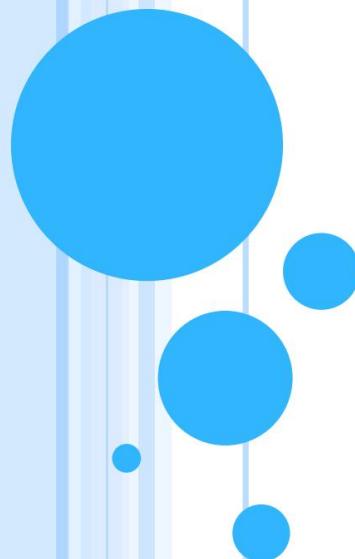


ICS102-  
برمجة الكمبيوتر

الفصل الأول:  
البدء مع  
جافا



## الخطوط العريضة

□ مقدمة إلى جافا.

□ مستويات لغة الكمبيوتر. □ المترجمون وكود المصدر وجهاز Java الظاهري □ رسائل الخطأ (بناء الجملة ووقت التشغيل والمنطق النحو والدلالة). □ خطأ).

□ المتغيرات في لغة جافا.

□ فئة السلسلة وطرقها في جافا.

## مقدمة إلى جافا

□ معظم الناس على دراية بلغة Java كلغة لتطبيقات الإنترنـت

□ سوف ندرس لغة Java كهدف عام  
لغة برمجة

□ سيكون بناء جملة التعبيرات والمهام مشابهًا لتلك الخاصة باللغات عالية المستوى الأخرى

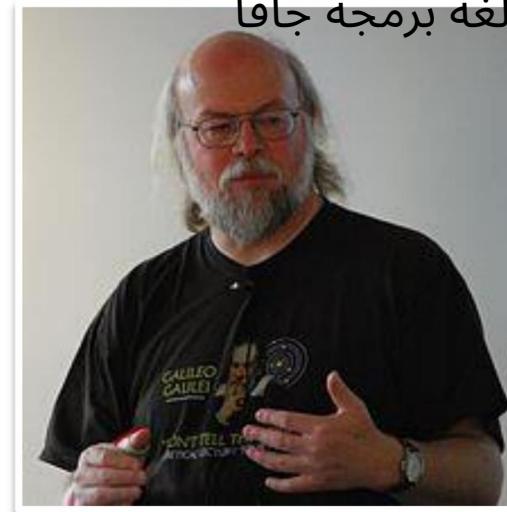
□ من المحتمل أن تكون التفاصيل المتعلقة بالتعامل مع السلسل ومخرجات وحدة التحكم الجديدة

## مقدمة إلى جافا: أصول جافا

### لغة

تم إنشاؤه بواسطة فريق Sun Microsystems بقيادة جيمس جوسلينج (1991) المعروف باسم والد

لغة برمجة جافا



في 27 يناير ، 2010، استحوذت شركة أوراكل على شركة صن مقابل 7.4 مليار دولار أمريكي.

[www.Oracle.com/us/sun/index.htm](http://www.Oracle.com/us/sun/index.htm)

## مقدمة إلى جافا: الكائنات والأساليب

جافا هي لغة برمجة كائنية التوجه (OOP).

منهجية البرمجة التي تنظر إلى البرنامج على أنه تتكون من كائنات تتفاعل مع بعضها البعض عن طريق الإجراءات (وتسمى الأساليب)

يقال أن الكائنات من نفس النوع لها نفس الشيء اكتب أو تكون في نفس الفصل

## برامح تطبيقات جافا

هناك نوعان من برامح Java: التطبيقات والتطبيقات الصغيرة

---

برامح تطبيق Java أو برامح "العادي" هو فئة ذات طريقة تسمى main

عند تشغيل برامح تطبيق Java، يقوم نظام وقت التشغيل تلقائياً باستدعاء الطريقة المسماة main

---

تبدأ جميع برامح تطبيقات Java بالبرامح الرئيسي طريقة

## تطبيقات جافا والتطبيقات الصغيرة

برنامج Java الصغير (تطبيق Java الصغير) هو برنامج Java مصمم للتشغيل من متصفح الويب

يمكن تشغيله من موقع على شبكة الإنترنت

يمكن أيضاً تشغيله باستخدام برنامج عارض صغير لتصحيح الأخطاء

تستخدم التطبيقات الصغيرة دائمًا واجهة النوافذ

في المقابل، قد تستخدم برامج التطبيقات واجهة نافذة أو وحدة تحكم (أي نص)  
للإدخال/الإخراج

سنقوم بعمل تطبيقات صغيرة في وقت لاحق من الفصل الدراسي

## مستويات لغة الكمبيوتر

### لغة رفيعة المستوى: لغة يتحدث بها الناس

يستطيع القراءة والكتابة والفهم

يجب ترجمة البرنامج المكتوب بلغة عالية المستوى إلى لغة يمكن للكمبيوتر فهمها قبل تشغيله

