Object Interaction 2

Creating cooperating objects

Module assessment

Four components:

- 3 assignments (“coursework”)
- 1 programming test

Assignment weights:

- Assignment 1: 20%
- Assignment 2: 40%
- Assignment 3: 40%

If programming test is passed:

final mark = coursework mark

If programming test is failed:

final mark = (coursework mark * 0.37)

Assignment submission: drop folder

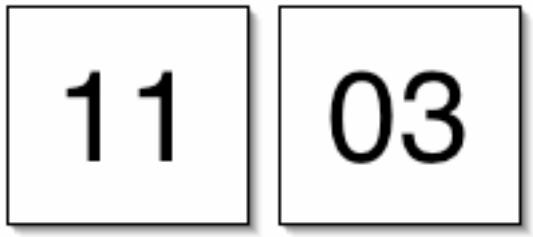
S:\proj\co320_ca13\assignment-1\<*login*>

Modularising the clock display

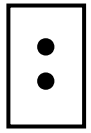
Reminder

One four-digit display?

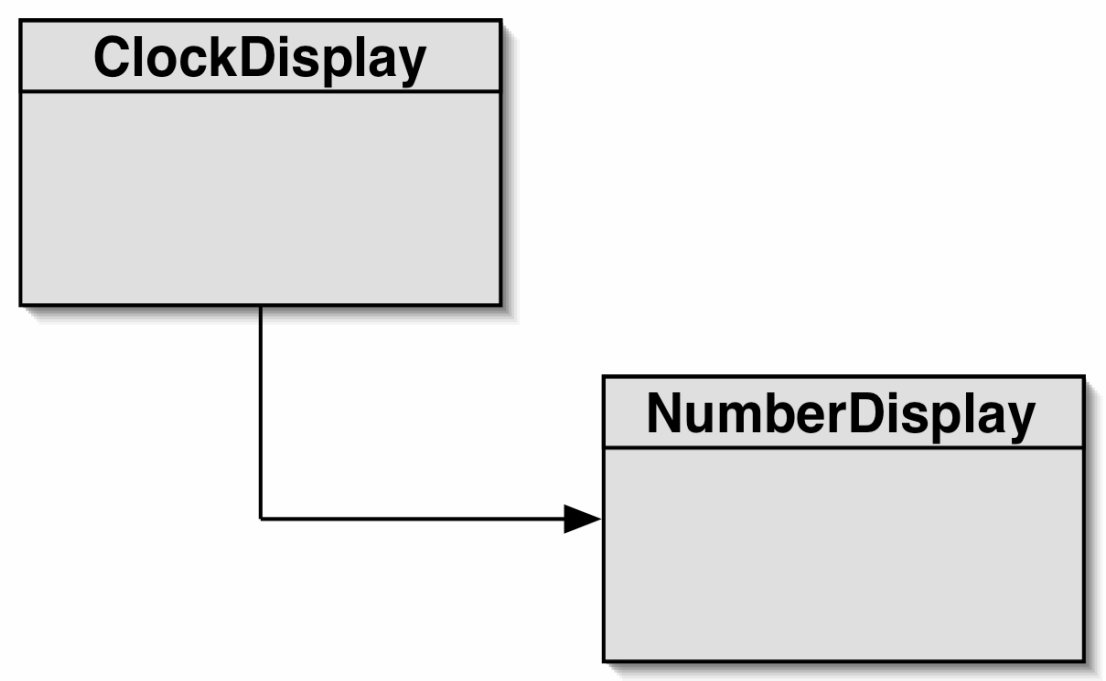
Or two two-digit displays?



