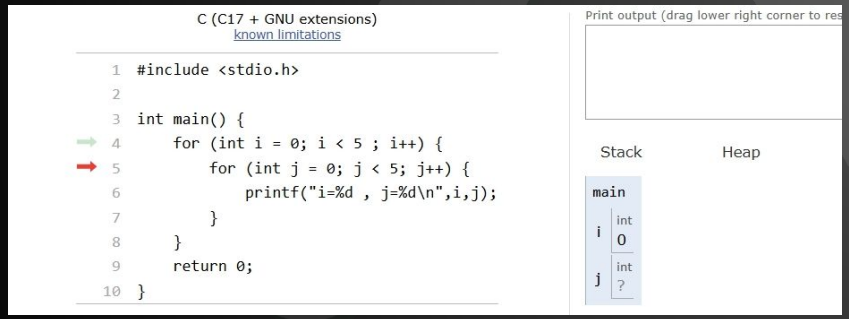
ما هي الحلقات المتداخلة؟

عند استخدام حلقة `for` بداخل حلقة `for` أخرى، فإن الحلقة الداخلية تتكرر بالكامل لكل تكرار للحلقة الخارجية. يمكن أن تحتوي الحلقة الداخلية على أي نوع من الحلقات (مثل `for`، `while`، أو `do-while`)، ولكن في هذا السياق، سنركز على حلقات `for`.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Nested Loops

```java
class Main {
   public static void main(String[] args) {
       for (int i = 0; i < 5 ; i++) {
           for (int j = 0; j < 5; j++) {
               System.out.println("i="+i+" , j="+j);
           }
       }
   }
}
```



مثال توضيحي:

نبدأ البرنامج وندخل الحلقة الخارجية بقيمة i=0, وثم ندخل بداخل الحلقة الخارجية فنجد حلقة ثانية بداخلها, فيقوم البرنامج ببدء وإنهاء الحلقة الداخلية ثم يخرج منها وينتهي عملها بالنسبة للدورة الخاصة بالحلقة الخارجية, وبما أننا أنهينا الدورة الأولى بالحلقة الخارجية فستزداد i بمقدار 1 , وستصبح i=1, وثم نبدأ بالدورة الثانية للحلقة الخارجية ونجد أيضا الحلقة الداخلية بداخلها , فنقوم ببدئها وإنهائها كما فعلنا من قبل, وهذه صورة توضيحية للدورة الأولى من الحلقة:

كما قلنا: بدأنا الحلقة الخارجية بقيمة i=0, والان سنمشي بالكود حتى نصل الحلقة الداخلية

السهم الأخضر يمثل الخطوة الحالية والأحمر الخطوة القادمة

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

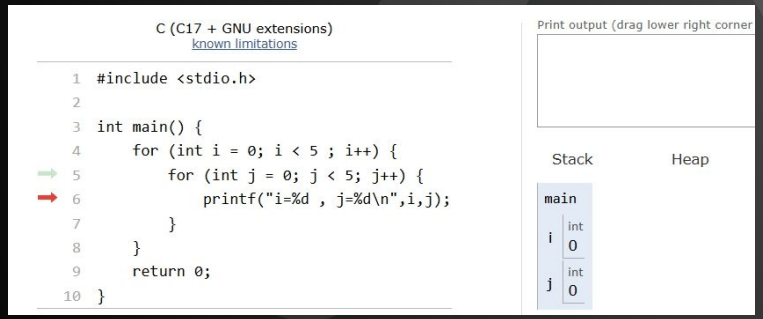
Nested Loops

```java
class Main {
   public static void main(String[] args) {
       for (int i = 0; i < 5 ; i++) {
           for (int j = 0; j < 5; j++) {
               System.out.println("i="+i+" , j="+j);
           }
       }
   }
}
```

مثال توضيحي:

الان دخلنا الحلقة الداخلية بقيمة 0=j ولا ننسا أننا لا زلنا بالدورة الأولى للحلقة الخارجية بقيمة 0=i , فالخطوة التالية هي طباعة النتيجة, فتقوم بطباعة التالي

```
i=0 , j=0
```



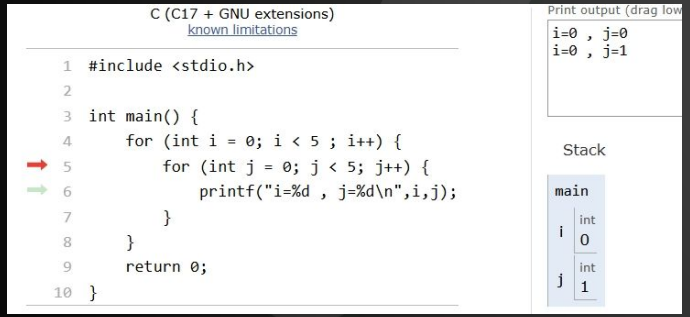
اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Nested Loops

```java
class Main {
   public static void main(String[] args) {
       for (int i = 0; i < 5 ; i++) {
           for (int j = 0; j < 5; j++) {
               System.out.println("i="+i+" , j="+j);
           }
       }
   }
}
```

مثال توضيحي:

تستمر الحلقة الداخلية حتى تنتهي



اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

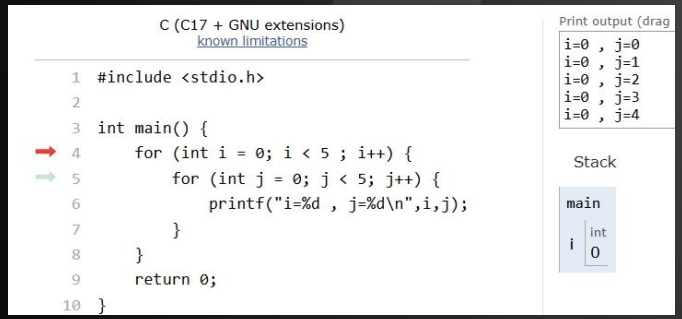
Nested Loops

```java
class Main {
   public static void main(String[] args) {
       for (int i = 0; i < 5 ; i++) {
           for (int j = 0; j < 5; j++) {
               System.out.println("i="+i+" , j="+j);
           }
       }
   }
}
```

مثال توضيحي:

والان انتهت الحلقة الداخلية, فسيتم اكمال الكود الذي بداخل الحلقة الخارجية بشكل طبيعي, ونظرا أنه لا يوجد شيء بداخل الحلقة الخارجية سوى الحلقة الداخلية, فستنتهي الدورة الاولى من الحلقة الخارجية وسنبدأ بالدورة الثانية

كما تلاحظون عندما تم الانتهاء من الحلقة الداخلية اختفى المتغير j , وهذا يعني ان الحلقة انتهت



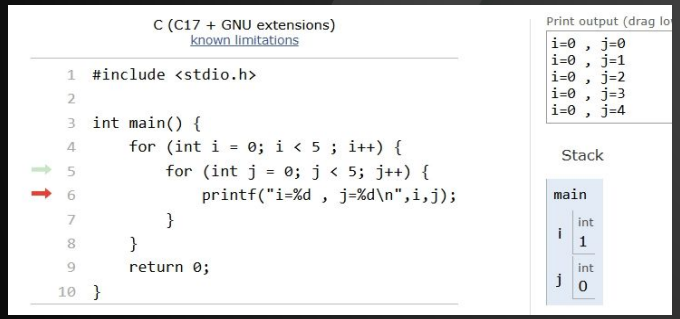
اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Nested Loops

```java
class Main {
   public static void main(String[] args) {
       for (int i = 0; i < 5 ; i++) {
           for (int j = 0; j < 5; j++) {
               System.out.println("i="+i+" , j="+j);
           }
       }
   }
}
```

مثال توضيحي:

والان بدأت الدورة الثانية من الحلقة ودخلنا بالحلقة الداخلية دخول جديد بتعريف جديد كما أننا لم نستخدمها بالسابق, فقيمة i=1 و قيمة j=0, وهذا يدل على أن الحلقة الداخلية بدأت من جديد, فكما تم تنفيذ الدورة الأولى سابقا سيتم تنفيذ الدورة الثانية الان



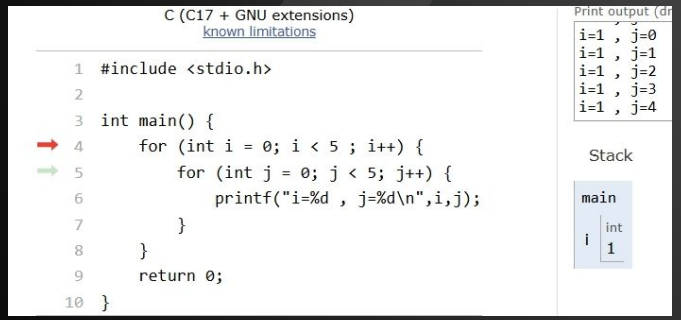
اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Nested Loops

```java
class Main {
   public static void main(String[] args) {
       for (int i = 0; i < 5 ; i++) {
           for (int j = 0; j < 5; j++) {
               System.out.println("i="+i+" , j="+j);
           }
       }
   }
}
```

مثال توضيحي:

انتهت الدورة الثانية من الحلقة الخارجية وهذه هي المخرجات, وتستمر الحلقة الخارجية بدوراتها حتى تنتهي



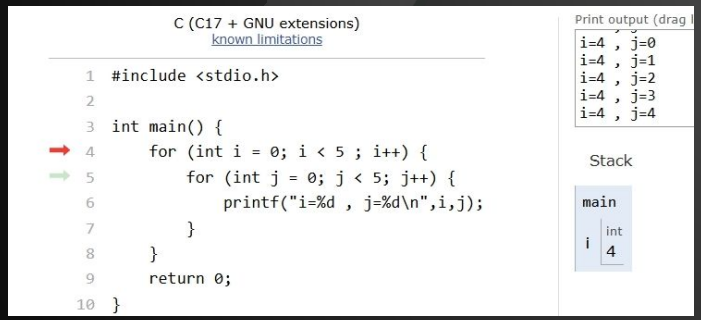
اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Nested Loops

```java
class Main {
   public static void main(String[] args) {
       for (int i = 0; i < 5 ; i++) {
           for (int j = 0; j < 5; j++) {
               System.out.println("i="+i+" , j="+j);
           }
       }
   }
}
```

مثال توضيحي:

تم الانتهاء من الدورة الأخيرة من الحلقة الخارجية, والخطوة التالية
ستكون return 0 , أي نهاية البرنامج



اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Nested Loops

```java
class Main {
   public static void main(String[] args) {
       for (int i = 0; i < 5 ; i++) {
           for (int j = 0; j < 5; j++) {
               System.out.println("i="+i+" , j="+j);
           }
       }
   }
}
```

```
i=0 , j=0
i=0 , j=1
i=0 , j=2
i=0 , j=3
i=0 , j=4
i=1 , j=0
i=1 , j=1
i=1 , j=2
i=1 , j=3
i=1 , j=4
i=2 , j=0
i=2 , j=1
i=2 , j=2
i=2 , j=3
i=2 , j=4
i=3 , j=0
i=3 , j=1
i=3 , j=2
i=3 , j=3
i=3 , j=4
i=4 , j=0
i=4 , j=1
i=4 , j=2
i=4 , j=3
i=4 , j=4

Process finished with exit code 0
```

مثال توضيحي:

المخرجات النهائية للبرنامج :

هذا البرنامج يعرض قيمة j,i في كل خطوة بالبرنامج

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Nested Loops

```java
class Main {
   public static void main(String[] args) {
       int sum_minutes = 0;

    for (int hours = 0; hours < 24; hours++) {

       for (int minutes = 0; minutes < 60; minutes++) {
              sum_minutes ++ ;
         }
    }

  System.out.print("minutes in a day: "+ sum_minutes);
   }
}
```

