First Day

1-Variables
2-Constants
3-Comments
4-Datatypes
5-Operators

مقدمة في Java

المتغيرات Variables

أثناء عملك مع لغة java، ستحتاج في مرحلةٍ ما إلى التعامل مع البيانات. وعند رغبتك في تخزين تلك البيانات، فإنك ستحتاج الى شيء يقوم بتخزينها لك، وهذا هو عمل المتغيرات. يمكن تخيل المتغيرات والنظر إليها على أنها الأوعية أو الحاويات التي تحتوي وتُخزن كل ما يوضع فيها.
إذاً، يمكن النظر للمتغير على أنه أسلوب بسيط لتخزين البيانات واسترجاعها بشكل مؤقت أثناء عمل البرنامج.

تعريف المتغيرات باستخدام int

. لتوضيح الفكرة لاحظ معي المثال التالي:

```
int age;
```

في المثال أعلاه قمنا بإنشاء و تعريف متغير بإسم `age`، وذلك لحفظ قيمة العُمر بداخله. الآن دعنا نقوم بإسناد قيمة العُمر `26` له:

```
age = 26;
```

لاحظ أننا قمنا باستخدام المتغير `age` في هذه المرة بدون استخدام التعبير `var`، وذلك لأنها تستخدم مرة واحدة فقط وهي أثناء تعريف المتغير، وبعد ذلك، سنتعامل مع المتغير بشكل مباشر من خلال اسمه فقط، وفي هذه الحالة هو `age`.
يمكنك أيضاً اختصار الخطوتين السابقتين في خطوة واحدة كالتالي:

```
int age = 26;
```

نلاحظ في المثال أعلاه أنه يمكننا تعريف المتغير وإسناد القيمة إليه في آنٍ واحد.
إذاً، يُمكن للمبرمج استخدام `age` في أماكن مُختلفة من البرنامج، وسيتم استبدالها بالقيمة `26`.
بالإضافة إلى ذلك، يمكن للمبرمج أن يقوم بتغيير قيمة المتغير أثناء البرنامج. فمثلاً، بعد تعريفنا للمتغير السابق `age`، يمكننا تغيير قيمته إن أردنا، ولتوضيح الفكرة لاحظ معي المثال التالي:

```
age = 30;
```

الثوابت Constants

تختلف الثوابت `constants` عن المُتغيرات `variables` في أنه **لا يمكن تغيير قيمتها بعد إسناد أول قيمة لها،** وستظل قيمة الثابت كما هي طيلة فترة البرنامج. لتعريف ثابت في `java` نستخدم كلمة `final` متبوعة بنوع المتغير.
لتوضيح الفكرة، لاحظ معي المثال التالي:

```
final int weekDays = 7;
```

أيام الأسبوع هي دائماً 7 أيام، أي أنه لا يمكننا تغيير عدد أيام الأسبوع، وعليه، يمكن تمثيلها في ثابت في البرنامج كما سبق ورأينا. ومن الأمثلة على الثوابت؛ تاريخ الميلاد، فالتاريخ الذي ولدت فيه ثابت ولن يتغير. يوضح السطر التالي تعريف ثابت لتمثيل تاريخ الميلاد.

```
final String dateOfBrith = "10/02/1990";
```

التعليقات Comments

في حالات معينة أثناء كتابة الكود، قد يحتاج المبرمج إلى وضع بعض الملاحظات أو التعليقات. فمثلاً، قد يحتاج إلى وضع ملاحظة لتذكيره بتعديل كود معين، فيقوم المبرمج حينها بكتابة بعض الملاحظات بجانب ذلك الكود للعودة إليه فيما بعد. وفي حالات أخرى، قد يعمل على الملف البرمجي أو المشروع البرمجي أكثر من شخص، وقد يحتاج أحد المبرمجين إلى أن يضع بعض الملاحظات لأعضاء الفريق، وهكذا. توفر التعليقات في java طريقة تُساعد المبرمج على كتابة ما يود من ملاحظات في البرنامج، وبالنسبة للغة java فإنها ستتجاهل تلك التعليقات، ولن تنظر لها على أنها تعليمات ستقوم بتنفيذها.