### لغة الآلة: لغة يستطيع الحاسوب فهمها

لغة منخفضة المستوى: لغة الآلة أو أي لغة مشابهة للغة الآلة المترجم: برنامج يقوم بترجمة برنامج لغة عالية المستوى إلى برنامج لغة منخفض المستوى مكافئ

تسمى عملية الترجمة هذه بالتجمیع

## كود بايت و Java الظاهري آلة

يقوم المترجمون لمعظم لغات البرمجة بترجمة البرامج عالية المستوى مباشرة إلى لغة الآلة لجهاز كمبيوتر معين

نظرًا لأن أجهزة الكمبيوتر المختلفة لها أجهزة مختلفة اللغات، هناك حاجة إلى مترجم مختلف لكل واحدة

في المقابل، يقوم مترجم Java بترجمة البرامج إلى كود، وهي لغة الآلة لجهاز كمبيوتر وهي يسمى Virtual Java آلة

بمجرد تجميعه إلى كود بايت، يمكن استخدام برنامج Java على أي جهاز كمبيوتر، مما يجعله محمولاً للغاية

## مصطلحات البرنامج

الكود : برنامج أو جزء من برنامج

كود المصدر (أو البرنامج المصدر): برنامج مكتوب بلغة عاليّة المستوي مثل Java

الإدخال لبرنامج المترجم

رمز الكائن: البرنامج ذو المستوى المنخفض المترجم

الإخراج من برنامج المترجم، على سبيل المثال، Java

رمز البايت

في حالة Java byte-code ، يتم الإدخال إلى مترجم Java byte-code.

## محمول الفئة

□ تنقسم برامج جافا إلى أجزاء أصغر تسمى الفئات

□ عادة ما يكون كل تعريف فئة في ملف منفصل ويتم تجميعه بشكل منفصل

□ محمول الفئة: برنامج يربط رمز البايت للفئات اللازمة لتشغيل جافا

برنام

□ في لغات البرمجة الأخرى، يسمى البرنامج المقابل رابط

## تجميع برنامج أو فئة جافا

يجب أن يكون كل تعريف فئة في ملف

الاسم هو نفس اسم الفئة متبوعاً .java -ب

يجب أن تكون فئة FirstProgram في ملف يسمى FirstProgram.java

يتم تجميع كل فئة باستخدام الأمر javac

متبوعاً باسم الملف الذي يوجد به الفصل

javac FirstProgram.java

والنتيجة هي برنامج بait كود اسمه الملف هو نفس اسم الفئة متبوعاً .class -ب

FirstProgram.class

## تشغيل برنامج جافا

يمكن إعطاء برنامج `java` أمر التشغيل (`java`) بعد تجميع جميع فئاته

قم بتشغيل الفصل الذي يحتوي على الطريقة الرئيسية فقط  
(سيقوم النظام تلقائياً بتحميل وتشغيل الفئات الأخرى، إن وجدت)

الطريقة الرئيسية تبدأ بالسطر:

الفراغ العام الثابت الرئيسي (`String[] args`)

اتبع أمر التشغيل حسب اسم الفصل  
فقط (لا يوجد ملحق `.avaj` أو `.ssalc`)

برنامج جافا الأول

## بناء الجملة والدلالات

### □ النحو : ترتيب الكلمات و

علامات الترقيم القانونية في اللغة، والقواعد النحوية للغة

### □ الدلالات: معنى الأشياء المكتوبة

أثناء اتباع قواعد بناء الجملة للغة

## نصيحة: رسائل الخطأ

### الخطأ : خطأ في البرنامج

□ تسمى عملية إزالة الأخطاء بالتصحيح

### خطأ نحوي: خطأ نحوي في البرنامج

□ يمكن للمترجم اكتشاف هذه الأخطاء، وسيقوم بإخراج ملف رسالة خطأ توضح ماهية الخطأ وأين يوجد الخطأ

### خطأ وقت التشغيل: خطأ لا يتم اكتشافه حتى يتم تشغيل البرنامج

□ لا يستطيع المترجم اكتشاف هذه الأخطاء: لا يتم إنشاء رسالة خطأ بعد التجميع، ولكن بعد التنفيذ

### خطأ منطقي: خطأ في الخوارزمية الأساسية للبرنامج

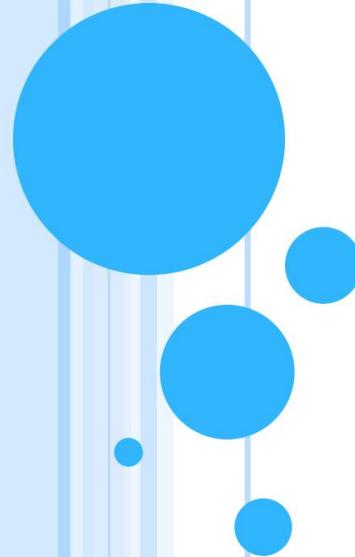
□ لا يستطيع المترجم اكتشاف هذه الأخطاء، ولا يوجد خطأ يتم إنشاء الرسالة بعد التجميع أو التنفيذ، لكن البرنامج لا يقوم بما يفترض به القيام به

