Object interaction

Creating cooperating objects
Based in lecture notes by David J. Barnes, Michael Kölling

Abstraction and modularisation

- **Abstraction** is the ability to ignore details of parts to focus attention on a higher level of a problem.

- **Modularisation** is the process of dividing a whole into well-defined parts, which can be built and examined separately, and which interact in well-defined ways.

A digital clock

11:03

Modularising the clock display

One four-digit display?

11:03

Or two two-digit displays?

11 03
**Implementation - NumberDisplay**

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

    Constructor and methods omitted.
}
```

**Implementation - ClockDisplay**

```java
public class ClockDisplay {
    private NumberDisplay hours;
    private NumberDisplay minutes;

    Constructor and methods omitted.
}
```

**Object diagram**

**Class diagram**
Quiz: What is the output?

- int a;
  int b;
  a = 32;
  b = a;
  a = a + 1;
  System.out.println(b);

- Person a;
  Person b;
  a = new Person("Everett");
  b = a;
  a.changeName("Delmar");
  System.out.println(b.getName());

Source code: NumberDisplay

```java
public NumberDisplay(int rollOverLimit) {
    limit = rollOverLimit;
    value = 0;
}
```

```java
public void increment() {
    value = (value + 1) % limit;
}
```
The modulo operator

- The 'division' operator (/), when applied to int operands, returns the result of an integer division.
- The 'modulo' operator (%) returns the remainder of an integer division.
- E.g., generally:
  \[ 17 \div 5 = \text{result } 3, \text{ remainder } 2 \]
- In Java:
  \[ 17 \div 5 = 3 \]
  \[ 17 \% 5 = 2 \]

Quiz

- What is the result of the expression \((8 \% 3)\)?
- What are all possible results of the expression \((n \% 5)\)?

Source code: NumberDisplay

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

Concepts

- abstraction
- modularisation
- classes define types
- class diagram
- object diagram
- object references
- primitive types
- object types
Objects creating objects

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

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

ClockDisplay object diagram

null

- null is a special value in Java
- All object variables are initialised to null.
- You can assign and test for null:
  ```java
  private NumberDisplay hours;
  if (hours == null) { ... }
  hours = null;
  ```

Objects creating objects

in class NumberDisplay:
```java
public NumberDisplay(int rollOverLimit);
```

formal parameter

in class ClockDisplay:
```java
hours = new NumberDisplay(24);
```

actual parameter
public void timeTick()
{
    minutes.increment();
    if (minutes.getValue() == 0)
    {
        // it just rolled over!
        hours.increment();
    }
    updateDisplay();
}

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

• internal method calls
  updateDisplay();
  ...
  private void updateDisplay()

• external method calls
  minutes.increment();

object . methodName ( parameter-list )
Concepts

- object creation
- overloading
- internal/external method call
- debugger