Week 3

Object Interaction 1

Creating cooperating objects

suggested reading:

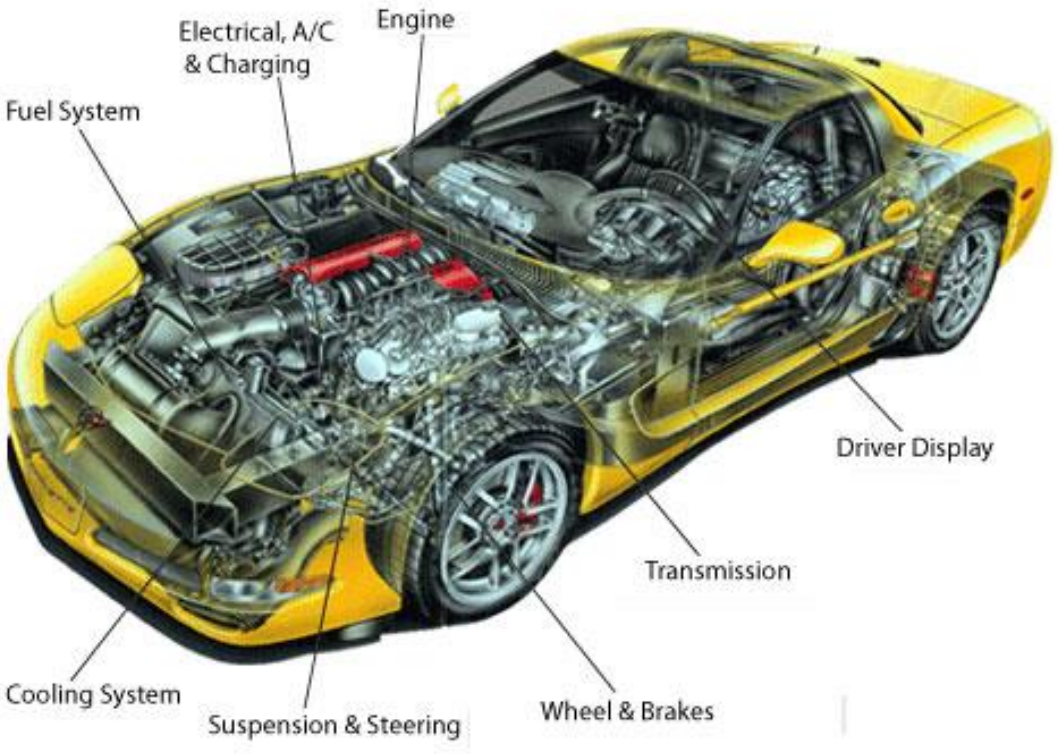
Textbook, Ch. 3

A digital clock

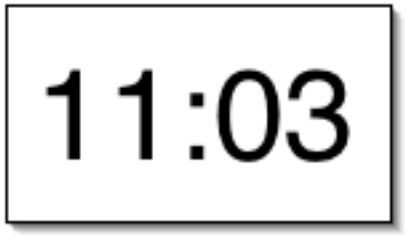
Abstraction and modularization

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

Abstraction and modularization

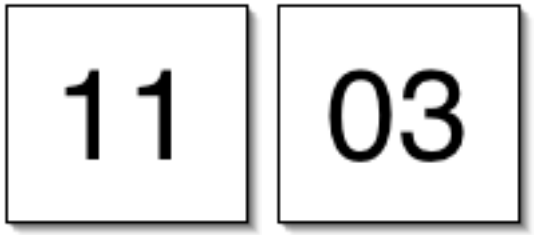


Modularising the clock display

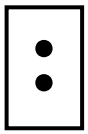


One four-digit display?

Or two two-digit displays?