## اتفاقيات التسمية

ابداً أسماء المتغيرات والفئات والأساليب والكائنات بحرف صغير، وحدد حدود "الكلمة"  
حرف كبير، واقتصر الأحرف المتبقية على أرقام وأحرف صغيرة

أعلى سرعة البنك معدل 1 مرة من الوصول

تبدأ أسماء الفئات بأحرف كبيرة  
خطاب، وبخلاف ذلك، الالتزام بالقواعد المذكورة أعلاه  
لتحيطMag الأول ماي كلاس



## المتغيرات في جافا

## الإعلانات المتغيرة

□ يجب أن يكون كل متغير في برنامج جافا  
أعلن قبل استخدامه

□ يخبر تعريف المتغير المترجم بنوع البيانات (النوع) التي سيتم تخزينها في المتغير

□ نوع المتغير يتبعه اسم متغير واحد أو أكثر مفصولة بفواصل، وتنتهي بفاصلة منقوطة

□ عادة ما يتم الإعلان عن المتغيرات قبل استخدامها مباشرة أو في بداية الكتلة (يشار إليها بقوس افتتاحي { )

□ تسمى الأنواع الأساسية في Java بالأنواع البدائية

`int numberOfBeans;`

وزن واحد مزدوج، الوزن الإجمالي:

## الأنواع البدائية

Display 1.2 Primitive Types

Type Name	Kind of Value	Memory Used	Size Range
boolean	true or false	1 byte	not applicable
char	single character (Unicode)	2 bytes	all Unicode characters
byte	integer	1 byte	-128 to 127
short	integer	2 bytes	-32768 to 32767
int	integer	4 bytes	-2147483648 to 2147483647
long	integer	8 bytes	-9223372036854775808 to 9223372036854775807
float	floating-point number	4 bytes	$-3.40282347 \times 10^{38}$ to $-1.40239846 \times 10^{-45}$
double	floating-point number	8 bytes	$\pm 1.76769313486231570 \times 10^{308}$ to $\pm 4.94065645841246544 \times 10^{-324}$

## بيانات المهمة مع

### الأنواع البدائية

في Java، يتم استخدام بيان المهمة لتغيير قيمة المتغير

يتم استخدام علامة التساوي (=) كعامل تخصيص

بيان المهمة يتكون من متغير  
الجانب الأيسر من عامل التشغيل، وتعبير على الجانب الأيمن من عامل  
 التشغيل

متغير = التعبير؛

يتكون التعبير من متغير أو رقم أو مزيج من المتغيرات والأرقام وعوامل التشغيل وأو  
استدعاءات الطريقة

= 98.6؛ درجة الحرارة！

count = numberOfBeans;

## بيانات المهمة مع

### الأنواع البدائية

#### □ عند تنفيذ بيان المهمة،

يتم تقييم التعبير أولاً، ثم يتم تعين المتغير الموجود على الجانب الأيسر من علامة التساوي مساوياً لقيمة التعبير

المسافة = المعدل \* الزمن؛

□ لاحظ أنه يمكن أن يحدث متغير على جانبي عامل التعين

العد = العد + 2؛

□ يتم تنفيذ عامل المهمة تلقائياً من اليمين إلى اليسار، بحيث يمكن ربط عبارات المهمة

number2 = number1 = 3؛

## نصيحة: تهيئة المتغيرات

متغير تم الإعلان عنه ولكن تم ذلك

ويقال إن عدم إعطاء قيمة بعد بطريقة ما غير مهياً

في بعض الحالات، يتم إعطاء متغير غير مهياً قيمة افتراضية

ومن الأفضل عدم الاعتماد على هذا

المتغيرات التي تمت تهيئتها بشكل صريح لها فائدة إضافية  
تحسين وضوح البرنامج

## نصيحة: تهيئة المتغيرات

□ يمكن الجمع بين الإعلان عن المتغير وتهيئته عبر بيان الإسناد

= عدد صحيح : 0

= المسافة المزدوجة : .5 \*

درجة الحرف = 'أ' :

□ لاحظ أنه يمكن تهيئة بعض المتغيرات ويمكن أن يظل بعضها الآخر غير مهياً في نفس الإعلان

```
int originalCount = 50, FinalCount;
```

## التهيئة الافتراضية المتغيرة

□ تتم تهيئة متغيرات المثيل تلقائياً

في جافا

□ تتم تهيئة الأنواع المنطقية إلى خطأ

□ تتم تهيئة البدائيات الأخرى إلى الصفر الخاص بها

يكتب

□ تتم تهيئة أنواع الفئات إلى قيمة فارغة

□ ومع ذلك، من الأفضل تهيئة متغيرات الحالة بشكل صريح في المنشئ

□ ملاحظة: لا تتم تهيئة المتغيرات المحلية تلقائياً

## قواعد أسماء المتغيرات [1]

يجب أن يبدأ اسم المتغير بحرف أو شرطة سفلية (\_ ) أو علامة الدولار ( \$ )

9.

