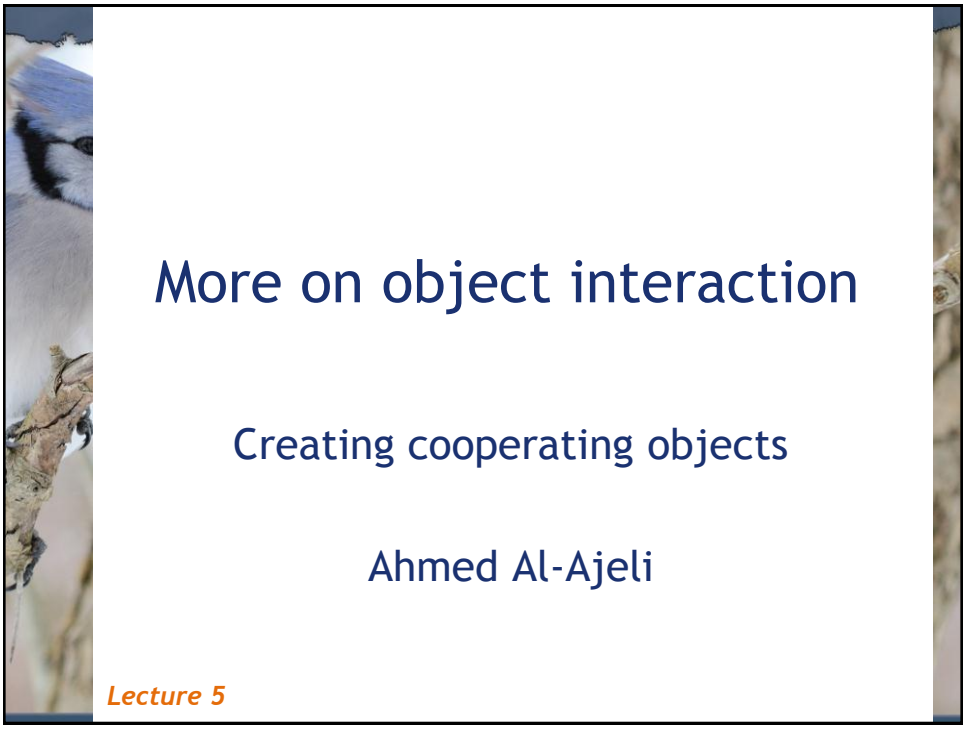
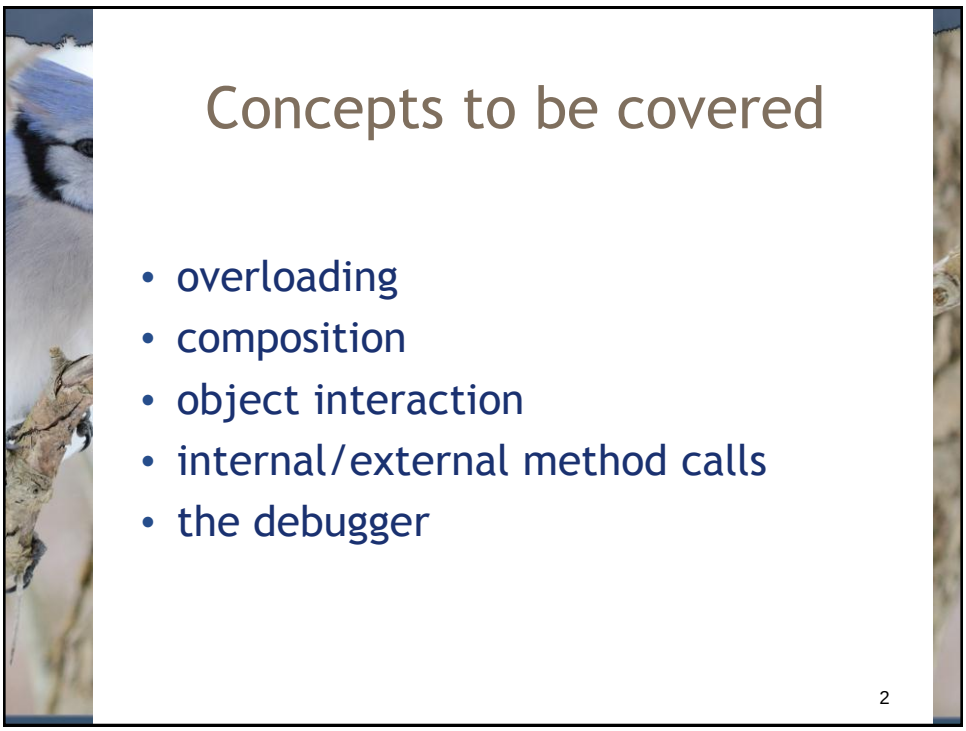


