

Variables, Objects & Classes

CSCU9A3
Data Structures, Objects and Algorithms

David Cairns

Derived from *Big Java* by Cay Horstmann
John Wiley & Sons

Declaring a Variable

Syntax `typeName variableName = value;`
or
`typeName variableName;`

Example

The type specifies what can be done with values stored in this variable.

String greeting = "Hello, Dave!";

See the rules for and table of examples of valid names.

A variable declaration ends with a semicolon.

Use a descriptive variable name.

Supplying an initial value is optional, but it is usually a good idea.

Assignment

- The assignment operator is `=`
- When you see the `=` symbol, e.g. `x = y + 1`, it may be safer to say "x takes the value of y + 1"

- It is used to change the value of a variable:

```
int width = 10;

width = 20;

width = width + 1;

width++;
```

Objects & Classes

- Object: entity that you can manipulate in your programs by calling methods
 - Each object is formed from a Class
 - The Class definition acts as a template indicating how objects made from that Class will behave.
 - The definition consists of a set of attributes and then some methods available to manipulate those attributes.
 - Usually you change attributes via method calls.
 - The attributes are normally hidden away inside the class (they are private)
 - When you write your own classes, you know the attributes are there but you should aim to make them private.
- There are many pre-built classes available in the core Java libraries
- You can easily make your own - they are no different in status to the library classes that come with Java.

A Car Class

```
public class Car
{
    private String make;
    private String model;
    private String colour;
    private int enginecc;

    public Car(String mk, String mod, String col, int cc)
    {
        make = mk;
        model = mod;
        colour = col;
        enginecc = cc;
    }

    public String description()
    {
        String desc = make + " " + model + ", " + colour;
        return desc;
    }

    public void changeColour(String col)
    {
        colour = col;
    }
}
```

CSCU9A3 - Autumn 2017

5

Example: Using the Car Class

```
public class SportsCars
{
    public static void main(String[] args)
    {
        SportsCars sc = new SportsCars();
        sc.go();
    }

    public void go()
    {
        Car c = new Car("Ferrari", "Enzo", "Red", 6000);
        c.changeColour("Blue");
        System.out.println("Current Car: " + c.description());
    }
}
```

CSCU9A3 - Autumn 2017

6

Methods

- **Method:** sequence of instructions that accesses the data of an object
 - You manipulate objects by calling its methods
- **Class:** declares the methods that you can apply to its objects

```
String greeting = "Hello";  
greeting.println(); // Error...  
greeting.length(); // OK
```

- **Public methods:** specify what you can do with the objects of a class
 - what you as a user can do with it / what services it offers
 - Classes often have private methods that do internal work
 - As a user of a class, you don't need to know or worry about them.
 - It can be useful to write private methods for your own classes

Overloaded Methods

- A class can have a number of methods with the same name, but different parameters
- Example: the `PrintStream` class has a set of `println` methods, one for each different type of thing you might want to print:

- `public void println(int output)`
- `public void println(double output)`
- `public void println(String output)`

Parameters

- **Parameter:** an input to a method

- `PrintStream println`

- Definition

```
public void println(String g)
```

- Usage

```
System.out.println(greeting)
```

Not all methods have parameters...

```
greeting.length()
```

Return Values

- **Return value:** a result that the method has computed for use by the code that called it:

```
// Find out the number of characters in greeting
// and store the result in variable n
int n = greeting.length();
```

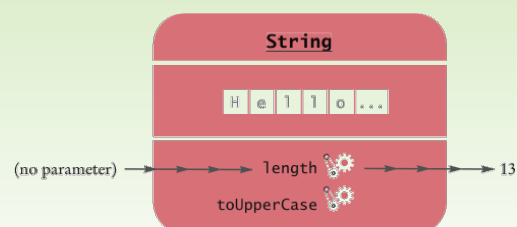


Figure 7 Invoking the length Method on a String Object

Passing on Return Values

- You can also use the return value as a parameter to another method call:

```
System.out.println(greeting.length());
```

- This is equivalent to:

```
int n = greeting.length();
System.out.println(n);
```

- It saves on typing in code and declaring temporary variables (in this case 'n') but can be harder to follow.
- You start at the inner most brackets and work your way out

Java API Guide

java.awt.im.spi java.awt.image java.awt.image.Renderable java.awt.print java.beans java.beans.beancontext java.io java.lang java.lang.annotation java.lang.instrument java.lang.management java.lang.ref java.lang.reflect java.math java.net	<p>Allocates a new <i>String</i> that contains characters from a subarray of the Unicode code point array argument.</p> <p><i>String</i>(<i>String</i> original) Initializes a newly created <i>String</i> object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.</p> <p><i>String</i>(<i>StringBuffer</i> buffer) Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.</p> <p><i>String</i>(<i>StringBuilder</i> builder) Allocates a new string that contains the sequence of characters currently contained in the string builder argument.</p>
Classes Boolean Byte Character Character.Subset Character.UnicodeBlock Class ClassLoader Compiler Double Enum File InheritableThreadLocal Integer Long Math Number Object Package Process ProcessBuilder Runtime RuntimePermission SecurityManager Short StackTraceElement StrictMath String StringBuffer StringBuilder System Thread ThreadGroup ThreadLocal Throwable	<p>Method Summary</p> <p>char charAt(int index) Returns the <i>char</i> value at the specified index.</p> <p>int codePointAt(int index) Returns the character (Unicode code point) at the specified index.</p> <p>int codePointBefore(int index) Returns the character (Unicode code point) before the specified index.</p> <p>int codePointCount(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this <i>String</i>.</p> <p>int compareTo(<i>String</i> anotherString) Compares two strings lexicographically.</p> <p>int compareToIgnoreCase(<i>String</i> str) Compares two strings lexicographically, ignoring case differences.</p> <p><i>String</i> concat(<i>String</i> str) Concatenates the specified string to the end of this string.</p> <p>boolean contains(<i>CharSequence</i> s) Returns true if and only if this string contains the specified sequence of char values.</p> <p>boolean contentEquals(<i>CharSequence</i> cs) Compares this string to the specified <i>CharSequence</i>.</p> <p>boolean contentEquals(<i>StringBuffer</i> sb) Compares this string to the specified <i>StringBuffer</i>.</p> <p>static <i>String</i> copyValueOf(char[] data) Returns a <i>String</i> that represents the character sequence in the array specified.</p>