قد يحتوي اسم المتغير على حروف وأرقام من 0 إلى

يجب ألا يحتوي اسم المتغير على مسافة أو أحرف خاصة.

يمكن أن يكون اسم المتغيرات بأي طول، ولكن كن حذراً. Java حساسة لحالة الأحرف  
 مما يعني أحرف كبيرة

الأحرف تختلف عن الأحرف الصغيرة  
 الشخصيات.

لا يمكن أن يكون اسم المتغيرات أيّاً من كلمات Java الأساسية (كلمة محظوظة)

## أمثلة على أسماء المتغيرات

الأسماء المسموح بها	الأسماء غير المسموح بها
درجة	الصف (اختبار)
GradeOnTest	اختبار الصف رقم 1
Grade_On_Test	3_درجة_الاختبار
\$راتب	اختبار الصف
_درجة	كتافة العملات
الصف الرئيسي1	

## تهيئة المتغيرات و أمثلة على الإعلان

□ المتغيرات في جافا المعلنة على النحو التالي: `اسم`  
متغير نوع البيانات: `على سبيل المثال: العمر;` `معدل التراكمي المزدوج;` `علم المنطقى.`

□ تتم تهيئة المتغيرات على النحو التالي: `نوع البيانات =`  
`القيمة;` `على سبيل المثال: العمر` `variableName`  
`= المعدل التراكمي المزدوج` `32: 3.5: علامة منطقية =`  
صحيح:

## بيانات التعيين المختصرة

مثال:	أي ما يعادل:
= العد : 2	= العد + العد : 2
= المبلغ - الخصم:	= المبلغ - المبلغ = الخصم:
* = المكافأة؛ 2	* = المكافأة = المكافأة؛ 2
= الوقت /	= الوقت =
import org.rushFactor;	= الوقت / عامل الاندفاع:
% = التغيير: 100	% = التغيير: التغيير: 100
* = المبلغ	* = المبلغ = المبلغ *
العد 1 + العد 2:	(العدد + العدد 1) 2:

## التوافق مع المهام

□ بشكل عام، لا يمكن تخزين قيمة نوع واحد في متغير من نوع آخر // **غير قانوني** int **intVariable** = 2.99; // **يؤدي**  
المثال أعلاه إلى **عدم** تطابق النوع

لأنه لا يمكن تخزين قيمة مزدوجة في متغير int

□ ومع ذلك، هناك استثناءات لهذا : doubleVariable = 2;

□ على سبيل المثال، يمكن تخزين قيمة int في قيمة مزدوجة يكتب

## التوافق مع المهام

بشكل عام، يمكن إسناد قيمة من أي نوع في القائمة التالية إلى متغير من أي نوع يظهر على يمينه

بأيت `قصير` `إنت` `طويل` `تعويم` `مزدوج`  
شار

لاحظ أنه كلما تحركت إلى أسفل القائمة من اليسار إلى اليمين، يصبح نطاق `القيم المسموح بها` للأنواع أكبر

يلزم وجود نوع صريح لتعيين قيمة من نوع واحد لمتغير يظهر نوعه على يساره في القائمة  
أعلاه (على سبيل المثال، `double to int`)

لاحظ أنه في `Java` لا يمكن تعيين `int` لمتغير من النوع `boolean` ولا يمكن تعيين  
`int` لمتغير من النوع `boolean`

# العوامل الحسابية و التعبيرات

كما هو الحال في معظم اللغات، يمكن تشكيل التعبيرات في Java باستخدام المتغيرات والثوابت والعوامل الحسابية

**□ هذه العوامل هي + (الجمع)، - (الطرح)، × (الضرب)، / (القسمة)، و٪ (الباقي) \***

□ يمكن استخدام التعبير في أي مكان يكون فيه من القانوني استخدام قيمة من النوع الناتج عن التعبير

إذا تم دمج العامل الحسابي مع int المعاملات، ثم النوع الناتج هو

إذا تم دمج عامل حسابي مع واحد أو اثنين من المعاملات المزدوجة ، فإن النوع الناتج هو مزدوج

## الأقواس وقواعد الأسبقية

□ يمكن وضع تعبير بين قوسين بالكامل لتحديد التعبيرات الفرعية التي يتم دمجها مع كل عامل بالضبط

□ إذا تم حذف بعض أو كل الأقواس في التعبير ، فسوف تتبع Java قواعد الأسبقية لتحديد، في الواقع، مكان وضعها

□ ومع ذلك، فمن الأفضل (وأحياناً من الضروري) تضمينها

## قواعد الأسبقية

### Display 1.3 Precedence Rules

---

#### *Highest Precedence*

First: the unary operators: +, -, ++, --, and !