سنتحدث في هذا الجزء عن أنواع التعليقات في java، وهي:

- تعليق السطر الواحد Single Line Comment
- تعليق متعدد الأسطر Multi-Line Comment

تعليق السطر الواحد Single Line Comment

عند رغبتنا في وضع تعليق في سطر واحد أو single-line comment، والذي سينتهي بنهاية السطر، سنستخدم `//` كعلامة لبداية التعليق. يوضح السطر التالي هذه الفكرة.

```
// This is a comment.
```

ليس بالضرورة أن يبدأ التعليق من بداية السطر، فقد يكون التعليق هو جزء من سطر برمجي. لتوضيح الفكرة، لاحظ معي المثال التالي:

```
int age = 25; // This is my age.
```

تعليق متعدد الأسطر Multi-line Comments

في بعض الحالات، قد نحتاج إلى كتابة تعليق طويل، يمتد على أكثر من سطر. في هذه الحالة، يمكننا استخدام أسلوب التعليق متعدد الأسطر Multi-Line Comment. ونقوم بذلك عن طريق كتابة الملاحظات بين `/* */`. يوضح المثال التالي هذا الأمر.

```
/* Write your
   comments here..
*/
```

قد يستخدم البعض أسلوب التعليق متعدد الأسطر كتعليق سطر واحد. لتوضيح الفكرة، لاحظ معي المثال التالي:

```
/* Write your comment here.. */
```

وبنفس الأسلوب، يمكنك استخدامها عند وجود سطر برمجي. لتوضيح الفكرة، لاحظ المثال التالي:

```
int age = 25; /*This is my age.*/
```

التسميات Naming

للتسميات في لغة java شروط ومن غير الممكن تسمية المتغيرات او الثوابت اذا خالفة هذه الشروط.

١. لا يمكن تسمية متغير يحتوي على كلمتين، لتوضيح الفكرة، لاحظ المثال التالي:

```
String my name = "khalid"; //Wrong
```

و عوضا عن ذلك نقوم بتسمية المتغيرات التي تحتوي على كلمتين عن طريق استخدام اسلوب كتابة Camel case, عن طريق كتابة اول كلمة بحرف ضغير ثم اول حرف من كل كلمة يكون حرفا كبيرا. لتوضيح الفكرة، لاحظ المثال التالي:

```
String myName = "Khalid"; //Correct
```

2. لا يمكن التسمية بأسماء تحتوي في داخلها على احدى الرموز الخاصة بالعمليات مثل علامة الجمع + و علامة لطرح الى آخره، لتوضيح الفكرة، لاحظ المثال التالي:

```
String +name = "Khalid" //Wrong
```

٣. لا يمكن التسمية بأسماء محجوزة في اللغة مثل كلمة `var` الخاصة بتعريف المتغيرات او كلمة `final` الخاصة بتعريف الثوابت

```
String final = "Khalid" //Wrong
```

انواع البيانات Datatype

تدعم لغة java عدداً من أنواع البيانات. يوضح الجدول التالي هذه الأنواع.

الوصف	النوع
لتمثيل الأعداد الصحيحة.	int
يستخدم لتمثيل الارقام التي تحتوي على النقطة العشرية	double
ويستخدم لتمثيل أنواع البيانات النصية مثل characters والنصوص strings.	String
أي بيانات من هذا النوع تكون ضمن قيمتين وهما true و false.	boolean
يستخدم لتمثيل الأحرف	char

فيما يلي، ستتم مناقشة كل نوع من هذه الأنواع بشكل مفصل.

تعريف متغير من نوع String

يمكننا استخدام ثلاث طرق مختلفة للتعامل مع النصوص على النحو التالي:

```
String message = "Welcome to java";
```

طريقة دمج النصوص باستخدام علامة "+"

يمكننا دمج نصين أو أكثر ليكونا نصا واحدًا، كما يمكننا دمج متغير ونص على النحو التالي:

```
String itemTwo = "java";
```

```
String message = "Welcome to " + itemTwo;
```

النصوص ومفهوم Escape Character

تستخدم هذه العمليات داخل النص String و كل واحدة منها تقوم بعملية محددة فعلى سبيل المثال \n يجعل ما بعده على سطر جديد و منها ، يوضح الجدول التالي عمليات Escape Character:

وظيفتها	رمز العملية	اسم العملية
يضيف عدد من المسافات في مكان وضعه	**\t**	**Horizontal Tab**
يقوم بجعل ما بعده على سطر جديد	**\n**	**Newline**
تقوم باضافة ' مكان وضعها	**\'**	**Single quote**
تقوم باضافة " مكان وضعها	\"	**Double quote**
تقوم باضافة \ مكان وضعها	\\	**Backslash**

تعريف متغير من نوع Number

يُعرّف المتغير من نوع number كتعريف المتغير العادي، ويُسند إليه قيمة رقم. لاحظ معي المثال التالي:

```
int valueType = 2;
```

تعريف متغير من نوع Double

يُعرّف المتغير من نوع Double كتعريف المتغير العادي، ويُسند إليه قيمة رقم عشري. لاحظ معي المثال التالي:

```
double valueType = 2.0;
```

تعريف متغير من نوع Boolean

يُعرّف المتغير من نوع boolean كسائر المتغيرات، ولكن يتميز النوع boolean عن غيره من بقية الأنواع أنه يحتوي على قيمتين فقط، أي أن أي متغير يكون نوعه boolean فستكون قيمته إما true أو false. لتوضيح الفكرة، لاحظ معي المثال التالي:

```
boolean value = true;
```

تعريف التغيرات بنوع المتغير

تحتوي لغة java على ميزة safe type، اي مبعنى انها تمتلك ميزة الأمان عند تعريف المتغيرات، لاحظ معي المثال التالي:

```
int intValue = 2;
```

في هذا المثال قنا بتعريف متغير من نوع عدد صحيح اي بمعنى انه لا يمكننا اسناد قيمة غير قيم الاعداد الصحيحة ولو اسندنا له قيمة عدد بخلاف هذا النوع فسوف يظهر لنا خطأ لأننا اسندنا قيمة لمتغير لا يقبل هذا النوع من القيم، لاحظ معي المثال التالي:

```
int intValue;
intValue = "Not int";
// Error: Type mismatch: cannot convert from String to intJava (16777233)
```

كذلك يمكننا تعريف بقية الأنواع الاخرى بإستخدام نوع المتغير لاحظ معي المثال التالي:

```
int intValue = 2;
double doubleValue  = 2.1;
String stringVale = "StringValue";
char characters = 's';
bool boolValue = true;
```

العمليات Operators

هناك عدد من العمليات المختلفة التي يمكنك استخدامها أثناء البرمجة، مثل العمليات الرياضية وعمليات المقارنة والعمليات المنطقية وغيرها من العمليات المُختلفة. سنتحدث في هذا الجزء عن مجموعة من أهم العمليات التي توفرها لغة java.

العمليات الحسابية Arithmetic Operators

ببساطة، يمكنك تنفيذ العمليات الرياضية المختلفة باستخدام الصيغة التالية:

```
result = left op right
```

حيث يمثل `op` نوع العملية الرياضية المُراد استخدامها، ويمثل كل من `left` و `right` القيمتين (أو المتغيرين أو الثابتين) اللذين سيتم تنفيذ العملية `op` عليهما. يوضح الجدول التالي أنواع العمليات الحسابية:

اسم العملية	رمز العملية	وظيفتها
Addition	+	تقوم بتنفيذ عملية الجمع.
Subtraction	-	تقوم بتنفيذ عملية الطرح.
Division	/	تقوم بتنفيذ عملية القسمة.
Multiplication	*	تقوم بتنفيذ عملية الضرب.

لتوضيح الفكرة، دعنا نقوم باستبدال `op` بأحد العمليات السابقة، وسنقوم هنا باختيار الجمع `+` كمثال يمكن تطبيقه على باقي العمليات الأخرى. يوضح السطر التالي هذا الأمر:

```
int result = 5 + 2;
```

في المثال أعلاه، قمنا بتنفيذ عملية الجمع باستخدام `+` وسيتم تخزين ناتج العملية -وهو في هذه الحالة 7- في المتغير `result`.