And a bit of glue …

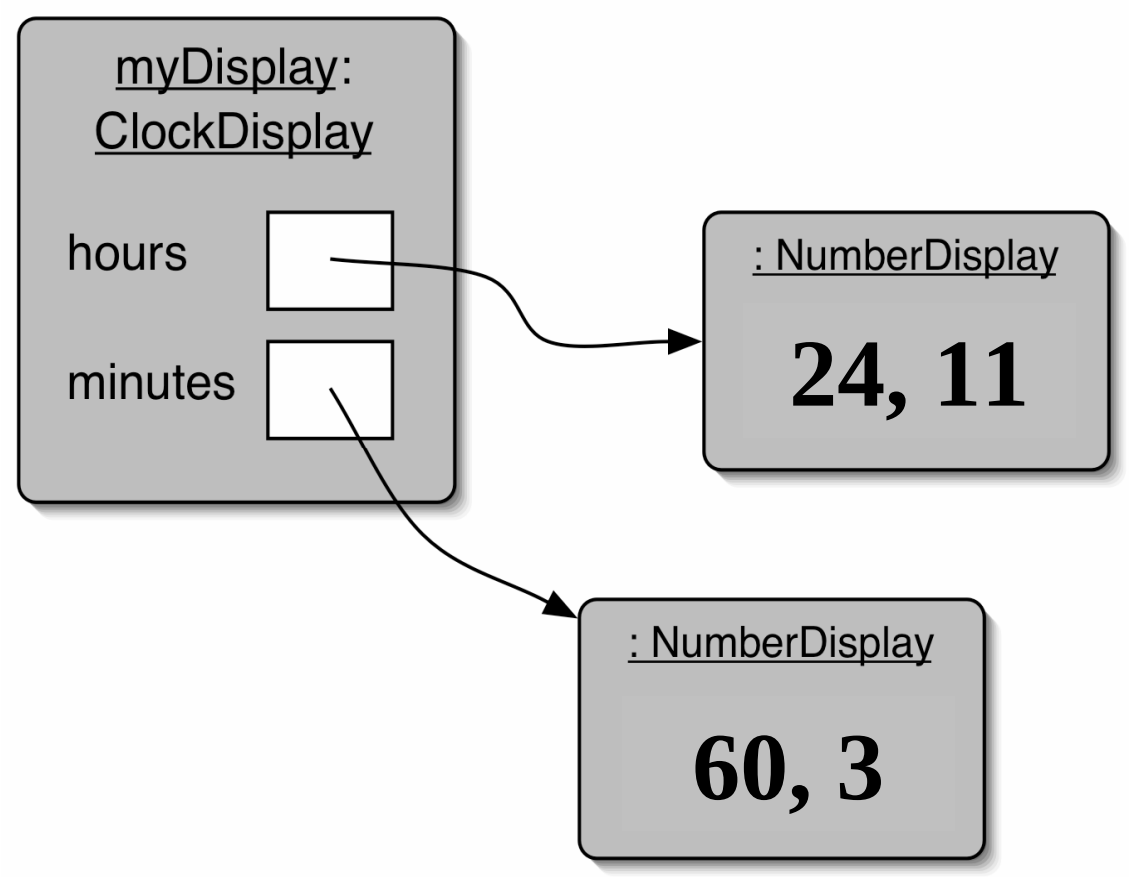


Class Diagram



This means that class **`ClockDisplay`** contains a *reference* (either through a *field*, method *parameter* or method *local variable*) to an object that is a **`NumberDisplay`**.