Second: the binary arithmetic operators: \*, /, and %

Third: the binary arithmetic operators: + and -

#### *Lowest Precedence*

---

## قسم الأعداد الصحيحة والنقطة العائمة

□ عندما يكون أحد المعاملين أو كليهما من نوع الفاصلة العائمة ، فإن القسمة تؤدي إلى نوع النقطة العائمة

15.0/2 يتم تقييمه إلى 7.5

□ عندما يكون كلا المعاملين من النوع الصحيح، يتم القسمة يؤدي إلى نوع عدد صحيح

□ يتم تجاهل أي جزء كسري لا يتم تقرير الرقم

15/2 يقيم إلى 7

□ كن حذرًا لجعل واحدًا على الأقل من المعاملات من نوع الفاصلة العائمة إذا كان الجزء الكسري مطلوبًا

المشغل \_ %

□ يتم استخدام العامل % مع المعاملات من النوع int لاستعادة المعلومات المفقودة بعد إجراء عملية تقسيم الأعداد الصحيحة

15/2 يقيم على الحاصل 7

15%2 تقيم على الباقي 1

□ يمكن استخدام عامل التشغيل % للعد بمقدار 2 أو 3 أو أي رقم آخر

□ للعد اثنين قم بإجراء العملية رقم ، 2% وعندما تكون النتيجة 0 يكون الرقم زوجي

## نوع الصب

يأخذ قالب النوع قيمة من نوع واحد وينتج قيمة من نوع آخر بقيمة "مكافأة"

إذا كان  $m$  و  $n$  عددين صحيحين ليتم تقسيمهما، و يجب الحفاظ على الجزء الجزئي من النتيجة، ويجب أن يتم تحويل واحد منها على الأقل إلى نوع النقطة العائمة قبل إجراء عملية القسمة

الجواب المزدوج =  $n / (مزدوج) m$ :

لاحظ أنه يتم وضع النوع المطلوب بالداخل قوسين مباشرة أمام المتغير الذي سيتم الإدلاء به

لاحظ أيضاً نوع وقيمة المتغير المراد يلقي لا يتغير

## مزيد من التفاصيل حول نوع الصب

□ عند تحويل الكتابة من نقطة عائمة إلى نوع عدد صحيح، يتم اقتطاع الرقم، وليس تقريبه

□ يتم تقييمه إلى 2 وليس 3

□ عندما يتم تعين قيمة نوع عدد صحيح لمتغير من نوع الفاصلة العائمة، تقوم Java بإجراء تحويل تلقائي للنوع يسمى إكراه النوع

= مزدوج د : 5

□ في المقابل، من غير القانوني وضع قيمة مزدوجة في متغير int بدون نوع صريح

إنت ط // : 5.5 = غير قانوني

□ صحيح int i = (int)5.5 //

## عوامل الزيادة والنقصان

□ يضيف عامل الزيادة  $(++)$  واحداً إلى قيمة المتغير

□ إذا كانت  $n$  تساوي 2، فإن  $n++$  أو  $n+=$  ستغير قيمة  $n$  إلى 3

□ يقوم عامل التناقص  $(--)$  بطرح واحد من قيمة المتغير

□ إذا كانت  $n$  تساوي 4، فإن  $--n$  أو  $n--$  ستغير قيمة  $n$  إلى 3

## عوامل الزيادة والنقصان

□ عندما يسبق أي عامل متغيره، ويكون جزءاً من تعبير، يتم تقييم التعبير باستخدام القيمة المتغيرة للمتغير

□ إذا كانت  $n$  تساوي 2 فإن  $(n++)^2$  تكون قيمتها 6

□ عندما يتبع أي عامل متغيره، ويكون جزءاً من تعبير، يتم تقييم التعبير باستخدام القيمة الأصلية للمتغير، وعندها فقط يتم تغيير قيمة المتغير

□ إذا كانت  $n$  تساوي 2 فإن  $(n++)^2$  تكون قيمتها 4

## أمثلة الزيادة والنقصان

الطبقة العامة {Incr}

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

كثافة العمليات ج:

ج = 5;

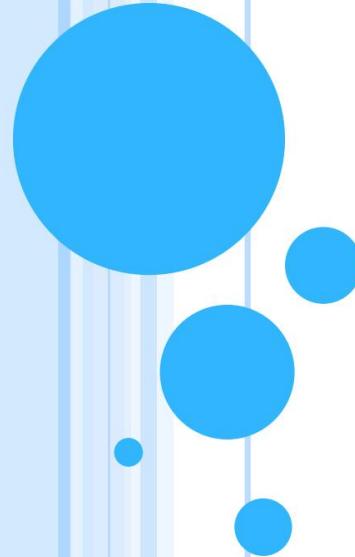
طباعة 5 System.out.println(); ج /

System.out.println(c++); //

طباعة 6 System.out.println(); ج //

تخطي السطر System.out.println(); //

ج = 5;



## سلسلة الفصل -

## سلسلة الفصل \_

لا يوجد نوع بدائي للسلسل في Java  
 الفئة هي فئة محددة مسبقاً في Java تستخدم لتخزين السلسل ومعالجتها  
 الكائنات من النوع String تكون من سلسل من الأحرف المكتوبة بين علامتي اقتباس مزدوجتين  
 أي سلسلة مقتبسة هي ثابتة من النوع String "يعيش طويلاً ويزدهر."  
 يمكن إعطاء متغير من النوع String القيمة من كائن سلسلة نعمة السلسلة = "عش طويلاً وازدهر.";

