Fundamentos de la Programación Orientada a Objetos

Interacción de objetos

Programación Orientada a Objetos
Facultad de Informática

Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense Madrid

Basado en el curso Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Interacción de objetos

- Los objetos no son entes individuales
  ... cooperan
  para llevar a cabo una tarea común
- Normalmente un programa no tiene objetos de una sola clase

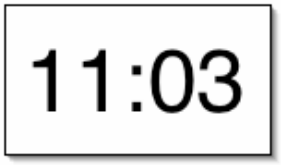
Ejemplo: un reloj digital

Abstracción y Modularización

- Abstracción
  - Habilidad para ignorar/prescindir de los detalles de las partes
  - Para centrar la atención en un nivel más alto de un problema
    - Ejemplo: mapas
- Modularización
  - Proceso de dividir un todo en partes bien definidas que pueden ser construidas y examinadas separadamente,
  - las cuales interactúan de maneras bien definidas
    - *Divide et vinces*

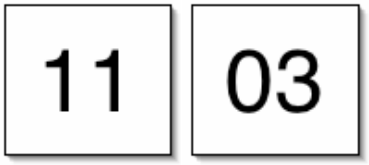
La abstracción permite ver el bosque
y la modularización los árboles que hacen el bosque

Modularización del visor del reloj



¿Un visor de cuatro dígitos?

¿O dos visores de dos dígitos?



Implementación: *NumberDisplay*

- Es más fácil que el de dos números
  - Y más reutilizable…

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

     // Constructor
     // Métodos
}
```

Implementación: *ClockDisplay*

- El reloj como la combinación de dos objetos NumberDisplay

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

     // Constructor
     // Métodos
}
```


Diagrama de objetos

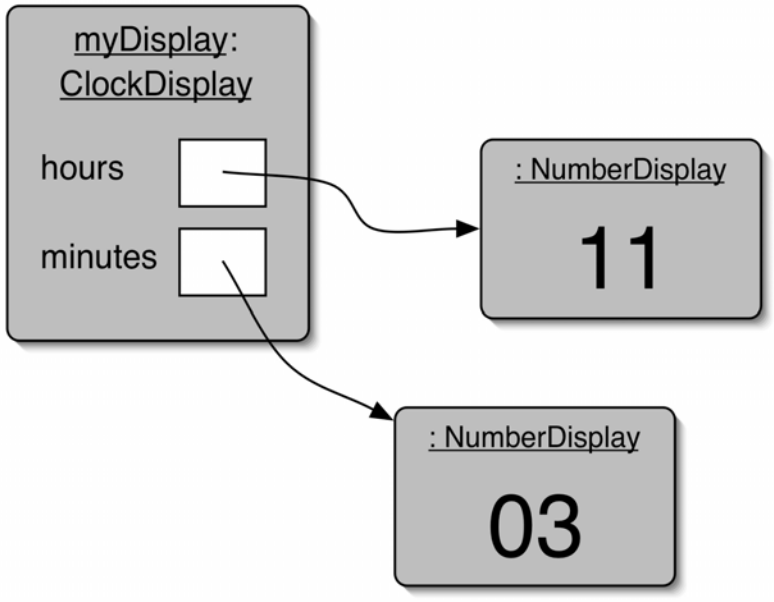
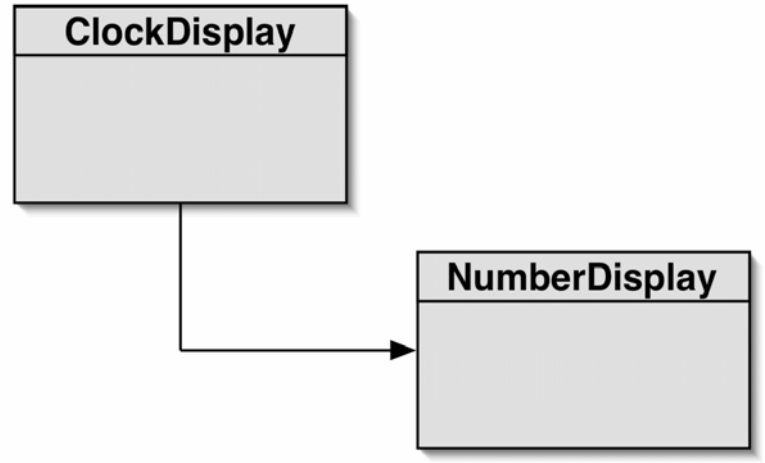


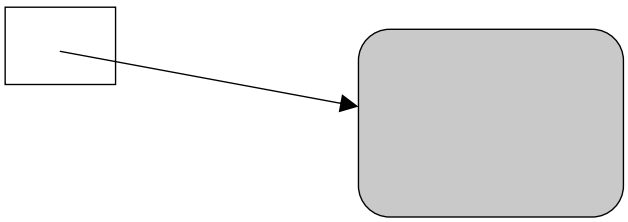
Diagrama de clases



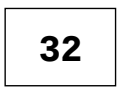
Tipos primitivos vs. Tipos de objetos

```
SomeObject obj;
```