سؤال: اكتب لي برنامج لحساب وطباعة عدد الدقائق في يوم كامل,
مستخدما حلقتين: الأولى للساعات والثانية للدقائق

البرنامج يستخدم حلقتين for للتكرار: الأولى لتكرار الساعات

(من 0 إلى 23) والثانية لتكرار الدقائق (من 0 إلى 59). في كل تكرار للحلقة الداخلية، يتم زيادة عدد الدقائق الكلي بمقدار 1. في النهاية، يتم طباعة الإجمالي. في كل دورة لحلقة الساعات يزداد المجموع بمقدار 60 (عدد الدقائق في الساعة), وتتكرر حلقة الساعات 24 مرة, مما يجعل المجموع النهائي

```
Total minutes in a day: 1440
```

1440=60*24

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

قال تعالى: " وَقُلِ ٱعۡمَلُواْ فَسَيَرَى ٱللَّهُ عَمَلَكُمۡ وَرَسُولُهُۥ وَٱلۡمُؤۡمِنُونَۖ وَسَتُرَدُّونَ إِلَىٰ عَٰلِمِ ٱلۡغَيۡبِ وَٱلشَّهَٰدَةِ فَيُنَبِّئُكُم بِمَا كُنتُمۡ تَعۡمَلُونَ "

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

في لغة JAVA، الدوال (`methods`) هي وحدات برمجية تساعد في تنظيم الكود وجعله أكثر قابلية للفهم وإعادة الاستخدام. يمكن أن تحتوي الدالة على مجموعة من التعليمات التي تنفذ مهمة محددة، ويمكن استدعاؤها من أماكن مختلفة في البرنامج. يساعد استخدام الدوال على تقسيم البرنامج إلى أجزاء أصغر وأكثر سهولة في الفهم والإدارة.

لماذا نستخدم الدوال؟

1. إعادة الاستخدام: يمكنك استدعاء الدالة عدة مرات بدلاً من كتابة نفس الكود مرارًا وتكرارًا.
2. تنظيم الكود: تقسيم المهام الكبيرة إلى دوال أصغر يسهل من عملية الفهم والإدارة.
3. التسهيل في الصيانة:

بد

لاً من تعديل الكود في أماكن متعددة، يمكنك تعديل دالة واحدة وستؤثر التغييرات على كل الأماكن التي تستخدم فيها الدالة.

4. تقليل الأخطاء: يسهل عليك اختبار الدوال بشكل منفصل للتحقق من عملها، مما يقلل من احتمالية الأخطاء في البرنامج.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

مكونات الـ method في لغة JAVA:

1. نوع الدالة (`Return Type`): يحدد نوع القيمة التي تقوم الدالة بإرجاعها بعد التنفيذ. إذا لم تُرجع الدالة أي قيمة، يُستخدم النوع `void`.
2. اسم الدالة (`Function Name`): هو اسم الدالة الذي يُستخدم لاستدعائها.
3. المعطيات (`Parameters`): هي القيم التي يمكن تمريرها إلى الدالة لتعمل عليها. يمكن أن تكون دالة بدون معطيات.
4. الجسم (`Body`): يحتوي على مجموعة التعليمات التي تنفذها الدالة.
5. الإرجاع (`Return`): الكلمة المفتاحية `return` تُستخدم لإرجاع قيمة من الدالة, يجب أن يكون نوع البيانات المرجعة مثل المعلن عنها في بداية الدالة.
6. المحددات (modifiers) : يعبربشكل أساسي عن مدى رؤية الـ method في الكود.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

استدعاء الـ method في لغة JAVA:

في لغة جافا الـ method call عبارة عن كتابة اسم الـ mehod في الـ main وتمرير قيم اذا كان يأخذ قيم

```
public class Main {
    public static void main(String[] args) {
       modifier return_data_type method_name(arguments)
    }
    modifier return_data_type  method_name(parameters) {
        //code
    }
}
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

```
Modifier return_type method_name(parameters) {
    // method body
    return value;
}
```

هناك 4 أنوع للدوال في لغة الـ JAVA.

1- دوال تستقبل مدخلات وتُرجع قيمة

2- دوال لا تستقبل مدخلات وتُرجع قيمة

3- دوال تستقبل مدخلات ولا وتُرجع قيمة

4- دوال لا تستقبل مدخلات ولا وتُرجع قيمة

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

```java
public class Main {
    public static void main(String[] args)
{
        int a =  add(2,3);
    }

    public static int add(int a, int b){
        return  a+b;
    }
}
```

1- دوال تستقبل مدخلات وترجع قيمة:

مثال توضيحي:

قمنا هنا بإنشاء دالة تدخل بها متغيرين من نوع int وترجع لي مجموعهما.

```java
public static int add(int a, int b)
```

نوع البيانات الذي سيتم إرجاعه من الدالة : `int`

اسم الدالة : `add`

إعلان المتغيرات التي سيتم استدعاؤها بالدالة: `int a, int b)`

إعلان الناتج النهائي الذي سوف ترجعه الدالة, والذي بمجرد الوصول إليه سينتهي عمل الدالة وسنخرج منها مرجعين النتيجة النهائية, نلاحظ أن نوع بيانات الإرجاع مطابق لما تم الإعلان عنها في اول كلمة بالدالة . `return  a+b;`

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

```java
public class Main {
    public static void main(String[] args)
{
        System.out.println("PI = "+ PI());
    }

    public static double PI(){
        return  3.14;
    }
}
```

2- دوال لا تستقبل مدخلات وتُرجع قيمة:

يوجد دوال لا تستقبل معطيات, ولكن سواء وجدت المعطيات أم لا, يمكننا إرجاع نتيجة

مثال توضيحي:

كما ترون هنا قمنا بتعريف دالة PI تقوم بإرجاع قيمة Pi لي وهي 3.14 , ولا نحتاج لمعطيات لحساب هذه النتيجة

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

```java
public class Main {
   public static void main(String[] args) {
      max(3,6);
   }
   public static void max(int x, int y){
       if (x > y)
           System.out.println(x+ " larger than " + y);
       else if (x == y)
           System.out.println(x + " equals " + y);
       else
           System.out.println(x+ " less than " + y);
   }
}
```

3- دوال تستقبل مدخلات ولا وتُرجع قيمة

يوجد دوال تستقبل قيمة وتقوم بإجراء العمليات عليها مثل طباعة وهكذا بدون الحاجة إلى إرجاع قيمة فعلية

مثال توضيحي:

كما ترون هنا قمنا بإنشاء دالة max والتي تستقبل معطيين ثم تقوم بتحليل من منهما (أكبر, أصغر أو يساوي), ثم طباعة الناتج كالشكال التالي:

طبعا نظرا أننا لن نقوم بإرجاع شيء , قمنا بوضع نوع الدالة `void`,

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

```java
public class Main {
    public static void main(String[] args) {
        hello();
    }
    public static void hello() {
        System.out.println("hello");
    }
}
```

4- دوال لا تستقبل مدخلات ولا وترجع قيمة

يوجد دوال لا تستقبل قيمة وتقوم بإجراء العمليات عليها مثل طباعة وهكذا بدون الحاجة إلى إرجاع قيمة فعلية

مثال توضيحي:

كما ترون هنا قمنا بإنشاء دالة `hello` والتي لا تستقبل أي معطيات ولا تقوم بإرجاع أي قيم, لكن تقوم بطباعة كلمة hello.

عادة نستخدم هذا النوع من الدوال لحفظ جملة طباعة نستخدمها في أكثر من موقع بالبرنامج, فنقوم بكتابتها مرة واحدة فقط ثم نستدعيها في اكثر من موقع

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

```
public class Main {
    public static void main(String[] args) {
        int x = -5;

        System.out.println("|" + x+  "| = " + abs(x));
    }

 public static int abs(int x) {
        if (x < 0)
            return x * -1;
        else
            return x;

    }
}
```

سؤال: قم بكتابة دالة تدخل بها رقم صحيح وترجع لك القيمة المطلقة له

وضعنا شرط بالدالة أنه اذا كان الرقم اصغر من صفر (سالب) ارجع لي الرقم مضروب بـ -1، واذا كان موجب ارجعه كما هو

المخرجات :

```
/Users/emangoat7/Library/Java/JavaVirtualMachines/open
|-5| = 5

Process finished with exit code 0
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

Overloading:

multiple methods with the same name but different parameters.

Overriding:

multiple methods with the same name but different body.(will be explained later)

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

```
public class Calculator {
    public int calculate(int a, int b) {
        return a + b;
    }

    public double calculate(double a, double
b){
        return a * b;
    }
}
```

Overloading:

multiple methods with the same name but different parameters.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

METHODS

```java
public class Calculator {
    public int calculate(int a, int b) {
        return a + b;
    }

    public double calculate(double a, double
b){
        return a * b;
    }
}
```

Overloading:

Multiple methods with the same name but different parameters.

This may lead to an error, The compiler can't decide which method to execute.

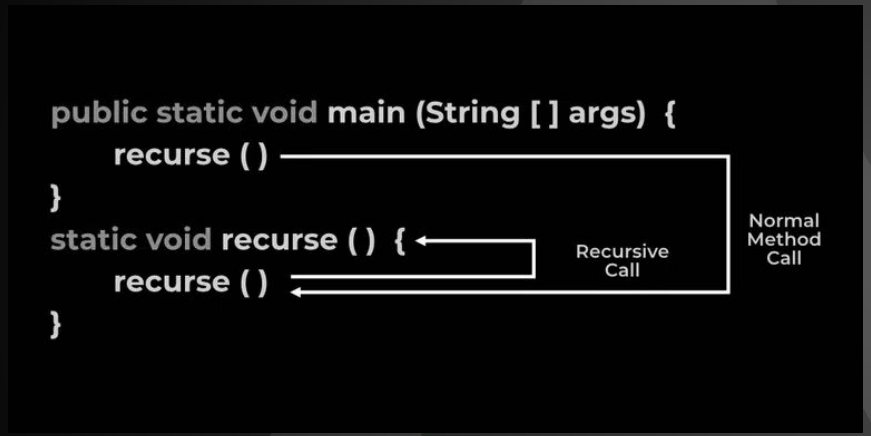
اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

RECURSION

يا عبادي إنَّما هيَ أعمالُكم أحصيها لكم ثمَّ أوفّيكم إيَّاها، فمن وجدَ خيرًا فليحمدِ اللَّهَ، ومن وجدَ غيرَ ذلِكَ فلا يلومنَّ إلَّا نفسَهُ).
[حديث قُدسي - رواه مسلم].

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Recursion



Recursion:

occurs when the definition of a concept or process depends on a simpler or previous version of itself.

يستخدم الـ recursion في الخوارزميات التي تحتاج الى تكرار نفس الخطوات، بشرط وجود شرط يوقف عملية الـ recursion ويتم فيها استدعاء الـ method بداخل نفسه.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Recursion

```
public class Main {
    public static void main(String[] args)
{
        System.out.println(factorial(3));
    }

    public static int factorial(int num){

        if (num==0||num==1)
            return 1 ;
        else return num*factorial(num-1);
    }
}
```

Recursion:

occurs when the definition of a concept or process depends on a simpler or previous version of itself.