## سلسلة السلسلة

التسلسل: استخدام العامل + على سلسلتين لربطهما لتكوين سلسلة واحدة أطول

إذا كانت التحية **تساوي** "مرحبا"

يساوي **javaClass** ، ثم التحية

"Hello class" يساوي **javaClass**

يمكن ربط أي عدد من السلسلة معاً

عندما يتم دمج سلسلة مع أي نوع آخر من العناصر تقريرياً، تكون النتيجة سلسلة

الجواب هو

+ تقييم 42

الجواب هو 42

## الفئات والأشياء والأساليب

الفئة هي اسم النوع الذي تكون قيمة  
أشياء

الكائنات هي كيانات تقوم ب تخزين البيانات و تأخذها  
إجراءات

تقوم كائنات فئة السلسلة ب تخزين البيانات التي تتكون من سلاسل من الأحرف

تسمى الإجراءات التي يمكن للكائن أن يتخذها بالطرق

يمكن للطرق إرجاع قيمة من نوع واحد و/أو  
تنفيذ إجراء

جميع الكائنات داخل الفصل لها نفس الأساليب، ولكن يمكن أن يكون لكل منها قيم بيانات  
مختلفة

## الفئات والأشياء والأساليب \_

استدعاء أو استدعاء طريقة: يتم استدعاء الطريقة للتنفيذ عن طريق كتابة اسم الكائن المستدعي، متبعًا بنقطة، متبوعة باسم الطريقة، متبعًا بين قوسين

ObjName.MethodName();

يُشار إلى هذا أحياناً باسم إرسال رسالة إلى الكائن

تحتوي الأقواس على المعلومات (إن وجدت) التي تحتاجها الطريقة

تسمى هذه المعلومات وسيطة (أو وسيطات)

## طرق السلسلة

□ تحتوي فئة السلسلة على العديد من الطرق المفيدة لتطبيقات معالجة السلسلة

□ يتم استدعاء أسلوب السلسلة عن طريق كتابة سلسلة كائن، ونقطة، واسم الطريقة، وزوج من الأقواس لإحاطة أي وسيطات

□ إذا قامت طريقة `String بإرجاع قيمة`، فيمكن وضعها في أي مكان حيث يمكن استخدام قيمة من نوعها سلسلة تحيية = "مرحبا" :

```
int count = Greeting.length();
Greeting.length(); الطول هو"nlthirp.tuo.metsyS
```

□ العد دائمًا من الصفر عند الإشارة إلى موضع أو فهرس حرف في سلسلة

## بعض الطرق في سلسلة الفصل

# (الجزء 1 من 8)

### Display 1.4 Some Methods in the Class String

#### `int length()`

Returns the length of the calling object (which is a string) as a value of type int.

#### **EXAMPLE**

After program executes `String greeting = "Hello!";`  
`greeting.length()` returns 6.

#### `boolean equals(Other_String)`

Returns true if the calling object string and the *Other\_String* are equal. Otherwise, returns false.

#### **EXAMPLE**

After program executes `String greeting = "Hello";`  
`greeting.equals("Hello")` returns `true`  
`greeting.equals("Good-Bye")` returns `false`  
`greeting.equals("hello")` returns `false`

Note that case matters. "Hello" and "hello" are not equal because one starts with an uppercase letter and the other starts with a lowercase letter.

# بعض الطرق في سلسلة الفصل

## (الجزء 2 من 8)

### Display 1.4 Some Methods in the Class String

`boolean equalsIgnoreCase(Other_String)`

Returns `true` if the calling object string and the `Other_String` are equal, considering uppercase and lowercase versions of a letter to be the same. Otherwise, returns `false`.

#### EXAMPLE

After program executes `String name = "mary!";`  
`greeting.equalsIgnoreCase("Mary!")` returns `true`

`String toLowerCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to lowercase.

#### EXAMPLE

After program executes `String greeting = "Hi Mary!";`  
`greeting.toLowerCase()` returns "hi mary!".

(continued)

## بعض الطرق في سلسلة الفصل

### (الجزء 3 من 8)

#### Display 1.4 Some Methods in the Class String

##### `String toUpperCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to uppercase.

##### **EXAMPLE**

After program executes `String greeting = "Hi Mary!";`  
`greeting.toUpperCase()` returns "HI MARY!".

##### `String trim()`

Returns a string with the same characters as the calling object string, but with leading and trailing white space removed. Whitespace characters are the characters that print as white space on paper, such as the blank (space) character, the tab character, and the new-line character '\n'.

##### **EXAMPLE**

After program executes `String pause = " Hmm ";`  
`pause.trim()` returns "Hmm".

(continued)

## بعض الطرق في سلسلة الفصل

### (الجزء 4 من 8)

#### Display 1.4 Some Methods in the Class String

`char charAt(Position)`

Returns the character in the calling object string at the *Position*. Positions are counted 0, 1, 2, etc.

#### **EXAMPLE**

After program executes `String greeting = "Hello!";`  
`greeting.charAt(0)` returns 'H', and  
`greeting.charAt(1)` returns 'e'.