And a bit of glue …



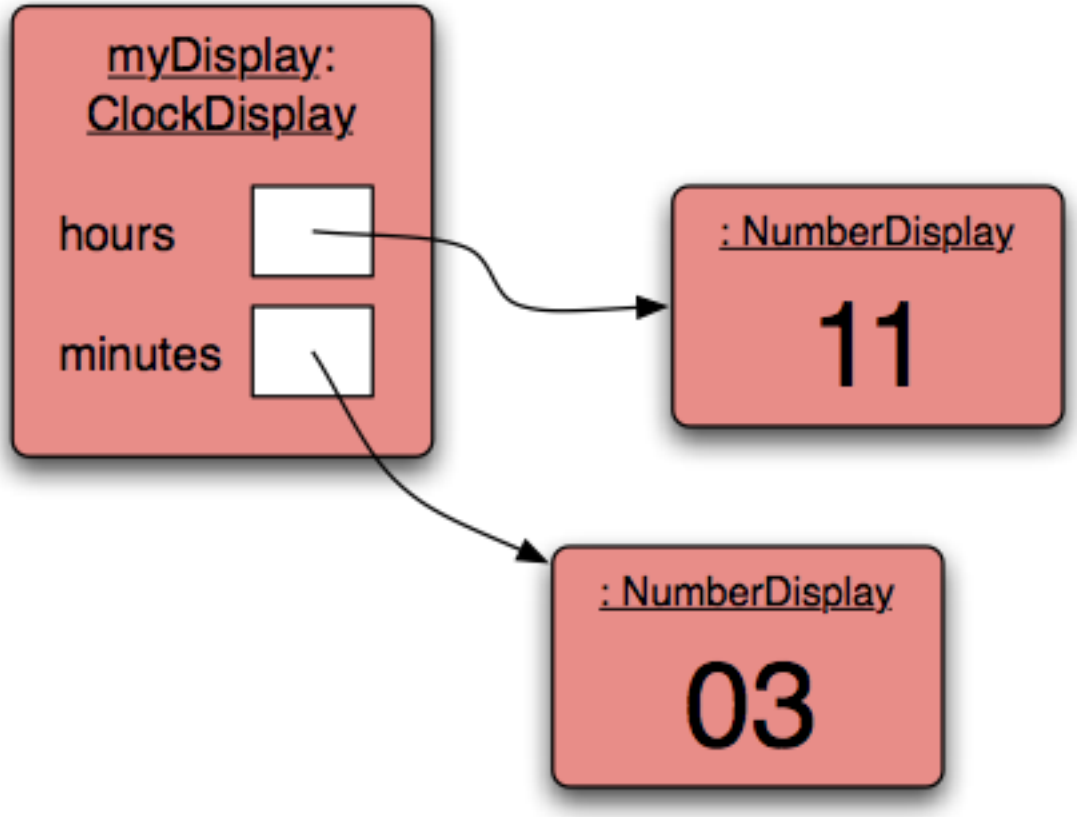
Implementation - NumberDisplay

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

    ... constructor omitted
    ... methods omitted
}
```

Implementation - ClockDisplay

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

    ... constructor omitted
    ... methods omitted
}
```