More on object interaction

Creating cooperating objects

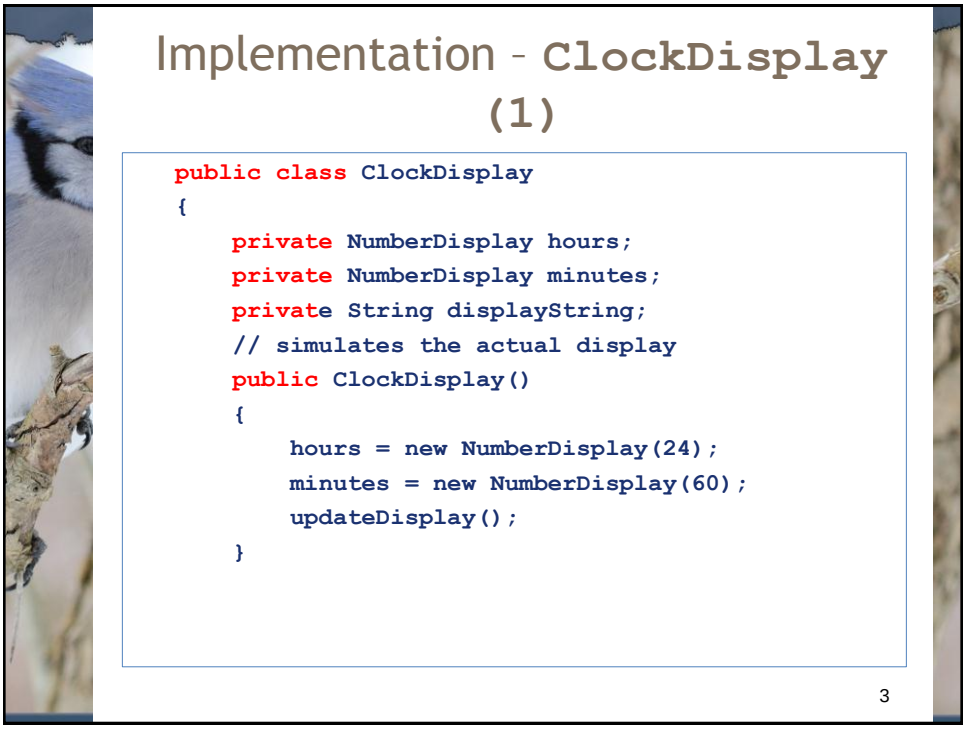
Ahmed Al-Ajeli

Lecture 5



Concepts to be covered

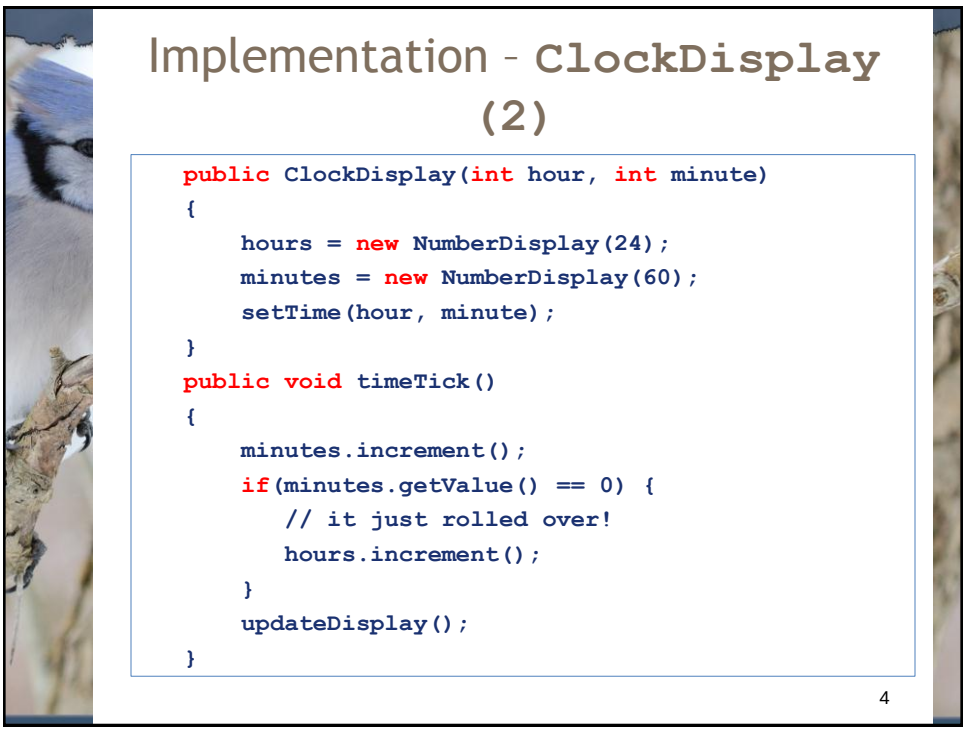
- overloading
- composition
- object interaction
- internal/external method calls
- the debugger