Object Diagram

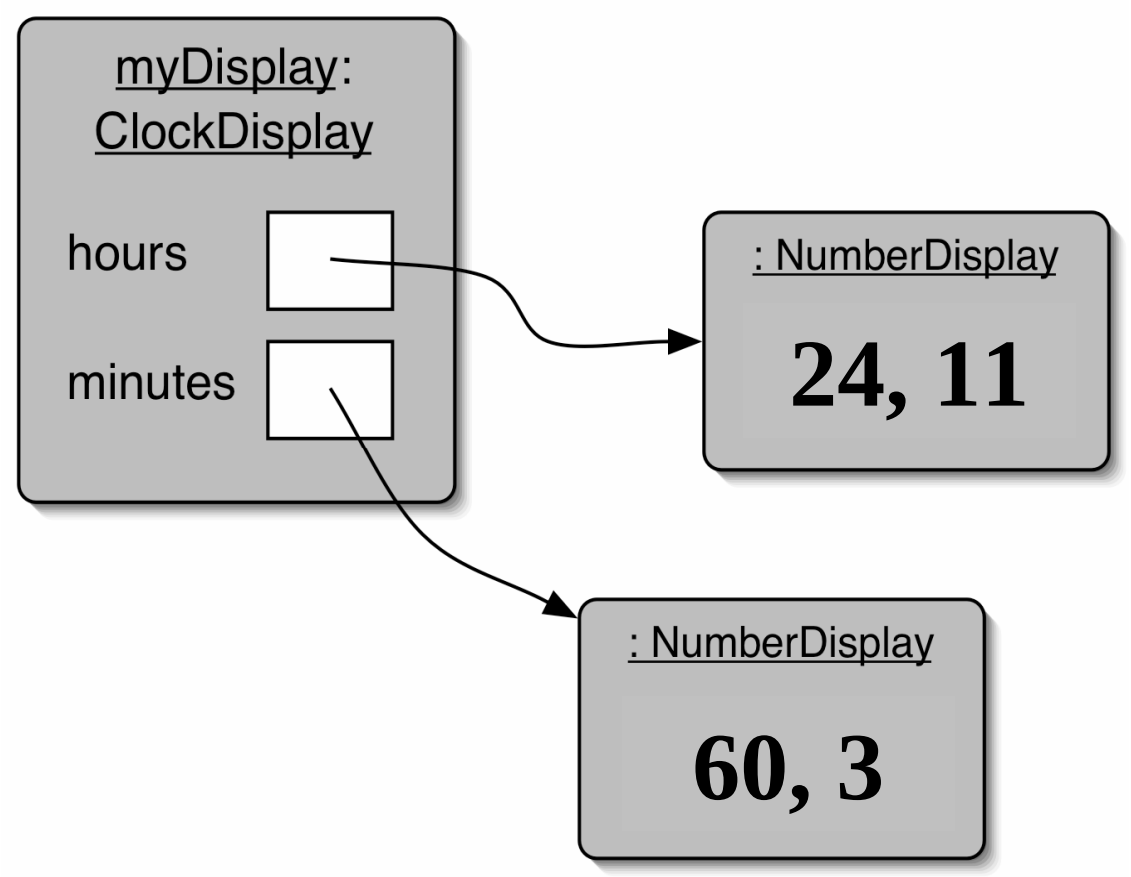


`ClockDisplay` actually contains two *references* (each is a *field*) to a **`NumberDisplay`**.

Objects containing Numbers

```
public class NumberDisplay
{
    private int limit;
    private int value;

    public NumberDisplay (int rollOver)
    {
        limit = rollOver;
        value = 0;          -- start at 0
    }

    ...  methods omitted
}
```

Object Diagram



`ClockDisplay` actually contains two *references* (each is a *field*) to a **`NumberDisplay`**.

Objects referencing Objects

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    ...  other fields

    public ClockDisplay()
    {
        ...  initialise all the fields
    }

    ...  more stuff
}
```

Objects referencing Objects

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    ...  other fields

    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        ...  initialise the other fields
    }
```

Objects referencing Objects

A constructor in class NumberDisplay:

```
public NumberDisplay (int rollOver)
{
    ...
}
```