Object diagram

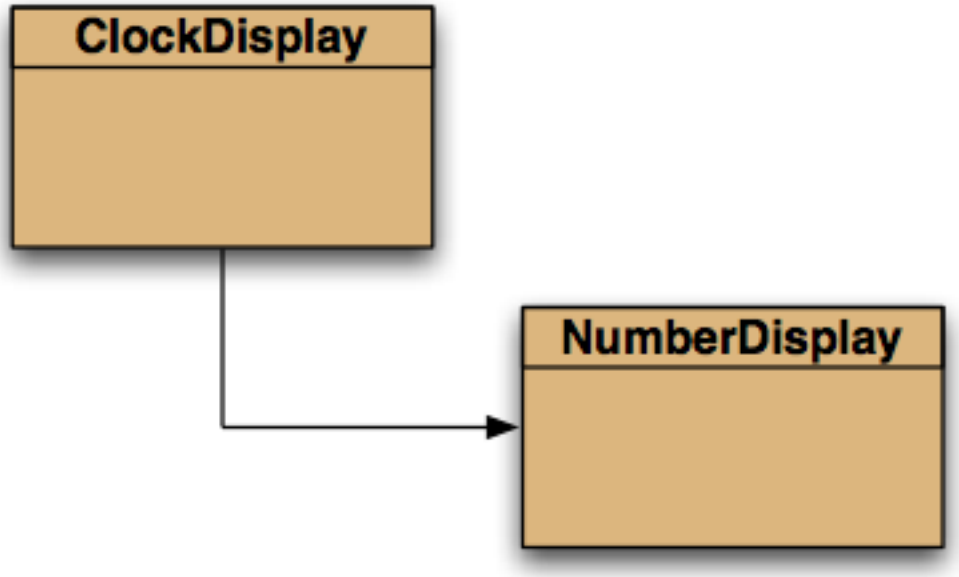


- Dynamic view at runtime (when the system is running)

Object diagram

- **Objects** exist at run-time
- An ***object diagram*** shows the objects and their relationships at one moment in time during the execution of an application
- It gives information about objects at runtime and presents the **dynamic view** of a program

Class diagram



- ClockDisplay depends on NumberDisplay
- ClockDisplay makes use of NumberDisplay

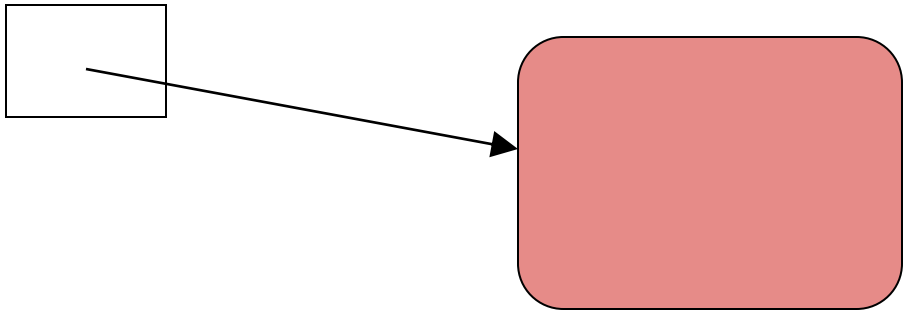
Class diagram

- **Classes** exist at compile time
- The ***class diagram*** shows the classes of an application and the relationships between them
- It gives information about the source code and presents the **static view** of a program

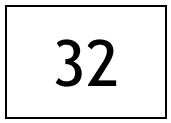
Primitive types vs. object types

```
SomeObject obj;
```