Example1:

من أهم الأمثلة وأوضحها على الـ recursion هو الـ factorial

```
factorial(3) =
3*factorial(2) =
3*2*factorial(1) =
3*2*1 = 6
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Recursion

```
public class Main {
    public static void main(String[] args) {
        System.out.println(sum(10));
    }

    public static int sum(int k) {
        if (k > 0) {
            return k + sum(k - 1);
        } else {
            return 0;
        }
    }
}
```

Example(2):

Use recursion to add all of the numbers up to 10.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Recursion

```
public static boolean blindroom(String word){
   if (word.length() <= 1) return true ;
   else if (word.charAt(0) != word.charAt(word.length()
- 1)) {
       return false ;
   } else
       return blindroom(word.substring(1 , word.length()
- 1)) ;
}
```

Example(3):

Write a Java recursive method to check if a given string is a palindrome.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Recursion

```
public static double pow(int x , int y){

   if (y==0) return 1 ;

   else
       return x * pow(x , y - 1) ;

}
```

Example(4):

Write a Java recursive method to calculate the exponentiation of a number (base) raised to a power (exponent).

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Recursion

```
public static int min(int num1 , int num2) {

    if (num2 == 0)
        return num1;

    int remainder = num1 % num2;
    return min(num2, remainder);
}
```

Example(5):

Write a Java recursive method to find the greatest common divisor (GCD) of two numbers.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Recursion

```
public static int sum_of_Odd_number(int num){
   if (num == 0) return 0 ;
   else {
       if(num%2 == 1){
           return num%10 + sum_of_Odd_number(num/10) ;
       }else
           return sum_of_Odd_number(num/10) ;
   }
}
```

Example(6):

Write a Java recursive method to find the sum of all odd numbers in an array.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

ARRAY

A	r	r	a	y

قال تعالى: (كُتِبَ عَلَيْكُمُ الْقِتَالُ وَهُوَ كُرْهٌ لَّكُمْ ۖ وَعَسَىٰ أَن تَكْرَهُوا شَيْئًا وَهُوَ خَيْرٌ لَّكُمْ ۖ وَعَسَىٰ أَن تُحِبُّوا شَيْئًا وَهُوَ شَرٌّ لَّكُمْ ۗ وَاللَّهُ يَعْلَمُ وَأَنتُمْ لَا تَعْلَمُونَ)

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Array:

are used to store multiple values in a single variable, instead of declaring separate variables for each value.

تعتبر الـ array كـ مخزن لتخزين عدّة قيم من نوع معيّن.

لتعريف الـ array :

يجب تحديد نوع الداتا وعددها.

```
DataType[ ] arrayName ;
```
تم إنشاء array فارغة

```
arrayName = new DataType[Array Length] ;
```
تم تحديد عدد عناصرها

```
arrayName = { data  }
```
اعطاء قيم

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Array:

are used to store multiple values in a single variable, instead of declaring separate variables for each value.

تعتبر الـ array كـ مخزن لتخزين عدّة قيم من نوع معيّن،

لتعريف الـ array :

يجب تحديد نوع الداتا وعددها.

```
dataType [] AraayName =  { ..... }
```

شاء اراي واعطائها قيم في خطوة واحدة

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

```
int [] numbers  =  { 1,2,3,4 };

 Numbers length = 5
 Numbers [0] = 1
 Numbers [1] = 2
 Numbers [2] = 3
 Numbers [3] = 4
```

Array:

are used to store multiple values in a single variable, instead of declaring separate variables for each value.

ترقيم العناصر يبدأ دائما من index 0، في حال كان عدد العناصر = n اذن الـ index يبدأ بـ 0 وينتهي بـ n - 1

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

```
int[ ] array = {1 , 2 , 2 , 3 ,0 , 6 , 3 };
```

Code	output
array.length	7
array[1]	2
array[7]	error

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

```
double[ ] array = new double[5];
array[0] = 3 ;
array[2] = 8.2 ;
array[3] = 3.2 ;

for (int i = 0; i < array.length; i++) {
   System.out.println(array[i]);
}
```

عند انشاء الـ array بهذا الشكل، فسيتم اعطاء قيم اولية
من نوع double وستكون قيمتها 0.0

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

```
double[ ] array = new double[5];
array[0] = 3 ;
array[2] = 8.2 ;
array[3] = 3.2 ;

for (int i = 0; i < array.length; i++) {
   System.out.println(array[i]);
}
```

لم يتم اعطاء قيم لـ i [4] array , [1] array لذلك بقيت قيمتها على الـ default value وهي 0.0

output	index
3.0	0
0.0	1
8.2	2
3.2	3
0.0	4

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

array of Object : String.

```
String[] name = new String[4] ;
name[0] = "aصeel" ;
name[1] = "adel" ;
name[3] = "qadah" ;
for (int i = 0; i < name.length; i++) {
    System.out.println(name[i]+" ");
}
```

لم يتم اعطاء قيم لـ i [2] array لذلك بقيت قيمتها على الـ default value وهي null

output	index
aصeel	0
adel	1
null	2
qadah	3

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

copy arrays

```
int[] array1 = {4 , 2 , 11 , 2 , 0} ;
```

عند انشاء array1 يتم حجز مكان لها في الذاكرة تشير إليه

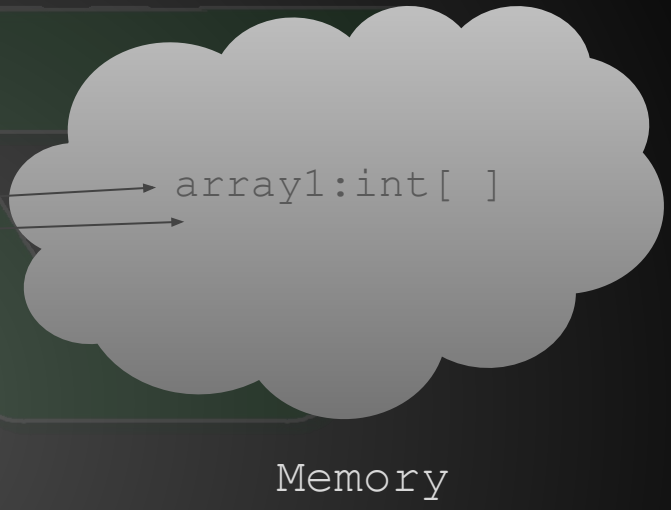
اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

copy arrays

```
int[] array1 = {4 , 2 , 11 , 2 , 0} ;
int[] array2 = array1 ;
```



عند انشاء array1 يتم حجز مكان لها في الذاكرة تشير إليه،
وعند مساواتها بـ array2 تصبح المصفوفة الثانية تشير إلى نفس الموقع.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

For - each loops:

```
int[] array1 = { 4 , 2 , 11 , 2 , 0} ;
int[] array2 = array1 ;
array2[0] = 1111 ;
array1[1] = 710 ;

for (int i : array1) {
   System.out.println(i);
}
```

output
1111
710
11
2
0

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

casting

```
double[] array1 = {4.1, 2.9, 11, 2.5, 0.2}
;
int[] array2 = new int[array1.length] ;

for (int i = 0; i < array1.length; i++) {
   array2[i] = array1[i] ;
}

for (int i = 0; i < array2.length; i++) {
   System.out.println(array2[i]);
}
```

الـ array الاولى معرفة كـ double اما الثانية فقيمها معرفة كـ int

لا يمكن اعطاء قيمة double ل int إذاً يلزمنا أن نقوم بعملية casting للرقم وتحويله إلى int

(int)

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

casting

```
double[] array1 = {4.1, 2.9, 11, 2.5, 0.2}
;
int[] array2 = new int[array1.length] ;

for (int i = 0; i < array1.length; i++) {
   array2[i] = (int) array1[i] ;
}

for (int i = 0; i < array2.length; i++) {
   System.out.println(array2[i]);
}
```

output
4
2
11
2
0

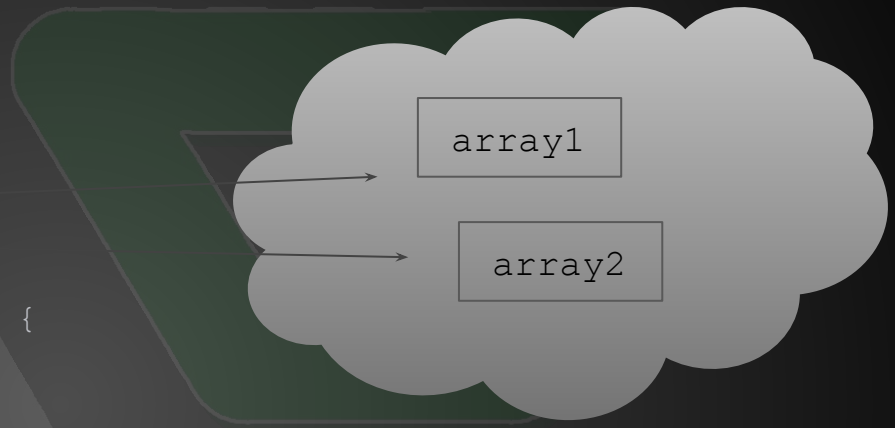
اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

Copy array.

```
int[] array1 = {4 , 2 , 11 , 2 , 0} ;
int[] array2 = new int[5];

for (int i = 0; i < array2.length; i++) {
   array2[i] = array1[i] ;
}

array2[0] = 1111 ;
array1[1] = 710 ;

for (int i : array1){
   System.out.println(i);
}
```



في هذا المثال تم انشاء array جديدة ولكنها تساوي الـ array الاولى في القيم، لذلك فلكل واحدة عنوان مختلف في الذاكرة، وعندما يتم التعديل على الثانية لن تتأثر الأولى.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

1D Array

Example:

Copy array.

```
int[] array1 = {4 , 2 , 11 , 2 , 0} ;
int[] array2 = new int[5];

for (int i = 0; i < array2.length; i++) {
   array2[i] = array1[i] ;
}

array2[0] = 1111 ;
array1[1] = 710 ;

for (int i : array1){
   System.out.println(i);
}
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D-ARRAY

A	r	r	a	y
A	r	r	a	y

قال تعالى: (إِنْ أَحْسَنتُمْ أَحْسَنتُمْ لِأَنفُسِكُمْ ۖ وَإِنْ أَسَأْتُمْ فَلَهَا ۚ فَإِذَا جَاءَ وَعْدُ الْآخِرَةِ لِيَسُوءُوا وُجُوهَكُمْ وَلِيَدْخُلُوا الْمَسْجِدَ كَمَا دَخَلُوهُ أَوَّلَ مَرَّةٍ وَلِيُتَبِّرُوا مَا عَلَوْا تَتْبِيرًا)

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

```
DataType[][] arrayName
```
فارغة array تم إنشاء

```
arrayName = new DataType[number or rows][number or columns] ;
```
تم تحديد عدد عناصرها

```
arrayName =   { data  },
              {data},
              {data};
```
اعطاء قيم

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

```
DataType[][] arrayName
```
فارغة array تم إنشاء

```
arrayName = new DataType[number or rows][number or columns] ;
```
تم تحديد عدد عناصرها

```
arrayName =   { data  },
              {data},
              {data};
```
اعطاء قيم

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

العمود

	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

الصف

```
matrix.length? 5   لإيجاد عدد الصفوف

matrix[0].length? 5 لإيجاد عدد الأعمدة



 int[][] matrix = new int [5][5]
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

```
int [][]matrix = {
        , {1, 2, 3, 4, 5}
        , {2, 3, 4, 5}
        , {3, 4, 5}
        , {4, 5}
        , {5}
};
```

```
matrix.length is 5

matrix[0].length is 5

matrix[1].length is 4

matrix[2].length is 3

matrix[3].length is 2

matrix[4].length is 1

matrix[5].length is ERROR
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

Example(1):

```
public class Main{
   public static void main (String [] args) {
       int[][] array = new int[5][6];
       int[] x = {1, 2};
       array[0] = x;
       System.out.println("array[0][1] is " + array[0][1]);
   }
}
```