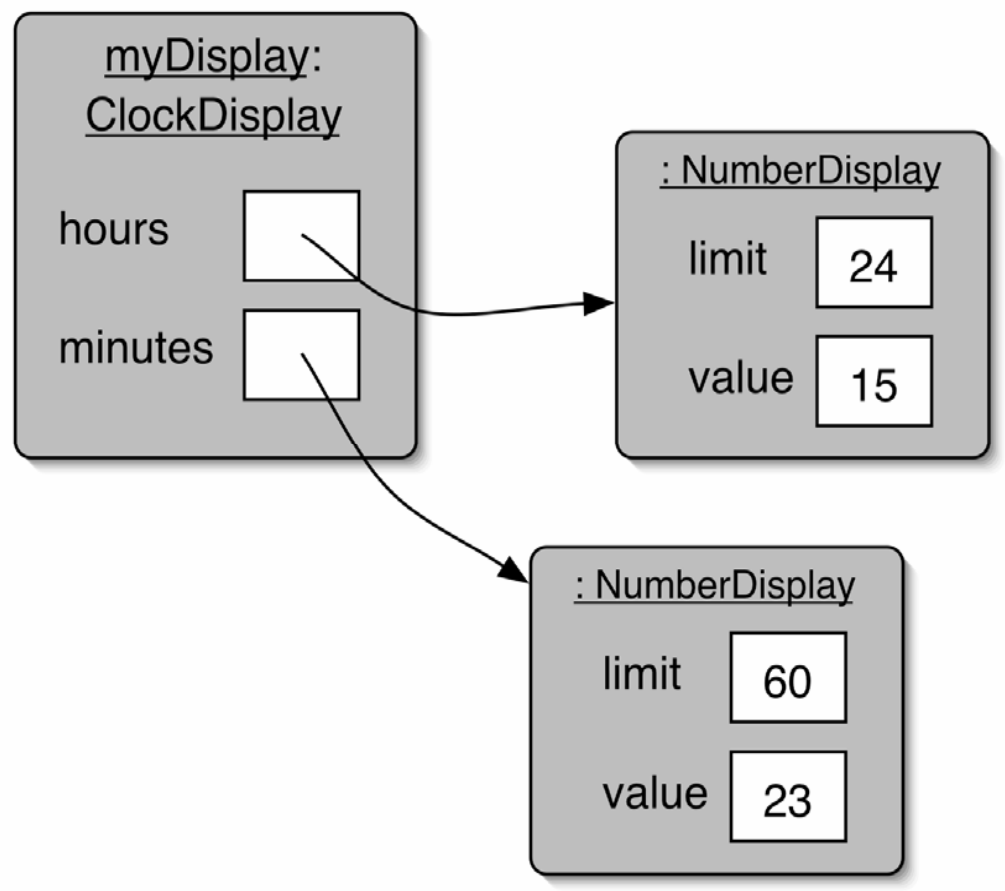
formal parameter

Code in class ClockDisplay:

```
hours = new NumberDisplay (24);
```

supplied argument

ClockDisplay Object Diagram



Objects referencing Objects

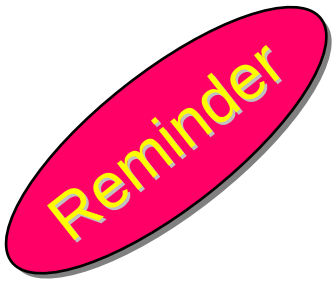
```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;

    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        updateDisplay();
    }
```

Method calling (example from ClockDisplay)

```
/**
 * Update the internal field that
 * represents the display.
 */
private void updateDisplay()
{
    displayString =
        hours.getDisplayValue() + ":" +
        minutes.getDisplayValue();
}
```

Source code: NumberDisplay

```
public String getDisplayValue()
{
    if(value < 10) {
        return "0" + value;
    }
    else {
        return "" + value;
    }
}
```



Method calling (example from ClockDisplay)

```
/**
 * Advance the time by one minute.
 */
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) {
        // it just rolled over!
        hours.increment();
    }
    updateDisplay();
}
```

Method Calls

Calling a method on *another* object:

```
objectName.methodName (argument, list)
```

Calling a method on *our own* object:

```
methodName (argument, list)
```

… or:

```
this.methodName (argument, list)
```

Method Call *Examples*

Calling a method on *another* object:

```
minutes.increment();
```

where:

```
public void increment() {...}
```

must be a *public* method in the **`class`** to which the **`minutes`** *object* belongs.

Note: if **`increment()`** had been a **`private`** method, we would not have been able to invoke it!

Method Call Examples

Calling a method on *our own* object:

```
updateDisplay();
```

where:

```
private void updateDisplay() {...}
```

must be a method in the same **`class`** as this code.

Note: we can invoke both **`public`** and **`private`** methods from our own class.

optionally

Method Call Examples

Calling a method on *our own* object:

```
this.updateDisplay();
```

where:

```
private void updateDisplay() {...}
```

must be a method in the same ***class*** as this code.

The name of the object on which class code executes is always **this** (from the perspective of that class code).

The `null` Object

`null` is a special `Object` in Java.

All `Object` variables (of any `class`) are initially `null`.

Variables can be tested to see if they are `null`.

```
private NumberDisplay hours;

public void makeHoursVisible() {
  if (hours == null) {
      ...  nothing to show
  } else {
      ...  display it
  }
}
```

The `null` Object

`null` is a special `Object` in Java.

All `Object` variables (of any `class`) are initially `null`.

Variables can be tested to see if they are `null`.

Variables can also be assigned to `null` – losing the reference to anything they were previously holding.

```
private NumberDisplay hours;

public void forgetHours() {
  hours = null;
}
```

Concepts

- modularity
- objects referencing other objects
- constructors and methods
- internal / external method calls
- the `null` object