object type



```
int i;
```



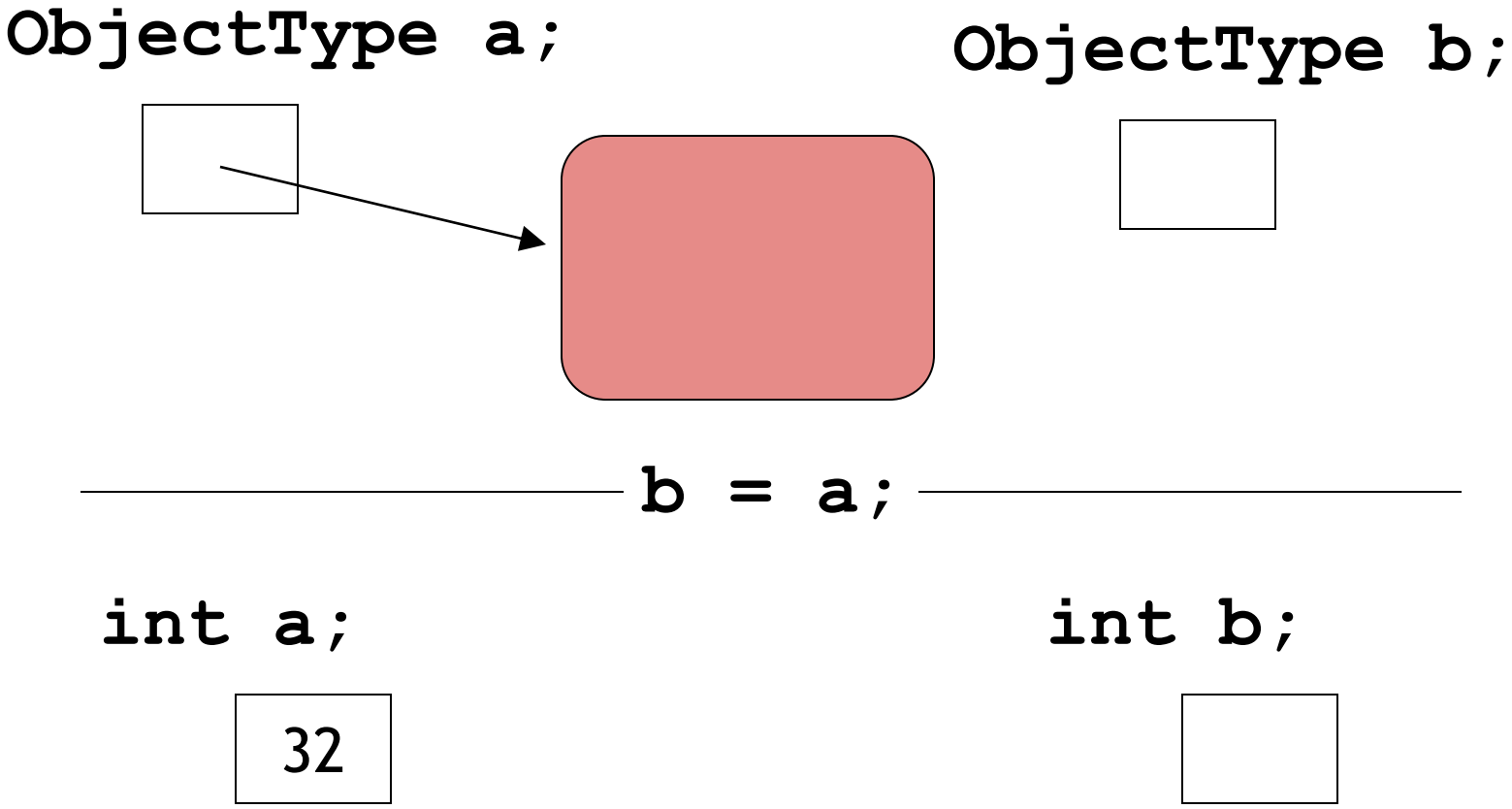
primitive type

Quiz: What is the output?