Output:

```
array[0][1] is 2
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

Example(2):

```
public class Main{
   public static void main (String [] args) {
       int[][] array = { {1, 2} ,  {3, 4} ,  {5, 6} };
       for (int i = array.length - 1 ;  i >= 0 ;  i--) {
           for (int j = array[ i ].length - 1 ; j  >= 0 ;  j--)
               System.out.print(array[ i ][ j ] + " ");
           System.out.println();
       }
   }
}
```

Output:

```
6 5
4 3
2 1
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

Example(3):

```
public class Main{
   public static void main (String [] args) {
       int[][] array = { {1, 2} ,  {3, 4} ,  {5, 6} };
       int sum = 0;
       for (int i = 0; i < array.length; i++)
           sum += array[i][ 0];
       System.out.println(sum);
   }
}
```

Output:

```
9
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

Example(4):

```
public class Main {
    public static void main(String[] args) {
        int[ ][ ] array = { {1, 2, 3, 4} , {5, 6, 7, 8} };
        System.out.println(m1(array)[ 0 ]);
        System.out.println(m1(array)[ 1 ]);
    }
    public static int[ ] m1(int[ ][ ] m) {
        int[ ] result = new int[ 2 ];
        result[ 0 ] = m.length;
        result[ 1 ] = m[ 0 ].length;
        return result;
    }
}
```

Output:

```
2
4
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

Example(5):

```
public class Main {
    public static void main(String[] args) {
        int [ ][ ] matrix = {
                { 1,2,3},
                { 4,5,6}
        };
        int element = matrix [ 0][1];
        System.out.println(element);
    }
}
```

Output:

2

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

Example(6):

```
public class Main {
    public static void main(String[] args) {
        // إنشاء مصفوفة ثنائية الأبعاد
        int[ ][ ] matrix = {
                {1, 2, 3},
                {4, 5, 6}
        };
        // تغيير قيمة العنصر في الصف الثاني والعمود الثالث
        matrix[1][2] = 99;
        // طباعة المصفوفة بعد التعديل باستخدام حلقتين for
        for (int i = 0; i < matrix.length; i++) { // الحلقة الخارجية تتنقل عبر
الصفوف
            for (int j = 0; j < matrix[ i ].length; j++) { // الحلقة الداخلية
تتنقل عبر الأعمدة في كل صف
                System.out.print(matrix[ i ][ j ] + " ");
            }
            System.out.println(); // الانتقال إلى السطر التالي بعد طباعة كل صف
        }
    }
}
```

Output:

```
1 2 3
4 5 99
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

Example(7):

```
public class Main {
    public static void main(String[] args) {
        // إنشاء مصفوفة ثنائية الأبعاد (2D-array)
        int[ ][ ] originalArray = {
                { 1, 2, 3},
                { 4, 5, 6},
                { 7, 8, 9}
        };
        // إنشاء مصفوفة جديدة تشير إلى نفس المصفوفة الأصلية
        int[ ][ ] referenceArray = originalArray;
        // تغيير قيمة في المصفوفة القديمة
        originalArray[0][0] = 99;
        // طباعة المصفوفة باستخدام الحلقات
        for (int i = 0; i < referenceArray.length; i++) {
            for (int j = 0; j < referenceArray[ i ].length; j++) {
                System.out.print(referenceArray[ i ][ j ] +  " ");
            }
            System.out.println();
        }
    }
}
```

Output:

```
99 2 3
4 5 6
7 8 9
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

Example(8):

```
public class Main {
    public static void main(String[] args) {
// إنشاء مصفوفة ثنائية الأبعاد (2D-array)
        int[ ][ ] array = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
        };
// محاولة الوصول إلى عنصر خارج حدود المصفوفة
        int value = array[3][0]; // هذا السطر سيتسبب في حدوث خطأ
        System.out.println(value);
    }
}
```

Output:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException :
Index 3 out of bounds for length 3
    at Main.main(Main.java:10)
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

2D Array

Example(9):

```
public class Main {
    public static void main(String[] args) {
    String[][] list = {
            {"aصeel" , "eman" ,"hannen"},
            {"ibrahem" , "wassem" , "shaymaa"},
            {"abed"}
    };
    int x = 0 , y = 0 ;
    for (int i = 0; i < list.length; i++) {
        for (int j = 0; j < list[i].length; j++) {
            if (list[i][j].equals("shaymaa")){
                x = i ;
                y = j ;
            }
        }
    }
    System.out.println("Shaymaa it's in index [" +x+"]"+"["+y+"]");
}
}
```

Output:

```
Shaymaa it's in index [1][2]
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

(OOP)
Object Oriented Programming

قال تعالى: " وَلِلَّهِ الْعِزَّةُ وَلِرَسُولِهِ وَلِلْمُؤْمِنِينَ وَلَٰكِنَّ الْمُنَافِقِينَ لَا يَعْلَمُونَ "

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Class

الـ Class : هو بمثابة قالب أو نموذج يُستخدم لإنشاء الـ (Objects). يحتوي الكلاس على متغيرات و(Methods) التي تحدد خصائص ومهمات الـ Object التي سيتم إنشاؤها منه.

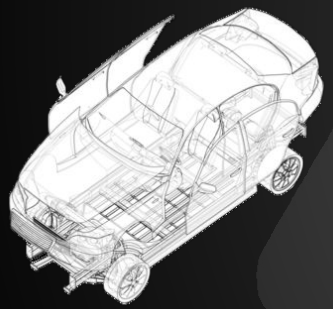
الـ Object : فهو نسخة محددة أو مثال ملموس& class معين. يمكنك التفكير في الClass كتصميم لسيارة، بينما الـ object هو السيارة الفعلية التي تم بناؤه وفقًا لذلك التصميم.

كل شيء بالحياة عبارة عن Object , مثلا أنا أصيل قدح أُعتبر Object من كلاس الإنسان , وأنت كذلك الأمر , ومثلا البيت الذي تسكن فيه عبارة عن Object من Class المنزل , والسيارات من كلاس السيارة , وهكذا على باقي الأمور .

هذا الـobject الذي تم انشاؤه من كلاس السيارة, وهو نسخة عن كلاس السيارة ولك له صفات يمكن ان تتشابه ويمكن ان تختلف عن باقي الobjects, فيمكن مثلا إنشاء سيارة تتشابه مع هذه ولكن تمتلك لون مختلف

```
Car mazda3 = new Car();
```

هذا هو الـ Class, وهو التصميم الهيكلي لـObjects السيارة
class Car



اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Class

```
class className{

    the variables :
    int , String ….

    the constructor :
    no-arg || arg

    the Methods

}
```

1- اسم الكلاس (`className`): يبدأ الكلاس بالكلمة المفتاحية class، ثم اسم الكلاس.

2- المتغيرات (`Variables`): شرح: المتغيرات هي الخصائص أو البيانات التي يمتلكها الكائن (object) الذي سيتم إنشاؤه من هذا الكلاس. أنواع المتغيرات: يمكن أن تكون الأنواع البدائية مثل int للأعداد الصحيحة، أو أنواعًا مرجعية مثل String للتعامل مع النصوص.

3- المُنشئ (`Constructor`): المُنشئ هو (method) خاصة تُستخدم لتهيئة الكائنات عند إنشائها. يُسمى بنفس اسم الكلاس ولا يحتوي على نوع إرجاع.

4- الأساليب (`Methods`): الدوال هي العمليات أو الأفعال التي يمكن للأوبجيكت تنفيذها. يتم استدعاء الدالة باستخدام أوبجيكت من الكلاس.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Object

```java
class Name {

    Name() {
        System.out.println("the name is empty ");
    }


    Name(String name) {
        System.out.println("the name is :" + name);
    }


    Name(String name1 , String name2){
        System.out.println("the name is :" +name1 + name2);
    }
}
```

(Object): هو عبارة عن وحدة أساسية لتمثيل نوع من الكيانات في العالم الحقيقي أو في النظام الذي تصممه. يتم إنشاء الكائنات بناءً على الكلاسات (Classes)، التي تعمل كقوالب تُستخدم لإنشاء الكائنات. Objects instance of classes

لإنشاء الـ object نحن بحاجة لعمل الـ constructor, والـ constructor هو ميثود خاصة ولكنه يحمل إسم الـ class ولا يمكن أن يكون له عائد مثل void,int.. ومن ميزاته أنه يُستدعى تلقائيا عند إنشاء الـ objects, وعند إنشاء الأوبجيكت يكون له نفس خصائص الكلاس الذي تم إنشائه منه

ويوجد 3 أنواع من الـ constructors وهي

1- default constructor : يتم إنشاؤها تلقائيا في حالة عدم إنشاء واحدة من المستخدم, وهو يعتبر no-arg constructor

2- no-arg constructor : لا يحتوي متغيرات في تعريفه

3- arg constructor : يحتوي متغيرات في تعريفه

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Constructor

```
class Name {
    // we dont have constructor
    String FirstName;
    String LastName;
}
```

ما هو (Default Constructor)؟

ال Default Constructor هو كونستركتر ينشئه الكومبايلر تلقائيًا في حال لم يقم المبرمج بتعريف أي منشئ صريح داخل الكلاس. يتم استخدامه لتهيئة الأوبجيكت عند إنشائها بقيم افتراضية.

- Initialize variables: يقوم ال Default Constructor باعطاء قيم المتغيرات داخل الـ object بالقيم الافتراضية الخاصة بنوع البيانات. مثلاً: العدد الصحيح (int): يتم تهيئته إلى 0. النصوص (String): يتم تهيئتها إلى null. الأنواع الأخرى: يتم تهيئتها إلى قيمها الافتراضية.

Name name = new Name() ;

في هذه الحالة لم يتم إنشاء constructor في كلاس الإسم, اذا سوف يتم إنشاء constructor by default

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Constructor

```
class Name {

    int x ;
    int y ;

    Name(String name1){
        System.out.println("the name is :"+name1);
    }
}
```

Name name = new Name() ;

في هذه الحالة نحن لم نقم بإنشاء no-arg constructor ولكننا أنشأنا arg constructor , اذاً لن يتم إنشاء constructor by default , وهكذا نحن لا نملك no-arg constructor , فناتج هذه العملية سيكون syntax error

وطبعا بما أننا لم نقم بإعطاء قيم للمتغيرات x,y فإنها تأخذ القيمة الافتراضية لها (0)

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Constructor

```java
public class Rectangle {
    int length ;
    int wide ;

    Rectangle(int newLength , int newWide){
        length=newLength ;
        wide=newWide ;
    }

    public int area(){
        return length*wide;
    }
}
```

هنا قمنا بإنشاء كلاس للمستطيل وقمنا بتعريف متغيرين فيه وهما الطول والعرض , ولكننا نريد أن نعطيهما قيم عندما يتم إنشاء الobject , إذاً نحن بحاجة إلى إنشاء constructor ووضع ال argumeant له حسب المعطيات التي انا بحاجة إلى إسناد قيمة لها

قمنا بإنشاء constructor with 2 arg واحدة فيهما للطول الجديد المراد إسناد قيمته , والآخر لإسناد قيمة العرض , فأصبحت المتغيرات في هذا الكلاس مساوية للتي ادخلناها بالـ object حين استدعائه

```java
Rectangle r = new Rectangle(4,5);

System.out.println(r.area()); //20
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Local variable

على فرض أننا قمنا بالconstructor بإنشاء متغيرات فيه تشابه أسماء المتغيرات المتواجدة بالكلاس , ويجب أن تعلم أن المتغيرات التي يتم إنشاؤها بـ methods or constructor تكون مرئية فقط على مستواهما, أي انك اذا قمت بإستدعاء المتغير خارجهما فإنك لم تعثر عليه , وفي حالة أنه تم إنشاء متغيران يحملان نفس الإسم الأول على مستوى الكلاس والثاني على مستوى الmethods or constructor , فإن الأولوية تكون للمتغيرات المحلية : local variable, وهذا يعني في هذه الحالة أن قيمة المتغيرات الخاصة بالكلاس لم تتغير وستبقى كما هي.

```
public class Rectangle {
    int length ;
    int wide ;

    Rectangle(int length , int wide){
        length=length ;
        wide=wide ;
    }

    public int area(){
        return length*wide;
    }
}
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Local variable

على فرض أننا قمنا بالconstructor بإنشاء متغيرات فيه تشابه أسماء المتغيرات المتواجدة بالكلاس , ويجب أن تعلم أن المتغيرات التي يتم إنشاؤها بـ methods or constructor تكون مرئية فقط على مستواهما, أي انك اذا قمت بإستدعاء المتغير خارجهما فإنك لم تعثر عليه , وفي حالة أنه تم إنشاء متغيران يحملان نفس الإسم الأول على مستوى الكلاس والثاني على مستوى الmethods or constructor , فإن الأولوية تكون للمتغيرات المحلية : local variable, وهذا يعني في هذه الحالة أن قيمة المتغيرات الخاصة بالكلاس لم تتغير وستبقى كما هي.