Implementation - ClockDisplay (1)

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;
    // simulates the actual display
    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        updateDisplay();
    }
```

3



Implementation - ClockDisplay (2)

```
public ClockDisplay(int hour, int minute)
{
    hours = new NumberDisplay(24);
    minutes = new NumberDisplay(60);
    setTime(hour, minute);
}
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) {
       // it just rolled over!
       hours.increment();
    }
    updateDisplay();
}
```

4

Implementation - `ClockDisplay` (3)

```
public void setTime(int hour, int minute)
{
    hours.setValue(hour);
    minutes.setValue(minute);
    updateDisplay();
}
public String getTime()
{
    return displayString;
}
private void updateDisplay()
{
    displayString = hours.getDisplayValue()
                    + ":" +
                  minutes.getDisplayValue();
}
}
```

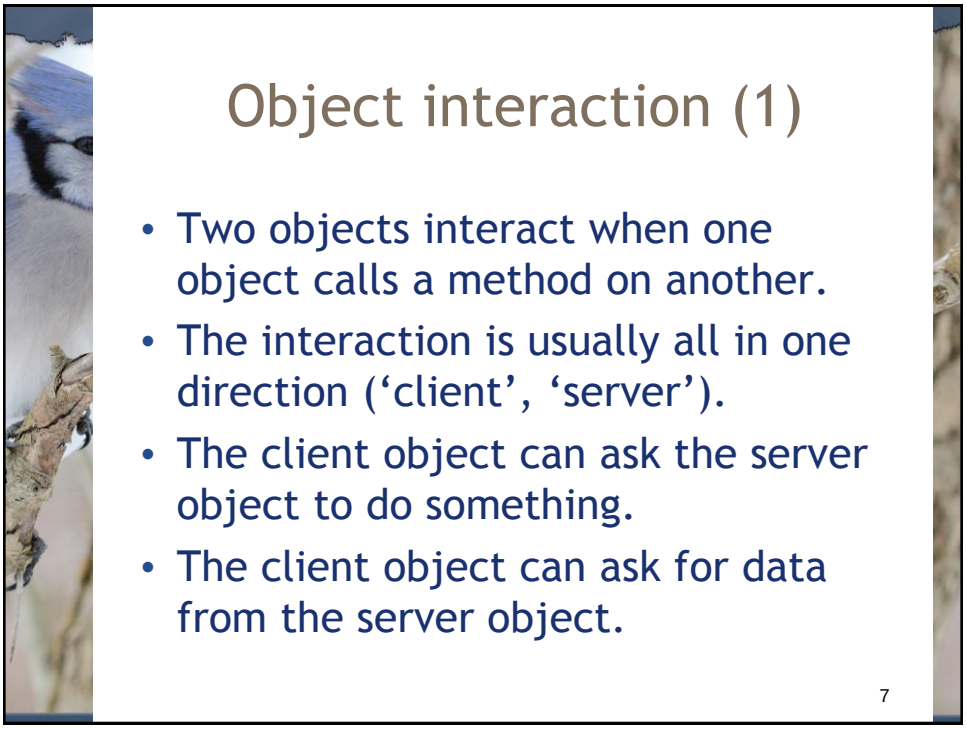
5

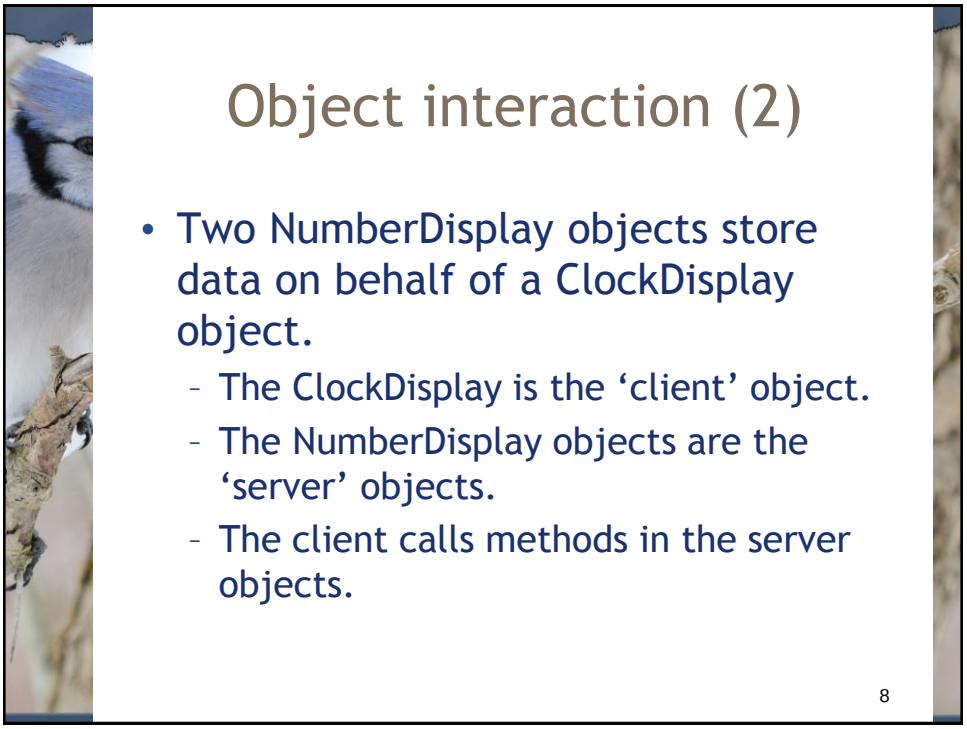
Overloading