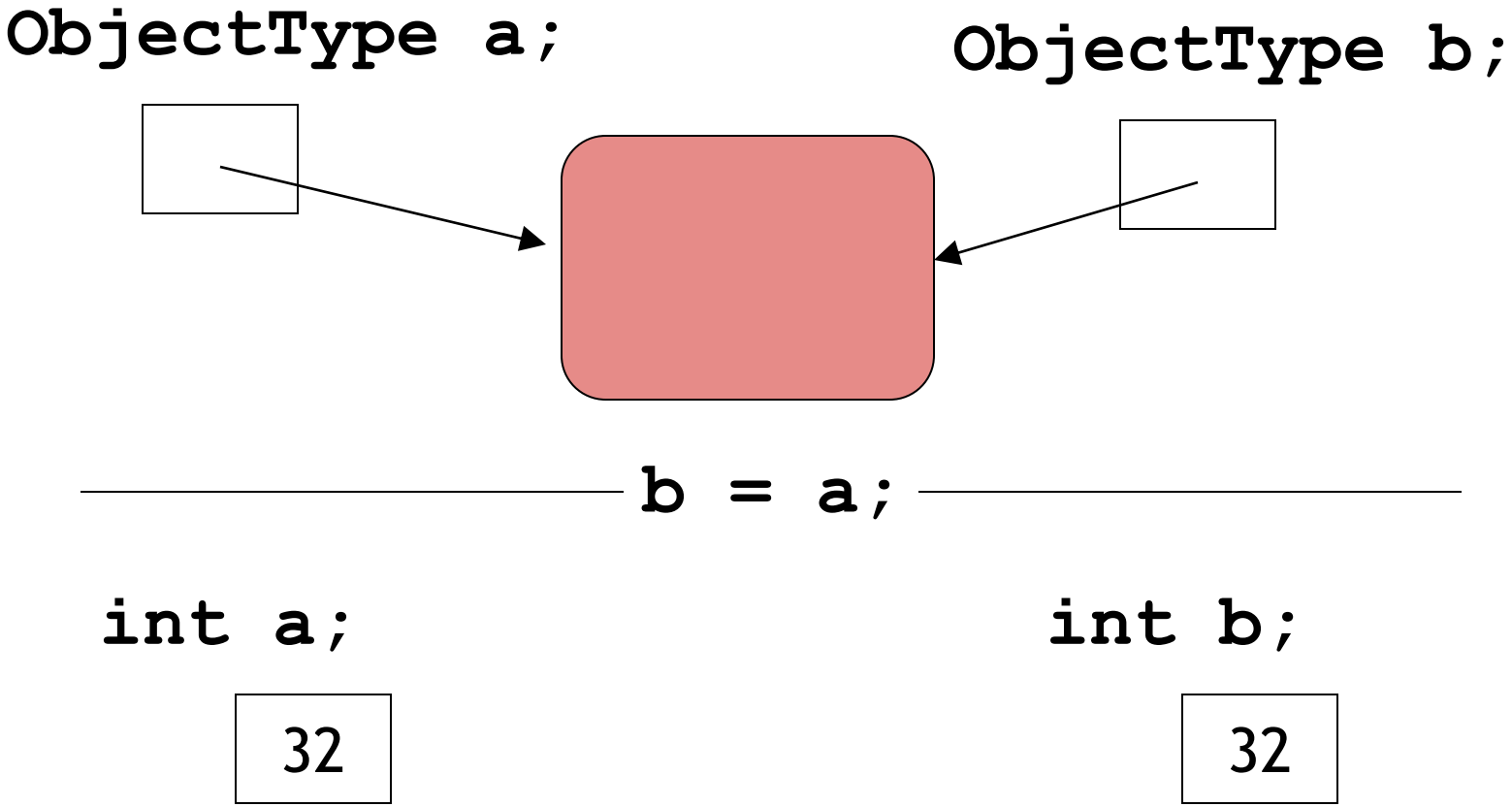
- ```
  int a;
  int b;
  a = 42;
  b = a;
  a = a + 1;                    // has b changed?
  System.out.println(b);
  ```

- ```
  Person a;
  Person b;
  a = new Person("Everett");
  b = a;
  a.changeName("Delmar");     // has b changed?
  System.out.println(b.getName());
  ```

Primitive types vs. object types



Primitive types vs. object types



Quiz: What is the output?

- ```
  int a;
  int b;
  a = 42;
  b = a;
  a = a + 1;                    // has b changed?
  System.out.println(b);
  ```


- ```
  Person a;
  Person b;
  a = new Person("Everett");
  b = a;
  a.changeName("Delmar");     // has b changed?
  System.out.println(b.getName());
  ```

Source code: NumberDisplay

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


public void increment()
{
    value = (value + 1) % limit;
}
```

The modulo operator

- The *'division'* operator (`/`), when applied to int operands, returns the *integer result* of an *integer division*.
- The 'modulo' operator (`%`) returns the *integer remainder* of an *integer division*.
- Example:

  17 / 5 = result 3, remainder 2

- In Java:

```
17 / 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

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

Concepts

- abstraction
- modularization
- primitive types
- object types
- class diagram
- object diagram
- object references