`String substring(Start)`

Returns the substring of the calling object string starting from *Start* through to the end of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position *Start* is included in the value returned.

#### **EXAMPLE**

After program executes `String sample = "AbcdefG";`  
`sample.substring(2)` returns "cdefG".

(continued)

## بعض الطرق في سلسلة الفصل

# (الجزء 5 من 8)

### Display 1.4 Some Methods in the Class String

#### `String substring(Start, End)`

Returns the substring of the calling object string starting from position *Start* through, but not including, position *End* of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position *Start* is included in the value returned, but the character at position *End* is not included.

#### **EXAMPLE**

After program executes `String sample = "AbcdefG";`  
`sample.substring(2, 5)` returns "cde".

#### `int indexOf(A_String)`

Returns the index (position) of the first occurrence of the string *A\_String* in the calling object string. Positions are counted 0, 1, 2, etc. Returns -1 if *A\_String* is not found.

#### **EXAMPLE**

After program executes `String greeting = "Hi Mary!";`  
`greeting.indexOf("Mary")` returns 3, and  
`greeting.indexOf("Sally")` returns -1.

(continued)

## بعض الطرق في سلسلة الفصل

# الجزء 6 من (8)

### Display 1.4 Some Methods in the Class String

**int indexOf(A\_String, Start)**

Returns the index (position) of the first occurrence of the string *A\_String* in the calling object string that occurs at or after position *Start*. Positions are counted 0, 1, 2, etc. Returns -1 if *A\_String* is not found.

#### **EXAMPLE**

After program executes `String name = "Mary, Mary quite contrary";`

`name.indexOf("Mary", 1)` returns 6.

The same value is returned if 1 is replaced by any number up to and including 6.

`name.indexOf("Mary", 0)` returns 0.

`name.indexOf("Mary", 8)` returns -1.

**int lastIndexOf(A\_String)**

Returns the index (position) of the last occurrence of the string *A\_String* in the calling object string. Positions are counted 0, 1, 2, etc. Returns -1, if *A\_String* is not found.

#### **EXAMPLE**

After program executes `String name = "Mary, Mary, Mary quite so";`

`greeting.indexOf("Mary")` returns 0, and

`name.lastIndexOf("Mary")` returns 12.

## بعض الطرق في سلسلة الفصل

### (الجزء 7 من 8)

#### Display 1.4 Some Methods in the Class String

```
int compareTo(A_String)
```

Compares the calling object string and the string argument to see which comes first in the lexicographic ordering. Lexicographic order is the same as alphabetical order but with the characters ordered as in Appendix 3. Note that in Appendix 3 all the uppercase letters are in regular alphabetical order and all the lowercase letters are in alphabetical order, but all the uppercase letters precede all the lowercase letters. So, lexicographic ordering is the same as alphabetical ordering provided both strings are either all uppercase letters or both strings are all lowercase letters. If the calling string is first, it returns a negative value. If the two strings are equal, it returns zero. If the argument is first, it returns a positive number.

#### EXAMPLE

After program executes `String entry = "adventure";`  
`entry.compareTo("zoo")` returns a negative number,  
`entry.compareTo("adventure")` returns 0, and  
`entry.compareTo("above")` returns a positive number.

(continued)

## بعض الطرق في سلسلة الفصل

### (الجزء 8 من 8)

#### Display 1.4 Some Methods in the Class String

```
int compareToIgnoreCase(A_String)
```

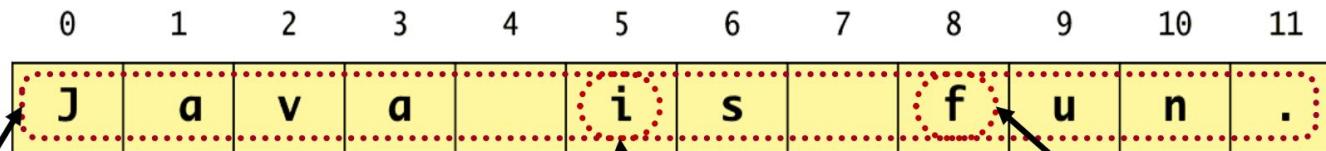
Compares the calling object string and the string argument to see which comes first in the lexicographic ordering, treating uppercase and lowercase letters as being the same. (To be precise, all uppercase letters are treated as if they were their lowercase versions in doing the comparison.) Thus, if both strings consist entirely of letters, the comparison is for ordinary alphabetical order. If the calling string is first, it returns a negative value. If the two strings are equal ignoring case, it returns zero. If the argument is first, it returns a positive number.

#### EXAMPLE

After program executes `String entry = "adventure";`  
`entry.compareToIgnoreCase("Zoo")` returns a negative number,  
`entry.compareToIgnoreCase("Adventure")` returns 0, and  
`"Zoo".compareToIgnoreCase(entry)` returns a positive number.

## فهارس السلسلة

```
String text = "Java";
```



نص

يشير هذا المتغير إلى السلسلة بأكملها.