```
public class Rectangle {
    int length ;
    int wide ;

    Rectangle(int length , int wide){
        length=length ;
        wide=wide ;
    }

    public int area(){
        return length*wide;
    }
}
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Local variable

local variable : هو متغير يُعرّف داخل طريقة (method) معينة. يكون مرئيًا فقط داخل الطريقة التي يُعرّف فيها. يتم إنشاؤه عند دخول الطريقة ويتم تدميره عند الخروج منها.

خصائص الـ local variable:

يتم إنشاء المتغير المحلي عند استدعاء الـ method ويُحذف عند انتهاء تنفيذها.

visibility : مرئية فقط داخل الطريقة التي تُعرّف فيها ولا يمكن الوصول إليها من خارج الطريقة.

initialization: يجب تعريف الـ local variable واعطائه قيمة قبل استخدامه، حيث لا يتم تعيين قيم افتراضية تلقائيًا له.

```
public class Rectangle {
    int length ;
    int wide ;

    Rectangle(int length , int wide){
        Length = length ;
        Wide = wide ;
    }

    public int area(){
        return length*wide;
    }
}
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

this keyword

```
public class Rectangle {
    int length ;
    int wide ;

    Rectangle(int length , int wide){
        this.length=length ;
        this.wide=wide ;
    }

    public int area(){
        return length*wide;
    }
}
```

يمكننا لحل هذه المشكلة إستخدام (this)، وهذا المفتاح عند استخدامه فهو يعني أنني اريد استخدام المتغير المرئي على مستوى هذا الكلاس، أي أن قيمة المتغيرات الخاصة بالكلاس الآن سوف تصبح مساوية للقيم المدخلة بالـ constructor .

ما هو this؟

في جافا، this هو مرجع (reference) يشير إلى الأوبجيكت الحالي (current object) الذي يتم التعامل معه في سياق الكلاس. يُستخدم this لتحديد المتغيرات والميثود والكونستركتر الخاصة بالكائن الحالي، ولتجنب الالتباس بين المتغيرات المحلية والمتغيرات(المرئية على مستوى الكلاس) في الكلاس.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

this keyword

```
public class Rectangle {
    int length ;
    int wide ;

    Rectangle(int length , int wide){
        this.length=length ;
        this.wide=wide ;
    }

    public int area(){
        return length*wide;
    }
}
```

استخدامات this

- تمييز الـ local variables والـ global variables : عندما يكون لديك متغير محلي (مثل في الكونستركتر أو الميثود) يحمل نفس الاسم كمتغير global في الكلاس، يمكنك استخدام this لتمييزهم.
- استدعاء كونستركتر اخر : يمكنك استخدام this لاستدعاء كوستركتر أخرى داخل نفس الكلاس.
- تمرير الاوبجيكت الحالي: يمكنك استخدام this لتمرير الاوبجيكت الحالي كمعطى إلى ميثود أو اوبجيكت أخرى.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

this keyword

```java
public class Plate {
    int number ;
    String name ;
    Plate(){}

    Plate(int number){
        this.number=number ;
    }
         global      local

    Plate(String name){
        this.name=name ;
    }

    Plate(int number,String name){
        this.number=number ;
        this.name=name ;
    }

    public void print(){
        System.out.print(this.number);
        System.out.print(" "+name);
    }
}
```

1- تمييز المتغيرات الـ global عن المتغيرات الـ local: عندما يكون لديك متغير محلي (في method or constructor) يحمل نفس الاسم كمتغير global في الكلاس، يمكنك استخدام this لتمييزهم

```java
Plate p1 = new Plate;

 //error  لم يتم وضع أقواس للكونستكتر

Plate p2 = new Plate(1111);

// number = 1111, name = null

Plate p3 = new Plate("Aصeel");

 // number = 0, name = Aصeel

Plate p4 = new Plate(1111,"Aصeel");

 // number = 1111 name = Aصeel

Plate p5 = new Plate("Aصeel",1111);

 // error: we don't have a constructor name/number
```

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

this keyword

```
class Student {

    Student(){
        System.out.println("Student");
    }

    Student(int id){
        this();
        System.out.println("id="+id);
         // this(); X error

    }

    public static void main(String[] args) {
        Student Aseel = new Student(1223063);
    }

}
```

2- استدعاء كونستركتر آخر: يمكنك استخدام this لاستدعاء كوستركتر أخر داخل نفس الكلاس.

هنا تم تمرير الـ no-arg constructor بالـ arg constructor ، فيتم تنفيذ الكونستركترال no-arg أولا، ثم يتم تطبيق الأسطر الأخرى

Student Aseel = new Student(1223063);

مثلا هنا تم الإستدعاء، فيطبع أولا كلمة Student، ثم ينفذ السطر الثاني ويطبع id=1223063

ولكن يوجد شرط في هذه الحالة، هو أنه عند استدعاء الكونستركتر في كونستركتر آخر، يجب أن يكون الاستدعاء في أول سطر، فإذا وضعناه بعد السطر الأول سوف يحدث erorr

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

this keyword

```
class Student {

    int id;
    String name;

     Student(int id , String name){
        this.id = id;
        this.name = name;
    }

    public String displayDetails(Student stu){
         return "Student ID: " + stu.id +
                "\nStudent Name: " + stu.name;
    }

    public void print(){
        System.out.println(displayDetails(this));

    }
}
```

3- تمرير الاوبجيكت الحالي:

يمكنك استخدام this لتمرير الاوبجيكت الحالي كمعطى إلى ميثود أو اوبجيكت أخرى.

قمنا بعمل ميثود لاسترجاع معلومات الطالب من اسم ورقم إلى ستيرنج.

ثم قمنا بعمل ميثود لطباعة هذه المعلومات, وقمنا بتمرير الأوبجيكت الحالي في داخل معاملات الميثود

```
Student aseel = new Student(1223063 ,
"Aseel");

Student eman = new Student(1220611 , "Eman");

aseel.print();

eman.print();
```

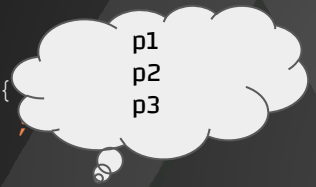
فرضا أننا قمنا بكتابة هذا الكود, فاستدعاء الميثود الأولى ستطبع لي محتويات الاوبجيكت الخاص بها (أصيل) والاستدعاء الثاني سيطبع لي محتويات الاوبجيكت الخاص بها (إيمان)

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Reference Equality

```java
public class Plate {
    int number ;
    String name ;
    Plate(){}

    Plate(int number){
        this.number=number ;
    }

    Plate(String name){
        this.name=name ;
    }

    Plate(int number,String name){
        this.number=number ;
        this.name=name ;
    }

    public void print(){
        System.out.print(this.number);
        System.out.print(" "+name);
    }
}
```




```java
Plate p1 = new Plate(1111,"Aصeel");

Plate p2 = p1;

Plate p3 = p2;
```

هنا، نقوم بإنشاء أوبجيكت جديد من نوع Plate وتعيين قيم له. هذا الأوبجيكت يُخزن في مكان معين في الذاكرة، و**p1** هو المرجع الذي يشيرإلى هذا المكان.

في هذه الخطوة، نعيّن p2 ليشيرإلى نفس الأوبجيكت الذي يشيرإليه p1. الآن، p1 و**p2** يشيران إلى نفس الاوبجيكت في الذاكرة.

p3 يتم تعيينه ليشيرإلى نفس الأوبجيكت الذي يشيرإليه p2. بما أن p2 يشيرإلى نفس الأوبجيكت مثل p1، فإن p3 يشيرأيضًا إلى نفس الأوبجيكت .

كيف يحدث ذلك؟

عندما تقوم بتعيين p1 = p2، فإنك لا تنشئ أوبجيكت جديدًا، بل تقوم ببساطة بنسخ مرجع p1 إلى p2. لذلك، كل من p1 و**p2** يشيران إلى نفس الأوبجيكت .

عملية p2 = p3 هي نفس العملية؛ يتم نسخ مرجع p2 إلى p3، مما يجعل p3 أيضًا يشيرإلى نفس الأوبجيكت .

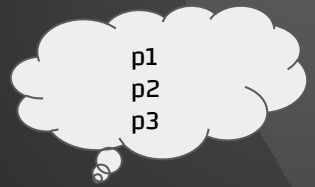
اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Reference Equality

```java
public class Plate {
    int number ;
    String name ;
    Plate(){}

    Plate(int number){
        this.number=number ;
    }

    Plate(String name){
        this.name=name ;
    }

    Plate(int number,String name){
        this.number=number ;
        this.name=name ;
    }

    public void print(){
        System.out.print(this.number);
        System.out.print(" "+name);
    }
}
```

```java
Plate p1 = new Plate(1111,"Aصeel");

Plate p2 = p1;

Plate p3 = p2;

p1.print(); // result = 1111 Aصeel

p2.number = 10 ;

p3.name = "eman" ;

p1.print(); // result = 10 Eman

System.out.println(p2==p3); // result = true
```



اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Array of Objects

```
ClassName[] arrayName;

arrayName = new ClassName[size];
```

مصفوفة الأوبجيكت " (Array of Objects)

مصفوفة الأوبجيكت في جافا هي نوع من المصفوفات التي تحتوي على مراجع إلى أوبجيكت من نوع معين. يمكن لمصفوفة الأوبجيكت تخزين عدة أوبجيكت من نفس النوع، ويمكن الوصول إلى كل أوبجيكت عبر مؤشره في المصفوفة.

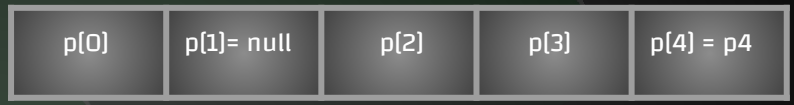
القيمة الافتراضية : عند إنشاء مصفوفة الأوبجيكت في جافا، يتم تعيين القيم الافتراضية لكل عنصر في المصفوفة. القيم الافتراضية تعتمد على نوع المصفوفة: للمصفوفات التي تحتوي على مراجع أوبجيكت (مثل Plate[]): القيمة الافتراضية هي null. يعني أن العناصر في المصفوفة لا تشير إلى أي أوبجيكت حتى يتم تعيين أوبجيكت فعلي لهم.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Array of Objects