عمليات المقارنة Comparison Operators

يمكنك تنفيذ عمليات المقارنة المختلفة باستخدام الصيغة التالية **(مع التنبيه على أنه يمكنك استخدامها في سياقات برمجية أخرى دون إسنادها إلى قيمة، مثل استخدامها كشرط مع جملة if كما سنرى لاحقًا)**:

```
result = left op right
```

يمثل `op` نوع عملية المقارنة المُراد استخدامها ويمثل كل من `left` و `right` القيمتين(أو المتغيرين أو الثابتين) اللذين سيتم تنفيذ العملية `op` عليهما. وستكون نتيجة عمليات المقارنة هي قيمة من نوع `boolean`، أي أن ناتج المقارنة سيكون إما `true` أو `false`. يوضح الجدول التالي هذا الأمر:

اسم العملية	رمز العملية	وظيفتها
Greater Than	>	تعيد true في حال كان left **أكبر من** right.
Less Than	<	تعيد true في حال كان left **أصغر من** right.
Greater Than or Equal	>=	تعيد true في حال كان left **أكبر من أو يساوي** right.
Less Then or Equal	<=	تعيد true في حال كان left **أصغر من أو يساوي** right.

Equal	==	تعيد true في حال كان left **يساوي** right من حيث القيمة فقط.
Not Equal	!=	تعيد true في حال كان left لا **يساوي** right من حيث القيمة فقط.

دعنا نقوم الآن باستبدال `op` بأحد العمليات السابقة، وسنقوم هنا باستبدالها بأكبر من `>` ، ونستخدمها بصيغة java على النحو التالي:

```
boolean result = 5 > 2;
```

في هذه الحالة، قمنا بتنفيذ عملية المقارنة باستخدام `>` وسيتم تخزين الناتج -وهو `true`- في المتغير `result`.

العمليات المنطقية Logical Operators

هناك ثلاث عمليات منطقية، اثنتان منهما تكتب بالصيغة التالية(**مع التنبيه على أنه يمكنك استخدامها في سياقات برمجية أخرى دون إسنادها إلى قيمة، مثل استخدامها كشرط مع جملة if كما سنرى لاحقًا**):

```
result = left op right
```

يمثل `op` نوع العملية المنطقية المُراد استخدامها ويمثل كل من `left` و `right` القيمتين(أو المتغيرين أو الثابتين) اللذين سيتم تنفيذ العملية `op` عليهما. وستكون نتيجة العمليات المنطقية هي قيمة من نوع `boolean`، أي أن ناتج المقارنة سيكون إما `true` أو `false`.

أما بالنسبة للعملية المتبقية، أي العملية الثالثة، فهي تكتب بالصيغة التالية :

```
result = op right
```

يوضح الجدول التالي العمليات المنطقية:

اسم العملية	رمز العملية	وظيفتها
And	&&	ستكون النتيجة true في حالة واحدة فقط، وهي إن كانت left و right كلاهما true.
Or	||	ستكون النتيجة false في حالة واحدة فقط، وهي إن كانت left و right كلاهما false.
Not	!	يقوم بعكس قيمة right. في حال كانت right تساوي true فستكون النتيجة

هي false، والعكس صحيح.		

توضح الأسطر التالية استخدام العمليات السابقة برمجياً:

```
boolean first = true;
boolean second = false;
boolean andResult = first && second; // false
boolean orResult = first || second; // true
boolean notResult = !(5 == 10); // true
```

نظرة على Increment و Decrement

من العمليات المتكررة أثناء البرمجة، عملية زيادة واحد على قيمة المتغير الحالية أو إنقاص واحد منها. تسمى عملية الزيادة في هذه الحالة Increment وتُسمى عملية الإنقاص Decrement. لتوضيح الفكرة العامة، لاحظ معي الأسطر التالية:

```
int number = 5;
number = number + 1; // 6
number = number - 1; // 5
```

في السطر الثاني، قمنا بزيادة واحد على قيمة `number`، لتصبح القيمة 6، وهذا هو المقصود بمفهوم Increment. وقمنا في السطر الثالث بإنقاص واحد من قيمة `number`، لتصبح 5، وهذا هو المقصود بمفهوم Decrement.

توفر لغة java طريقة مُختصرة لتنفيذ كلتا العمليتين السابقتين، وذلك من خلال استخدام معامل الزيادة `++` لزيادة واحد على قيمة المتغير، ومعامل الإنقاص `--` لإنقاص واحد من قيمة المتغير. لتوضيح الفكرة، دعنا نقوم بإعادة كتابة المثال السابق بالطريقة المُختصرة في المثال التالية:

```
int number = 5;
number++; // number = number + 1 (increment)
number--; // number = number - 1 (decrement)
```