Tipo de objeto

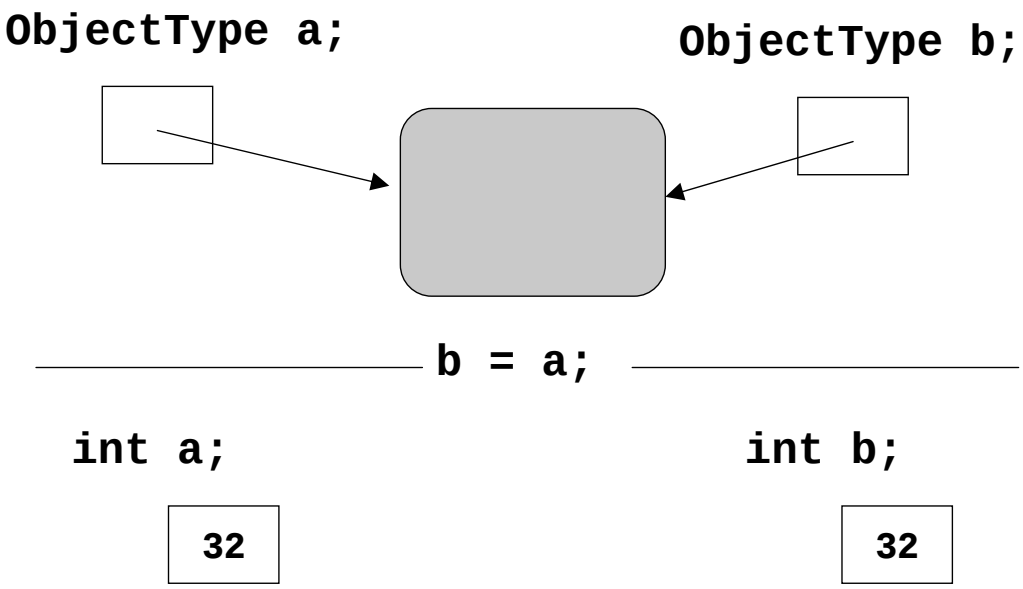


```
int i;
```



Tipo primitivo

Tipos primitivos vs. Tipos de objetos



Código Fuente: *NumberDisplay*

- Método constructor parametrizado con el máximo número que se visualizará
- Método de incremento del número
  - Gestionan el estado interno del objeto: variable value

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

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

Código Fuente: *NumberDisplay*

- Método que indica lo que hay que visualizar
  - Presentación del objeto
    - Obsérvese que el estado interno del objeto es un entero
    - Pero lo que se visualizará es un String

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

Operador de concatenación de String

Operadores

- Operadores lógicos
  - a **&&** b // true si a Y b lo son
  - a || b // true si a O b lo son
  - !a // lo contrario de a
- Operadores relacionales
  - a **==** b // igualdad (se puede utilizar para referencias)
  - a != b // desigualdad (se puede utilizar para referencias)
  - a < b // mayor
  - <= > >=
- Operadores aritméticos
  - + - * / %
- Precedencia de operadores
  - Ante la duda, usad paréntesis: (expresión)

Código Fuente: *ClockDisplay*

- Un objeto que se construye creando otros objetos

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

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


Código Fuente: *ClockDisplay*

- Y se llama a métodos de los objetos con los que se ha construido

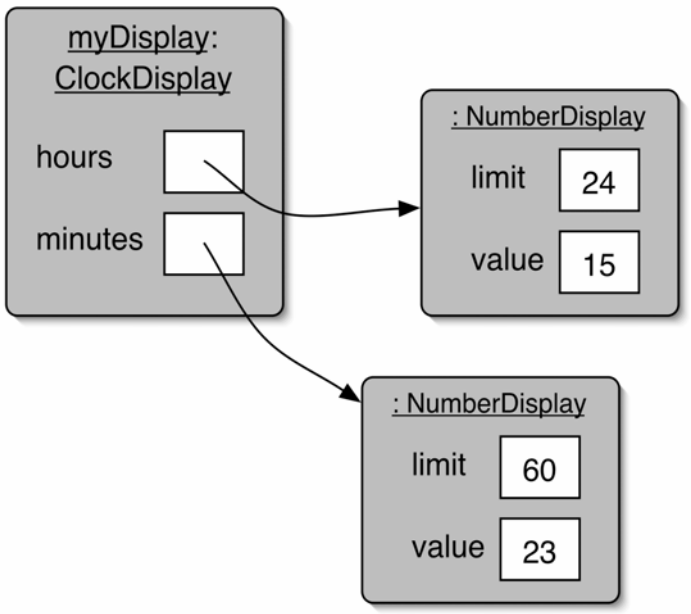
```
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) {
        // it just rolled over!
        hours.increment();
    }
    updateDisplay();
}
```

Código Fuente: *ClockDisplay*

- Método interno: private

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


Diagrama de objetos de ClockDisplay



Objetos que crean objetos

in class NumberDisplay:

```
public NumberDisplay(int rollOverLimit);
```

formal parameter

in class ClockDisplay:

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

actual parameter

Llamadas a metodos

- Llamadas a métodos internos (de la misma clase)

```
updateDisplay();
...
private void updateDisplay()
```

- Llamadas a métodos externos
  - Obligatorio especificar la referencia al objeto externo

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

- En el primer caso también se podría hacer

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

Llamadas a metodos

- En general

objeto . *método* (*lista_de_parámetros*)

Resumen

- abstraction
- modularization
- classes define types
- class diagram
- object diagram
- object references
- primitive types
- object types
- object creation
- overloading
- internal/external method call
- debugger