```
Plate[ ] plates = new Plate[5] ;

plates[0] = new Plate(1111 ,"Aصeel");

plates[2] = new Plate(2000 , "eman");

plates[3] = new Plate(1595 , "hannen");

Plate plate4 = new Plate() ;

plates[4] = plate4;

for (int i = 0; i < plates.length; i++) {
   if(plates[i].number == 1111)
       plates[i].print();
}
```



في هذا العنصر من ال array لا يوجد object مخزن فيه, أي أن قيمته null ولا يمكن إستدعاء أي متغير او ميثود من null لأنها غير موجودة , فسوف يظهر على الشاشة error يعرف بـ:NullPointerException. ولتفاديه يجب أن نضع شرطاً أن العنصر المراد استدعائه لا يساوي null

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Array of Objects

```
Plate[ ] plates = new Plate[5] ;

plates[0] = new Plate(1111 ,"Aصeel");

plates[2] = new Plate(2000 , "eman");

plates[3] = new Plate(1595 , "hannen");

Plate plate4 = new Plate() ;

plates[4] = plate4;

for (int i = 0; i < plates.length; i++) {
   if (plates[i]!=null)
       if(plates[i].number == 1111)
       plates[i].print();
}
```

p[0]	p[1]= null	p[2]	p[3]	p[4] = p4

the output: 1111 Aصeel

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

static

على فرضاً أننا احتجنا إلى متغير يتشاركه جميع ال object , ما الطريقة لإنشاء هكذا متغير ؟static keyword، فما خصائصه ؟

1. **الملكية في الكلاس** static يعني أن العضو (سواء كان متغيرًا أو ميثود) ينتمي إلى الكلاس ككل، وليس إلى أي كائن معين من هذا الكلاس. يمكن الوصول إلى الأعضاء الثابتة مباشرة عبر اسم الكلاس، دون الحاجة لإنشاء كائن من الكلاس.

2. **التخزين المشترك** المتغيرات الثابتة تشارك نفس القيمة بين جميع أوبجيكت الكلاس. هذا يعني أن أي تغيير في قيمة المتغير الثابت يؤثر على جميع الأوبجيكت . يتم تخصيص مساحة واحدة فقط في الذاكرة للمتغير الثابت، وهذا يوفر ذاكرة ويقلل من الحاجة لتخزين نسخ متعددة من نفس البيانات.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

static

على فرضاً أننا احتجنا إلى متغير يتشاركه جميع ال object , ما الطريقة لإنشاء هكذا متغير ؟, static keyword , فما خصائصه ؟

3. **التحكم في الوصول** يمكن الوصول إلى الـ static من أي مكان في الكلاس باستخدام اسم الكلاس مباشرة، وهذا يمكن أن يكون مفيدًا في حالات معينة حيث تحتاج إلى الوصول إلى القيم أو الميثود دون الاعتماد على أوبجيكت .الستاتيك ميثود يمكنها فقط الوصول إلى الستاتيك الأخرى في الكلاس ولا يمكنها الوصول إلى non static

4. (**Static Blocks**) هي كتل من الكود تُنفذ مرة واحدة فقط عند تحميل الكلاس، ويمكن استخدامها لتهيئة المتغيرات الثابتة أو إعداد الكلاس.

5. **التقييد على this** لا يمكن استخدام this في الطرق الثابتة، لأن this يشير إلى الأوبجيكت الحالي، ولكن في الطرق الثابتة، لا يوجد أوبجيكت حالياً، بل يتعامل الكلاس بشكل كامل.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

static

```
class Name {
    static int counter = 0 ;

    Name() {
        counter++ ;
        System.out.println(counter);
    }

    Name(String name) {
        System.out.println(counter);
        counter++ ;
    }
    static void method(){
        counter+=10 ;
    }

    public static void main(String[] args) {
        Name name1 = new Name();
        Name name2 = new Name("aseel");
        Name name3 = new Name("eman");
        Name name4 = new Name("qadah");
        Name.method() ;
        System.out.println(Name.counter);
        name3.method() ;
        System.out.println(name1.counter);
    }

}
```

```
1
1
2
3
14
24
```

1. counter يصبح 1 ويطبع 1.

2. counter حاليًا 1 ويطبع 1. ثم يتم زيادة counter إلى 2.

3. counter حاليًا 2 ويطبع 2. ثم يتم زيادة counter إلى 3.

4. counter حاليًا 3 ويطبع 3. ثم يتم زيادة counter إلى 4.

```
5.Name.method();
```

يستدعي الـ static method، التي تزيد counter بمقدار 10. counter يصبح 14.

```
6.System.out.println(Name.counter);
```

يطبع قيمة counter، والتي هي 14.

```
7.name3.method();:
```

يستدعي الـ static method، عبر object name3.

counter يزيد بمقدار 10 مرة أخرى ليصبح 24.

```
8.System.out.println(name1.counter);:
```

يطبع قيمة counter عبر object name1. لأن counter ثابت، تكون قيمته 24.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

static

ما الذي يمكن للـ static method والـ instance method ان تستدعيه:

	static method	Instance method
Instance data field	false	true
static data field	true	true
Instance method	false	true
static method	true	true

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Modifiers

01 **public**

- يُمكن الوصول إلى العنصر المُعرف كـ public من أي كلاس آخر في البرنامج، سواء كان ذلك داخل نفسها (package) أو خارجها

02 **private**

- يُقصر الوصول إلى العنصر المُعرف كـ private على الكلاس الذي تم تعريفه فيه فقط. لا يمكن الوصول إليه من أي كلاس آخر، حتى لو كان ضمن نفس الـ package

03 **default**

- اذا لم يُحدد أي موديفاير، فإن العنصر يكون له مستوى وصول ديفولت حيث يُمكن الوصول إليه من أي كلاس داخل نفس الحزمة، لكن ليس من خارجها

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Getters && Setters

```java
public class Person {

    private String name;

    public String getName(){
        return name;
    }
    public void setName(String name){
        this.name = name;
    }

}
```

تعتبر السيتر (Setters) والجيتر (Getters) جزءًا مهمًا من مفهوم (Encapsulation). توفر وسيلة للوصول إلى المتغيرات الخاصة (private variables) في الكلاس بشكل منظم وآمن.

مفهوم (Encapsulation) هو مبدأ برمجي يتضمن حماية البيانات داخل الكلاس من التعديلات غير المرغوب فيها والوصول المباشر من خارج الكلاس. يتم تحقيق ذلك عن طريق:

- تحديد المتغيرات كـ private: بحيث تكون متاحة فقط داخل الكلاس.
- استخدام طرق السيتر والجيتر للوصول إلى وتحديث القيم الخاصة بالمتغيرات.

1. (Getters): رُستخدم لقراءة (أو الحصول على) قيمة المتغيرات الخاصة. توفر وسيلة آمنة للوصول إلى البيانات دون السماح بالتعديل المباشر عليها, يجب أن تكون من نفس نوع البيانات الذي سنقوم بإنشاء الميثود له, لكي تستطيع ارجاع قيمته

2. (Setters): تُستخدم لتحديث (أو تعيين) قيمة المتغيرات الخاصة. توفر وسيلة آمنة لتحديث البيانات مع إمكانية تنفيذ منطق إضافي إذا لزم الأمر, ويكون من نوع void

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Getters && Setters

مثال توضيحي:

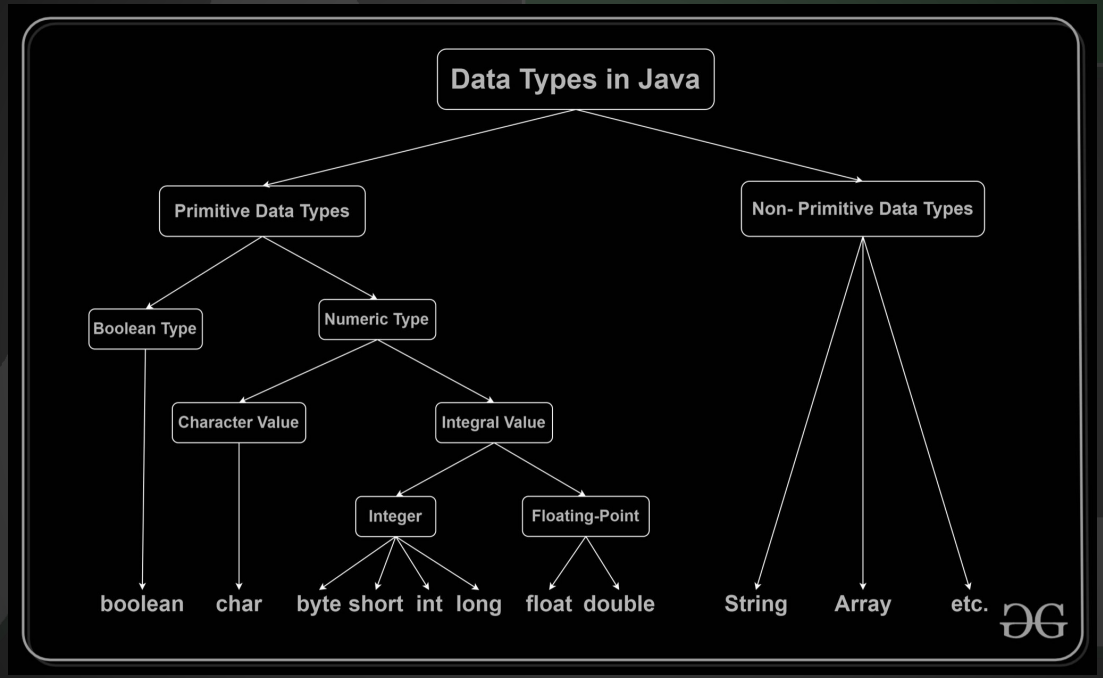
يوضح مبدأ عمل السيتر والجتير

```java
class Person {

    private String name;

    Person(String name){
        this.name = name;
    }

    public String getName() {
        return name;

    }
    public void setName(String name) {
        this.name = name;
    }

     public static void main(String[] args) {
         Person person = new Person("Aseel");
         System.out.println(person.getName()); // print Aseel
         person.setName("Eman");
         System.out.println(person.getName()); // print Eman
     }

}
```

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Primitive / Reference: data dype



اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Primitive / Reference: data dype

```java
public class Main {
    public static void main(String[] args)
{
        int x = 5 ;
        increment(x);
        System.out.println("x = "+x);
    }
    public static void increment(int x) {
        x = x + 1;
    }
}
```

لكن ماذا لو قمنا بعمل هذا الشيء, هل سيتأثر المتغير x وتزداد قيمته ؟

المخرجات :

```
C:\Users\
x = 5
```

طبعا لن تتأثر قيمة المتغير,

لأنه عند تمرير قيمة من نوع primitive يتم نسخ قيمة المعطى في متغير جديد داخل الدالة, مما يعني أن التغييرات التي تحدث على المعطى داخل الدالة لا تؤثر على المتغير الاصلي.

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Primitive / Reference: data dype

