Variables, Objects & Classes

CSCU9A3
Data Structures, Objects and Algorithms

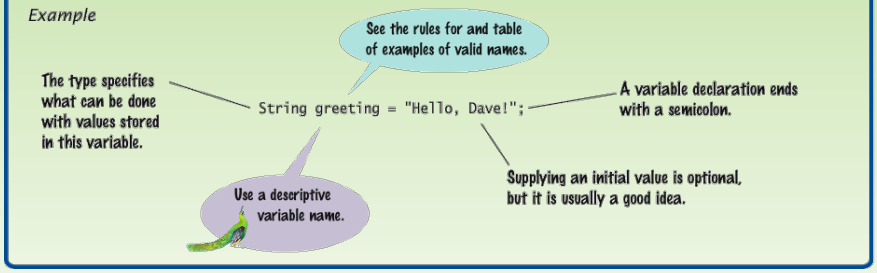
David Cairns

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

Declaring a Variable

Syntax *typeName variableName = value;*
or
typeName variableName;

Example



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.
      - Ususally 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;
	}
}
```


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());
    }
}
```

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();
```

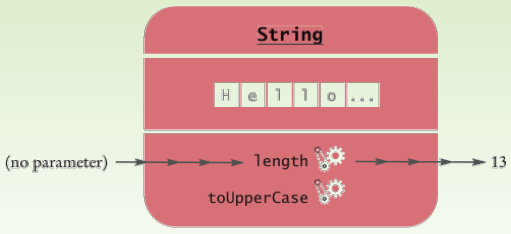


Figure 7 Invoking the `length` Method on a `String` Object

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