- A class may contain more than one constructor, or more than one method of the same name, as long as each has a distinctive set of parameters.
- Example: `ClockDisplay` class has two constructors:

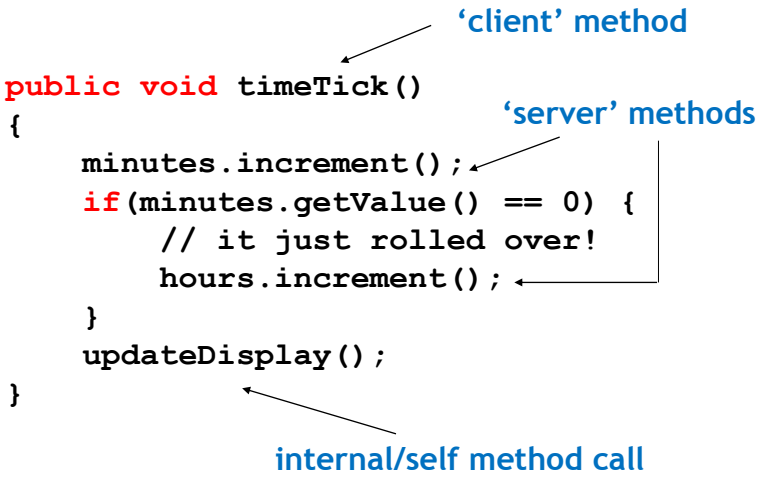
```
ClockDisplay()
ClockDisplay(int hour, int minute)
```

6


Object interaction (1)
• Two objects interact when one object calls a method on another.
• The interaction is usually all in one direction (‘client’, ‘server’).
• The client object can ask the server object to do something.
• The client object can ask for data from the server object.
7


Object interaction (2)
• Two NumberDisplay objects store data on behalf of a ClockDisplay object.
– The ClockDisplay is the ‘client’ object.
– The NumberDisplay objects are the ‘server’ objects.
– The client calls methods in the server objects.
8

Method calling



9

External method calls

- General form:

Dot notation

object . *methodName* (*params*)

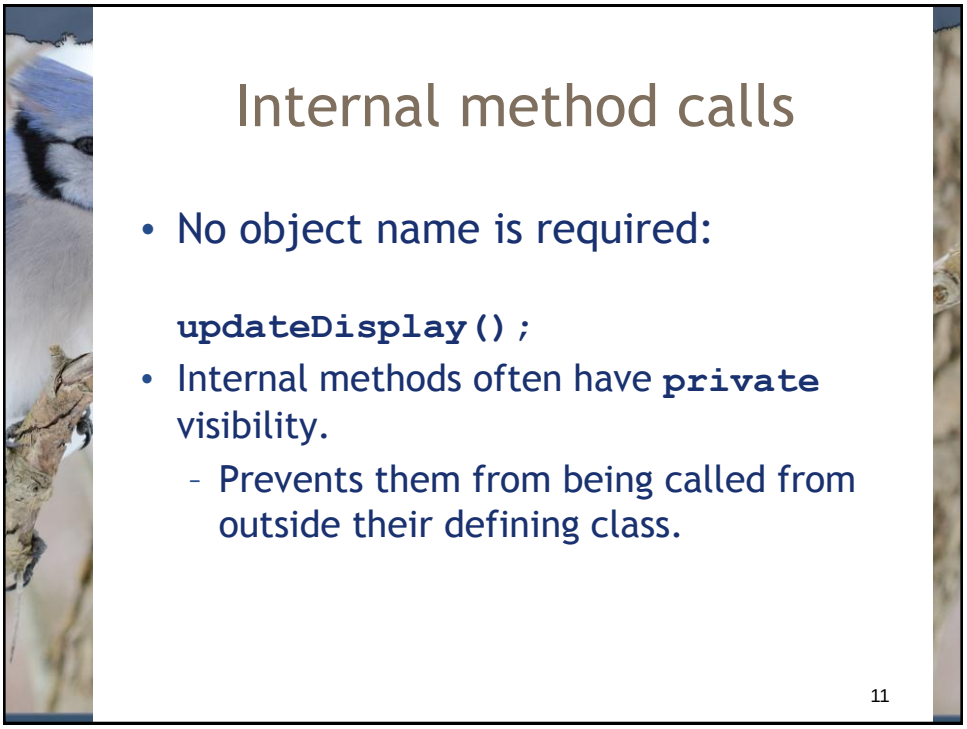
NB: object, **not** class name

- Examples:

```
hours.increment()

minutes.getValue()
```

10



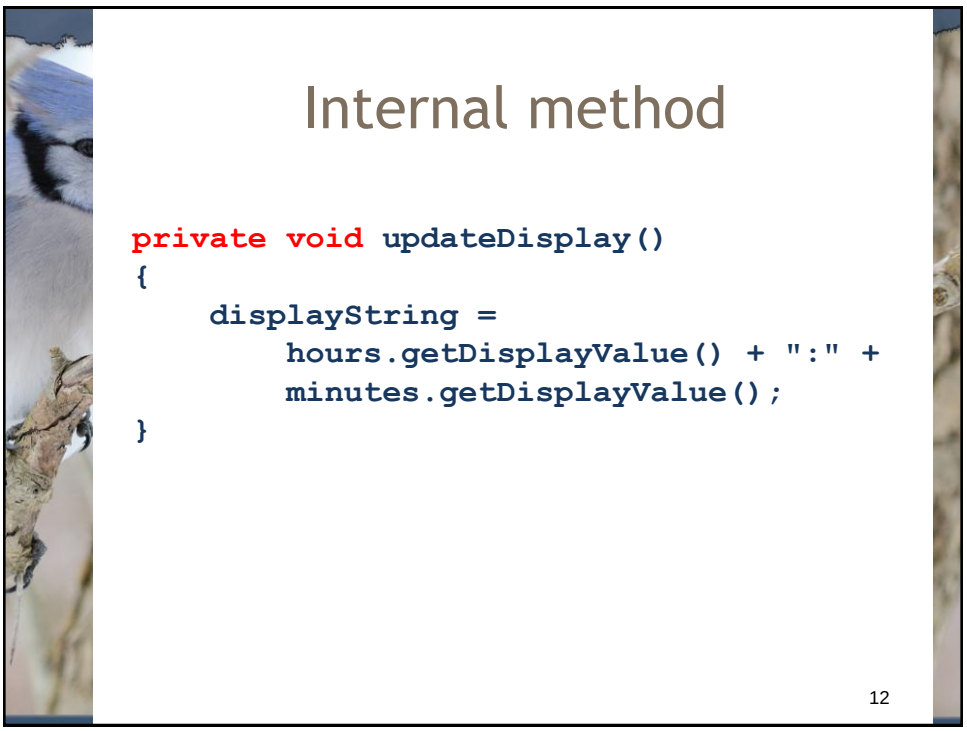
Internal method calls

- No object name is required:

```
updateDisplay();
```

- Internal methods often have **`private`** visibility.
  - Prevents them from being called from outside their defining class.

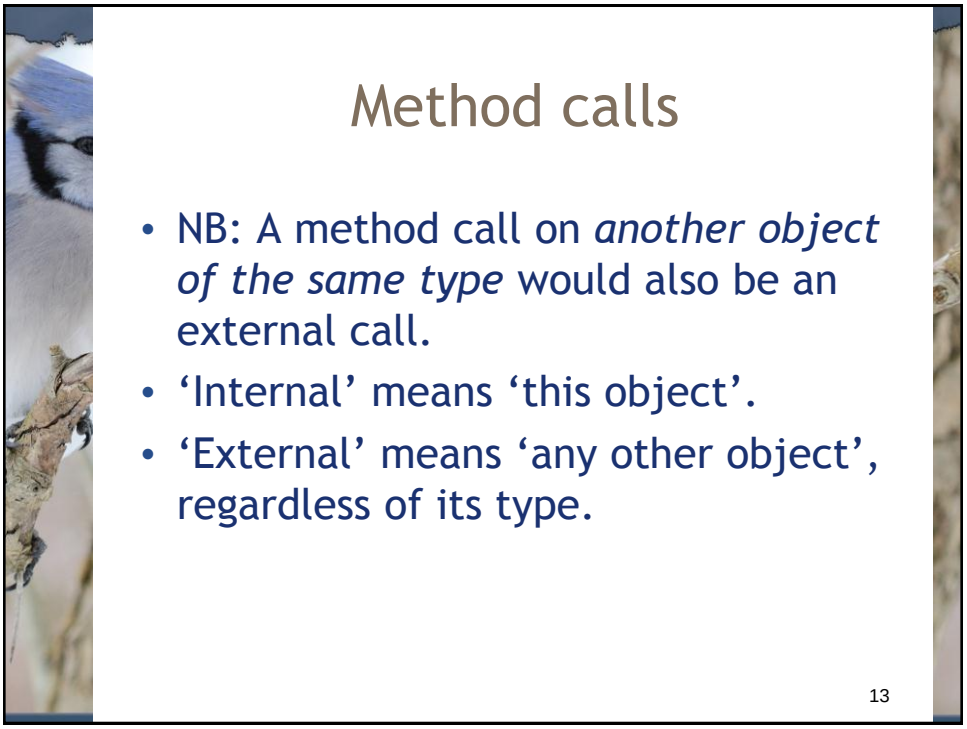
11



Internal method