```java
class Example {

    static int[] arr ;

    public static void rev(int[] arr){
        int[] temp =new int[arr.length];

        for(int i=0; i< arr.length; i++){
            temp[arr.length-1-i] = arr[i];
        }

        for(int i=0; i< arr.length; i++){
            arr[i] = temp[i];
        }

    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        rev(arr);

        for(int i : arr){
            System.out.print(i);
        }
    }
}
```

2. (Reference Data Types – non primitive) عندما يتم تمرير refrence إلى method يتم تمرير refrence الأوبجيكت (عنوان الذاكرة) وليس الأوبجيكت نفسه:

(Pass by Reference): يعني أن الميثود تتلقى ريفرنس الأوبجيكت الأصلي. أي تغيير في الأوبجيكت داخل الميثود سيؤثر على الأوبجيكت الأصلي.

مثال توضيحي:

عند استدعاء rev(arr) :

الخطوة الأولى: يتم نسخ عناصر المصفوفة الأصلية arr إلى مصفوفة جديدة temp في ترتيب معكوس.

الخطوة الثانية: يتم نسخ العناصر من temp إلى arr، مما يحدث تعديلاً في المصفوفة الأصلية arr.

نتيجة الكود: بعد تنفيذ الطريقة rev, تكون المصفوفة arr قد تم عكس ترتيب عناصرها. وعند طباعة العناصر، سترى النتيجة المعكوسة:

Output : 5 4 3 2 1

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Immutable Types

في جافا، مفهوم (Immutable Types) يشير إلى الأوبجيكت التي لا يمكن تعديل حالتها بعد إنشائها. الأوبجيكت غير القابلة للتغيير تحافظ على قيمتها الأصلية مما يوفر أمانًا أكبر للكود.

الخصائص الأساسية لـ Immutable Types :

- عدم القابلية للتعديل: بعد إنشاء اوبجيكت من النوع immutable، لا يمكن تعديل حالته الداخلية. أي عملية تبدو وكأنها تعدل الاوبجيكت في الواقع، تنشئ اوبجيكت جديد.

- الموديفير الثابتة: يمكن أن تكون جميع الحقول الخاصة (private) وغير قابلة للتغيير (final)، ويكون الوصول إليها فقط من خلال طرق لا تعدل الحالة (getters).

String هو مثال لنوع immutable في جافا..

طبعا لو عرفنا سترينج ثم قمنا بتغيير قيمتها فإننا فعليا نغير الريفرنس الخاص بالسترينج ونجعله يؤشر على موقع اخر يحمل قيمة مختلفة, اما السترينج نفسه لا يمكن تعديل شيء من خصائصه.

يوجد نوع اخر من السترينج يمكن تعديله سيتم شرحه لاحقا

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Immutable Types

```java
final class Student {

    private final String name;
    private final int age;

    public Student(String name, int age){
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

طريقة بناء الـ Immutable class :

1- الكلاس final: نعرف الكلاس كـ final لضمان عدم إمكانية توارثه. هذا يحافظ على عدم إمكانية تعديل سلوكه من خلال الكلاسات الفرعية (سيتم شرح هذا لاحقا في ال inheritance).

2- المتغيرات private final : المتغيرات خاصة و final، مما يعني أنها يمكن تعيينها فقط مرة واحدة عند إنشاء الكائن.

3- الكونستركتر : يعين القيم للمتغيرات . بما أن المتغيرات final، يتم تعيينها فقط في الكونستركتر .

4- (gettar) توفر الوصول للمتغيرات ولكن لا تسمح بتعديلها.

```java
Student student = new Student("Aseel", 20);
```

في حالة أننا قمنا بهذه العملية فهذا الاوبجيكت لا يمكن التعديل على قيمه بتاتا, ولكن يمكن تغيير الريفرنس الخاص به هكذا

```java
student = new Student("Eman", 18);
```

هكذا تم تغيير الريفرنس الخاص بالأوبجيكت إلى ريفرنس اخر بالذاكرة, ولكن أين يذهب الريفرنس الذي كان يؤشر عليه سابقا ؟

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

OOP(Object-Oriented Programming)

Garbage Collection

```java
class Student {

    String name;
    int age ;

    public static void change(String name , int age) {
        name = "Changed";
        age = 18;
    }

    public static void main(String[] args) {

        String word = "Hello";
        word = "Hi" ;

        String name = "Aseel" ;
        int age = 20 ;
        change(name , age);
        System.out.println("Name: " + name + " Age: "+ age);

    }
}
```

Garbage Collection : هو آلية تلقائية في Java تقوم بإدارة وتحسين استخدام الذاكرة عن طريق تنظيف الذاكرة من الكائنات التي لم يعد هناك حاجة إليها. بمعنى آخر، هو عملية إزالة الكائنات غير المستخدمة من الذاكرة لتحريرها للاستخدام من قبل كائنات جديدة.

1- المتغيرات التي يتم تعريفها بداخل الميثود والكوستركتر يتم إزالتها بعد إتمام مهمة الميثود او الكوستركتر , أي أنه بمجرد إنتهاء هذه الميثود سيزول المؤشر changed الذي تم تعريفها بداخلها, مما سيؤدي إلى إنهياره حين الخروج من الميثود, لهذا تبقى قيمة المتغيرات كما هي في هذه الحالة

2- عند تغيير المؤشر الخاص بالأوبجيكت لمؤشر اخر يتم إزالة المؤشر الأول بما أنه أصبح غير مستخدم

Chapter (10)

StringBuilder, Relations , Wrapper and Big Classes

قال تعالى: " الْيَوْمَ تُجْزَىٰ كُلُّ نَفْسٍ بِمَا كَسَبَتْ ۚ لَا ظُلْمَ الْيَوْمَ ۚ إِنَّ اللَّهَ سَرِيعُ الْحِسَابِ "

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

StringBuilder

يعتبر كلاس StringBuilder واحد من الكلاس الأساسية التي تستخدم لمعالجة الستريغ بطريقة فعالة. تختلف StringBuilder عن كلاس String في أنها توفر إمكانية تعديل السلاسل النصية بدلاً من إنشاء أوبجيكت جديد في كل مرة يتم تعديل النص.

1. كلاس StringBuilder موجود في باكيج java.lang، وهي تستخدم لإنشاء وتعديل الستريغ المتغيرة. على عكس String، حيث يكون الأوبجيكت غير قابل للتعديل (immutable)، فإن الأوبجيكت من نوع StringBuilder قابلة للتعديل (mutable). هذا يعني أنه يمكن تعديل محتوى StringBuilder بدون إنشاء أوبجيكت جديد في الذاكرة.

2. لماذا نستخدم StringBuilder؟

عندما نستخدم String لتعديل النصوص، يتم إنشاء أوبجيكت جديد في كل مرة يتم إجراء تعديل على النص الأصلي، مما يؤدي إلى زيادة في استخدام الذاكرة وزيادة الوقت المستغرق. StringBuilder يحل هذه المشكلة عن طريق تعديل النص الأصلي نفسه دون الحاجة إلى إنشاء أوبجيكت جديد.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

StringBuilder

```
class StringBuilder {

    public StringBuilder(){
        this(16);   // default capacity
    }

    public StringBuilder(String s){
        // constructor implementation
    }

    public StringBuilder(int capasty){
        // constructor implementation
    }

}
```

يوجد لدينا 3 كونستركتر للـ StringBuilder .

```
StringBuilder str = new StringBuilder();
```

1- no argument : يقوم بإنشاء أوبجيكت StringBuilder فارغ بسعة افتراضية (عادة 16 حرفًا).

```
StringBuilder str = new StringBuilder("Aseel");
```

2- (String) : يقوم بإنشاء كائن StringBuilder يحتوي على السلسلة النصية "Aseel".

```
StringBuilder str = new StringBuilder(50);
```

3- int : يقوم بإنشاء أوبجيكت StringBuilder بسعة أولية مقدارها 50 حرفًا، ولكن بدون محتوى نصي.

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

StringBuilder Class
Simple method for StringBuilder object

	Methods	الشرح
1	`append(char[] data)`	يضيف مصفوفة من الحروف إلى الـ StringBuilder.
2	`append(char[] data, int offset, int len)`	يضيف جزء من مصفوفة الحروف إلى StringBuilder، بدءًا من offset وطول len.
3	`append(String s)`	يضيف قيمة من نوع بيانات أساسي (int, float, boolean, إلخ) كجزء من السلسلة.
4	`append(primitiveType v)`	يضيف قيمة من نوع بيانات أساسي (int, float, boolean, إلخ) كجزء من السلسلة.
5	`delete(int start, int end)`	يحذف النص من السلسلة بين الموضعين start و end.
6	`deleteCharAt(int index)`	يحذف الحرف في الموضع المحدد (index).
7	`insert(int offset, char[] data)`	يقوم بإدراج مصفوفة من الحروف عند الموضع المحدد في السلسلة النصية.

StringBuilder Class
Simple method for StringBuilder object

	Methods	الشرح
8	`insert(int offset, primitiveType b)`	يقوم بإدراج قيمة من نوع البيانات الأساسي عند الموضع المحدد في السلسلة النصية.
9	`insert(int offset, primitiveType b)`	يقوم بإدراج سلسلة نصية (String) عند الموضع المحدد في السلسلة النصية.
10	`insert(int offset, String s)`	يستبدل النص بين الموضعين start و end بالنص الجديد s.
11	`replace(int start, int end, String s)`	يستبدل النص بين الموضعين start و end بالنص الجديد s.
12	`reverse()`	يقوم بعكس ترتيب الحروف في السلسلة النصية.
13	`setCharAt(int index, char ch)`	يستبدل الحرف في الموضع المحدد بـ ch.
14	`length()`	إرجاع طول النص الحالي
15	`capacity()`	إرجاع السعة الحالية لـ StringBuilder

StringBuilder methods

```java
StringBuilder sb  = new StringBuilder("Hello");

sb.append(" World");

System.out.println(sb);  // Output: Hello World
```

1. append

هذه الميثود تستخدم لإضافة بيانات (نصوص أو قيم) إلى نهاية StringBuilder.

```java
StringBuilder sb  = new StringBuilder("Hello World");

sb.delete(5, 11);

System.out.println(sb);  // Output: Hello
```

2. delete

هذه الميثود تستخدم لحذف مجموعة من الحروف من StringBuilder بين موقعين معينين.

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

StringBuilder methods

```java
StringBuilder sb  = new StringBuilder("Hello
World");

sb.deleteCharAt(5);

System.out.println(sb);  // Output: HelloWorld
```

3. deleteCharAt

تستخدم لحذف حرف واحد بناءً على موقعه.

```java
StringBuilder sb  = new StringBuilder("Hello World");

sb.insert(5, ",");

System.out.println(sb);  // Output: Hello, World
```

4. insert

هذه الميثود تستخدم لإدخال نص أو قيمة في مكان معين داخل StringBuilder.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

StringBuilder methods

```java
StringBuilder sb = new StringBuilder("Hello World");

sb.replace(6, 11, "Java");
System.out.println(sb);  // Output: Hello Java
```

5. replace

تستخدم لاستبدال مجموعة من الحروف داخل StringBuilder بين موقعين معينين بنص جديد.

```java
StringBuilder sb = new StringBuilder("Hello");

sb.reverse();

System.out.println(sb);  // Output: olleH
```

6. reverse

تقوم بعكس ترتيب الحروف في StringBuilder.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

StringBuilder methods

```java
StringBuilder sb = new StringBuilder("Hello");

sb.setCharAt(1, 'a');

System.out.println(sb);  // Output: Hallo
```

7. setCharAt

تقوم بتغيير حرف معين في موقع محدد داخل StringBuilder.

```java
StringBuilder sb = new StringBuilder("Hello");

System.out.println(sb.capacity());

// Default capacity + length of string (16 + 5 = 21)

System.out.println(sb.length());

// Output: 5
```

8. Capacity && length

Capacity تقوم بإرجاع السعة الحالية لـ StringBuilder
(المساحة المتاحة قبل إعادة تخصيص الذاكرة).