ترجع الطريقة موضع السلسلة.

text.charAt(8)  
 يقوم الطريقة بإرجاع الحرف في  
الموضع رقم 8.

## تسلسل الهروب

- شرطة مائلة عكسية (\) تسبق مباشرةً أ<sup>ي</sup>شير الحرف (أي بدون أي مسافة) إلى تسلسل هروب أو حرف هروب
- لا يحتوي الحرف الذي يلي الشرطة المائلة العكسية معناها المعتاد
- وعلى الرغم من أنها تكون من رمzin إلا أنها تعتبر حرفًا واحدًا

### Display 1.6 Escape Sequences

```
\\" Double quote.  
\' Single quote.  
\\\ Backslash.  
\n New line. Go to the beginning of the next line.  
\r Carriage return. Go to the beginning of the current line.  
\t Tab. White space up to the next tab stop.
```

## أمثلة

اختبار السلسلة = "مرحبا سعد"; اختبار

```
test.toUpperCase(); System.out.println(test);
```

=

System.out.println("abc\ndef");

System.out.println("abc\\ndef");

## تمارين

لكل من هذه العبارات تحديد خاصته  
نتيجة

نص السلسلة = "برمجة جافا" :

أ. `text.substring(0, 4)`

ب. طول النص ()

ج. `text.substring(8, 12)`

د. `text.substring(0, 1) + text.substring(7, 9)`

هـ. `6) + text.substring(text.length() - 3, text.length())`

`text.substring(5,`

## تعليقات

□ يبدأ تعليق السطر بالرموز //، ويؤدي إلى تجاهل المترجم لبقية السطر

□ يستخدم هذا النوع من التعليق لكاتب الكود أو للمبرمج الذي يقوم بتعديل الكود

□ يبدأ تعليق الكتلة بزوج الرموز / ، \* وينتهي بزوج الرموز \*

□ يتجاهل المترجم أي شيء بينهما

□ هذا النوع من التعليق يمكن أن يمتد لعدة أسطر

□ يوفر هذا النوع من التعليقات وثائق لمستخدمي البرنامج

# تعليقات

Display 1.8 Comments and a Named Constant

```

1  /**
2  Program to show interest on a sample account balance.
3  Author: Jane Q. Programmer.
4  E-mail Address: janeq@somemachine.etc.etc.
5  Last Changed: September 21, 2004.
6 */
7 public class ShowInterest
8 {
9     public static final double INTEREST_RATE = 2.5;

10    public static void main(String[] args)
11    {
12        double balance = 100;
13        double interest; //as a percent

14        interest = balance * (INTEREST_RATE/100.0);
15        System.out.println("On a balance of $" + balance);
16        System.out.println("you will earn interest of $"
17                           + interest);
18        System.out.println("All in just one short year.");
19    }
20 }  

21 }  



Although it would not be as clear, it is  
legal to place the definition of  
INTEREST_RATE here instead.


```

**SAMPLE DIALOGUE**

On a balance of \$100.0  
you will earn interest of \$2.5  
All in just one short year.

## فئة StringTokenizer \_

يتم استخدام فئة StringTokenizer لاستعادة الكلمات أو الرموز المميزة في سلسلة متعددة الكلمات

يمكنك استخدام أحرف المسافات البيضاء لفصل كل رمز مميز، أو يمكنك تحديد الأحرف التي ترغب في استخدامها كفواصل

من أجل استخدام فئة StringTokenizer ، تأكد من تضمين ما يلي في بداية الملف:

```
import java.util.StringTokenizer;
```

## بعض الطرق في

### فئة StringTokenizer (الجزء 1 من 2)

#### Display 4.17 Some Methods in the Class StringTokenizer

The class StringTokenizer is in the `java.util` package.

```
public StringTokenizer(String theString)
```

Constructor for a tokenizer that will use whitespace characters as separators when finding tokens in theString.

```
public StringTokenizer(String theString, String delimiters)
```

Constructor for a tokenizer that will use the characters in the string delimiters as separators when finding tokens in theString.

```
public boolean hasMoreTokens()
```

Tests whether there are more tokens available from this tokenizer's string. When used in conjunction with `nextToken`, it returns `true` as long as `nextToken` has not yet returned all the tokens in the string; returns `false` otherwise.

(continued)

## بعض الطرق في

### فئة StringTokenizer (الجزء 2 من 2)

#### Display 4.17 Some Methods in the Class StringTokenizer

```
public String nextToken()
```

Returns the next token from this tokenizer's string. (Throws NoSuchElementException if there are no more tokens to return.)<sup>5</sup>

```
public String nextToken(String delimiters)
```

First changes the delimiter characters to those in the string `delimiters`. Then returns the next token from this tokenizer's string. After the invocation is completed, the delimiter characters are those in the string `delimiters`.

(Throws NoSuchElementException if there are no more tokens to return. Throws NullPointerException if `delimiters` is null.)<sup>5</sup>

```
public int countTokens()
```

Returns the number of tokens remaining to be returned by `nextToken`.