```
private void updateDisplay()
{
    displayString =
        hours.getDisplayValue() + ":" +
        minutes.getDisplayValue();
}
```

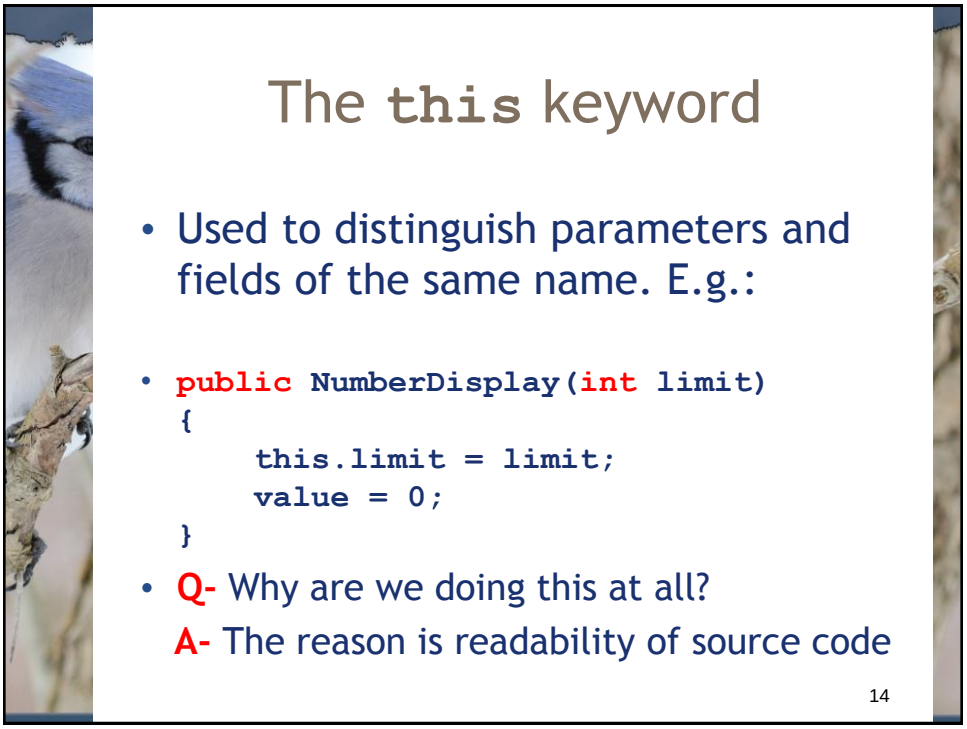
12



Method calls

- NB: A method call on *another object of the same type* would also be an external call.
- 'Internal' means 'this object'.
- 'External' means 'any other object', regardless of its type.

13



The `this` keyword

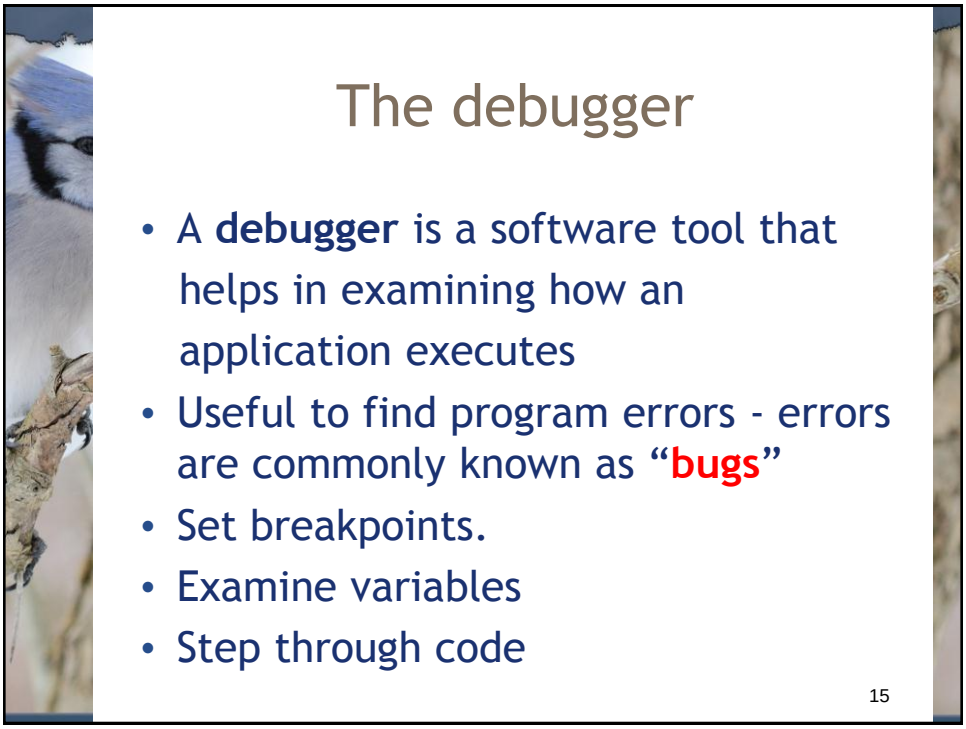
- Used to distinguish parameters and fields of the same name. E.g.:

```
public NumberDisplay(int limit)
{
    this.limit = limit;
    value = 0;
}
```

- **Q-** Why are we doing this at all?

  A- The reason is readability of source code

14


The debugger
• A debugger is a software tool that helps in examining how an application executes
• Useful to find program errors - errors are commonly known as "bugs"
• Set breakpoints.
• Examine variables
• Step through code
15