Length تقوم بإرجاع عدد الحروف الموجودة في الـ
StringBuilder.

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

StringBuilder methods

```java
StringBuilder sb = new StringBuilder("Hello");

char ch = sb.charAt(1);

System.out.println(ch);  // Output: e
```

9. charAt

تقوم بإرجاع الحرف الموجود في موقع معين.

```java
StringBuilder sb  = new StringBuilder("Hello World");

String subStr  = sb.substring(6, 11);

System.out.println(subStr);  // Output: World
```

10. substring

تقوم بإرجاع جزء معين من النص في StringBuilder بناءً على موقعين معينين.

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

Wrapper Classes

	Primitive Data Types	Wrapper Classes
1	int	Integer
2	char	Character
3	float	Float
4	double	Double
5	boolean	Boolean
6	byte	Byte
7	short	Short
8	long	Long

تُستخدم "Wrapper Classes" لتغليف (أو تحويل) الأنواع الأساسية (Primitive Data Types) إلى (Objects). السبب في وجودها هو أن Primitive Data Types ليست أوبجيكت في جافا، ولا يمكن استخدامها في السياقات التي تتطلب أوبجيكت (مثل ArrayList) والتي لا تعمل إلا مع الاوبجيكت .

في جافا، لدينا 8 أنواع بريميتيف وهي:

int,char,float,double,boolean,byte,short,long هذه الأنواع بدائية لأنها لا تحتوي على ميثود أو متغيرات ؛ هي فقط تمثل القيم.

للتعامل مع هذه الأنواع ككائنات، قامت جافا بتوفير مجموعة من "Wrapper Classes" التي تغلف هذه المتغيرات داخل أوبجيكت . هذا يعني أن لكل نوع بدائي في جافا هناك كلاس يلتف حوله ويقدمه كأوبجيكت.

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

Wrapper Classes

```java
// Autoboxing
int primitiveValue = 5;
Integer wrapperObject = primitiveValue;

// Unboxing
Integer wrapperObject = new Integer(10);
int primitiveValue = wrapperObject;
```

Autoboxing و Unboxing:

Autoboxing: هو عملية تحويل تلقائي للأنواع الـ primitive إلى wrapped objects الخاصة بها.

Unboxing: هو عملية التحويل العكسية، أي تحويل wrapped object إلى نوعه primitive.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Wrapper Classes

	Methods Integer Class	الشرح
1	`parseInt(String s)`	يحول سلسلة نصية إلى قيمة عددية من نوع int.
2	`valueOf(String s)`	يحول سلسلة نصية إلى Integer.
3	`toString(int i)`	تحويل عدد صحيح إلى نص.
4	`compare(int x, int y)`	مقارنة عددين صحيحين.
5	`max(int a, int b)`	يعيد القيمة الأكبر بين عددين صحيحين.
6	`min(int a, int b)`	يعيد القيمة الأصغر بين عددين صحيحين.
7	`sum(int a, int b)`	يعيد مجموع عددين
8	`toBinaryString(int x)`	يعيد String للرقم بالنظام الثنائي 10 -> 1010

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

Big (Integer && Decimal)

```java
import java.math.BigInteger;

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

BigInteger num1 = new BigInteger("12345678901234567890123456789O");
BigInteger num2 = new BigInteger("98765432109876543210987654321O");

        // الجمع
        BigInteger sum = num1.add(num2);
        System.out.println("Sum: " + sum);

        // الطرح
        BigInteger difference = num1.subtract(num2);
        System.out.println("Difference: " + difference);

        // الضرب
        BigInteger product = num1.multiply(num2);
        System.out.println("Product: " + product);

        // القسمة
        BigInteger quotient = num1.divide(num2);
        System.out.println("Quotient: " + quotient);

        // باقي القسمة
        BigInteger remainder = num1.mod(num2);
        System.out.println("Remainder: " + remainder);

        // التحقق مما إذا كان العدد أوليًا
        boolean isPrime = num1.isProbablePrime(1);
        System.out.println("Is prime: " + isPrime);
    }
}
```

كلاس BigInteger في Java يستخدم للتعامل مع الأرقام الكبيرة جدًا التي تتجاوز حدود النوعين int و long.

يمكن لـ BigInteger تخزين أرقام صحيحة (Integers) بحجم غير محدد، مما يجعله مفيدًا في التعامل مع العمليات الحسابية الكبيرة مثل التشفير أو الحسابات الرياضية المعقدة.

ميزات BigInteger

1- حجم غير محدود: يدعم BigInteger التعامل مع الأرقام الصحيحة ذات الحجم الكبير جدًا.

2- عدم القابلية للتغيير: جميع الكائنات من نوع BigInteger غير قابلة للتغيير (immutable)، مما يعني أنه عند إجراء أي عملية حسابية يتم إنشاء أوبجيكت جديد يمثل النتيجة بدلاً من تعديل الأوبجيكت الأصلي.

3-دعم للعمليات الحسابية: يدعم العمليات الحسابية الشائعة مثل الجمع والطرح والضرب والقسمة، بالإضافة إلى العمليات الرياضية المعقدة مثل القسمة الطويلة والحسابات النسبية.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

Big (Integer && Decimal)

```java
BigDecimal a = new BigDecimal("123.45");

BigDecimal b = new BigDecimal("678.90");

BigDecimal result = a.add(b); //result = 802.35

BigDecimal result = a.subtract(b); // result = -555.45

BigDecimal result = a.multiply(b); // result = 83777.205

int result = a.compareTo(b); // result = -1

BigDecimal result = a.pow(2); // result = 15229.9025

BigDecimal result = a.abs(); // result = 123.45
```

كلاس BigDecimal في Java مصمم للتعامل مع الأرقام العشرية الكبيرة بدقة عالية، مما يجعله خيارًا ممتازًا للحسابات المالية والتجارية حيث يتطلب الدقة العالية لتجنب الأخطاء الناتجة عن الفقد في الدقة في الأنواع العائمة مثل float أو double.

ميزات BigDecimal

1- دقة عالية: يوفر BigDecimal دقة حسابية عالية، مما يجعله مثاليًا للعمليات المالية.

2- عدم القابلية للتغيير: BigDecimal هو كذلك غير قابل للتغيير (immutable)، مما يعني أن كل عملية حسابية تنتج أوبجيكت جديد.

3- دعم للعمليات الحسابية المعقدة: يدعم عمليات مثل الجمع، الطرح، الضرب، القسمة، والتحكم بدقة القسمة والتقريب.

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

RelationShip

العلاقات في Java تعتبر جزءًا مهمًا جدًا من (OOP). هي التي تصف كيفية ارتباط الـ (objects) ببعضها البعض وتفاعلها ضمن النظام. هناك أنواع متعددة من العلاقات التي يمكن أن توجد بين الكائنات في Java، وأهمها هي:

1- Association

2- Aggregation

3- Composition

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

RelationShip

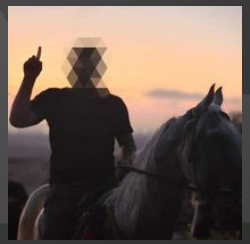
```java
class Teacher {
    public String name;
    public Teacher(String name) {
        this.name = name;
    }
}

class Student {
     String name;
    public Student(String name) {
        this.name = name;
    }
    public void attendClass(Teacher teacher) {
        System.out.println(name + " is attending class
                           with " + teacher.name);
    }
}
public class Main {
    public static void main(String[] args) {
        Teacher teacher = new Teacher("أصيل قدح");
        Student student = new Student("وسيم جودت");
        student.attendClass(teacher);
    // output : وسيم جودت is attending class with أصيل قدح
    }
}
```

1. (Association)

تشير إلى علاقة عامة بين 2 أوبجيكت. يمكن لأوبجيكت واحد استخدام أو التعامل مع أوبجيكت آخر. تكون هذه العلاقة غير مملوكة، أي أن الأوبجيكت التي تشارك في العلاقة يمكن أن تكون موجودة بشكل مستقل.

مثال: العلاقة بين Teacher و Student. المعلم يمكن أن يتعامل مع عدة طلاب، وكذلك الطالب يمكن أن يتعامل مع عدة معلمين.

Teacher



Student

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

RelationShip

```java
class Car {
    String model;
    public Car(String model) {
        this.model = model;
    }
}

class Person {
    String name;
    Car car;

    public Person(String name, Car car) {
        this.name = name;
        this.car = car;
    }

    public void showInfo() {
        System.out.println(name + " owns a " + car.model);
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car("mazda");
        Person person = new Person("morhaf", myCar);
        person.showInfo();
    }
}
```

2. (Aggregation)

هي نوع من الـ association حيث يكون للأوبجيكت "جزء" من أوبجيكت آخر، ولكن الأجزاء يمكن أن توجد بشكل مستقل عن الأوبجيكت الكلي. بمعنى آخر، يمكن للأوبجيكت المالك أن توجد حتى لو تم تدمير الأوبجيكت الكلي.

مثال: كلاس Person يحتوي على أوبجيكت Car كجزء منها، ولكن السيارة ليست مملوكة بالكامل للفئة Person، مما يعني أنه يمكن استخدام نفس الأوبجيكت Car من قبل أوبجيكت أخرى إن وجدت. وهذا هو مفهوم (Aggregation).

Has - a

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.

RelationShip

```java
class Room {
    String name;
    public Room(String name) {
        this.name = name;
    }
}
class House {
    Room[] rooms ;
    public House() {
        rooms = new Room[5];
    }
    public void addRoom(Room room) {
        for (int i = 0; i < rooms.length; i++) {
            if (rooms[i] == null) {
                rooms[i] = room;
                break;
            }
        }
    }
    public void listRooms() {
        System.out.println("Rooms in the house:");
        for (Room room : rooms) {
            if (room != null)
            System.out.println(room.name);
        }
    }
}
public class Main {
    public static void main(String[] args) {
        House house = new House();
        house.addRoom(new Room("Living Room"));
        house.addRoom(new Room("Kitchen"));
        house.listRooms();
    }
}
```

3. العلاقة (Composition)

العلاقة هي نوع أقوى من العلاقة association حيث يكون الأوبجيكت المكون جزءًا من الأوبجيكت الكلي ولا يمكن أن يوجد بدون الأوبجيكت الكلي. إذا تم تدمير الأوبجيكت الكلي، يتم أيضًا تدمير الأجزاء المكونة.

مثال: العلاقة بين House و Room. الغرف لا يمكن أن توجد بدون المنزل.

part of

ملاحظة: بما أن الغرفة هي جزء من المنزل ولا يمكن أن تتواجد الغرفة بدون المنزل, فيتم إنشاء أوبجيكت الغرفة في داخل معطيات الميثود أو الكونستركتر الخاصة بالمنزل, هنا لدينا ميثود لإضافة الغرف , فنضيفها بالشكل التالي

وهذا هو الفرق بين العلاقتين

اللهم انصر المجاهدين وثبّت أقدامهم, واحفظ أهلنا في قطاع غزة.

اللهم استخدمنا ولا تستبدلنا، انفعنا وانفع بنا، اصطفينا واصنعنا على عينك
واصطنعنا لنفسك، سُدّ بنا ثغور أمتك واكفنا شرّ الرياء وابعد عنا التخاذل
والتقاعس، وجازنا بما أنت أهله، ولا تفتنّا ولا تستبدلنا ولا تحرمنا يا الله

قال تعالى: "وَآخِرُ دَعْوَاهُمْ أَنِ الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ"

اللهم انصر المجاهدين وثبّت أقدامهم، واحفظ أهلنا في قطاع غزة.