UNIVERSITÀ DEGLI STUDI DI MILANO
matteo.re@unimi.it
https://homes.di.unimi.it/re/arch2-lab-2015-2016.html

DIPARTIMENTO DI INFORMATICA
Laboratorio di Architetture degli Elaboratori II
Turno B : (G – Z)

Oggetti

SOMMARIO:

1) Jump Table
2) Istruzione jalr
3) Librerie caricate dinamicamente
4) Oggetti (dal punto di vista concettuale)
5) Oggetti (condivisione in memoria del codice macchina)
6) Oggetti in memoria statica
7) Esercizi

Ultimo aggiornamento: 27/5/2016

In un linguaggio orientato agli oggetti, un set di dati e procedure viene raggruppato in un **oggetto**. La programmazione avviene creando oggetti e invocando le **procedure** all'interno degli oggetti (in alcuni linguaggi ci si riferisce ad esse con il nome di **metodi**).

In questo laboratorio vedremo come si presentano gli oggetti a basso livello.

Indirizzi in memoria

```
.text
sub1:     li       $v0,4              # syscall code 4, print string
          la       $a0,messH          # indirizzo della stringa da stampare
          syscall                     # chiamata
          jr       $ra                # ritorno al chiamante
          .data
messH:    .asciiz  "Hello "

          .text
sub2:     li       $v0,4              # syscall code 4, print string
          la       $a0,messW          # indirizzo della stringa da stampare
          syscall                     # chiamata
          jr       $ra                # ritorno al chiamante
          .data
messW:    .asciiz  "World\n"

.data
sub1add:  .word sub1                  # indirizzo della prima subroutine
sub2add:  .word sub2                  # indirizzo della seconda subroutine
```

In un linguaggio orientato agli oggetti, un set di dati e procedure viene raggruppato in un **oggetto**. La programmazione avviene creando oggetti e invocando le **procedure** all'interno degli oggetti (in alcuni linguaggi ci si riferisce ad esse con il nome di **metodi**).

In questo laboratorio vedremo come si presentano gli oggetti a basso livello.

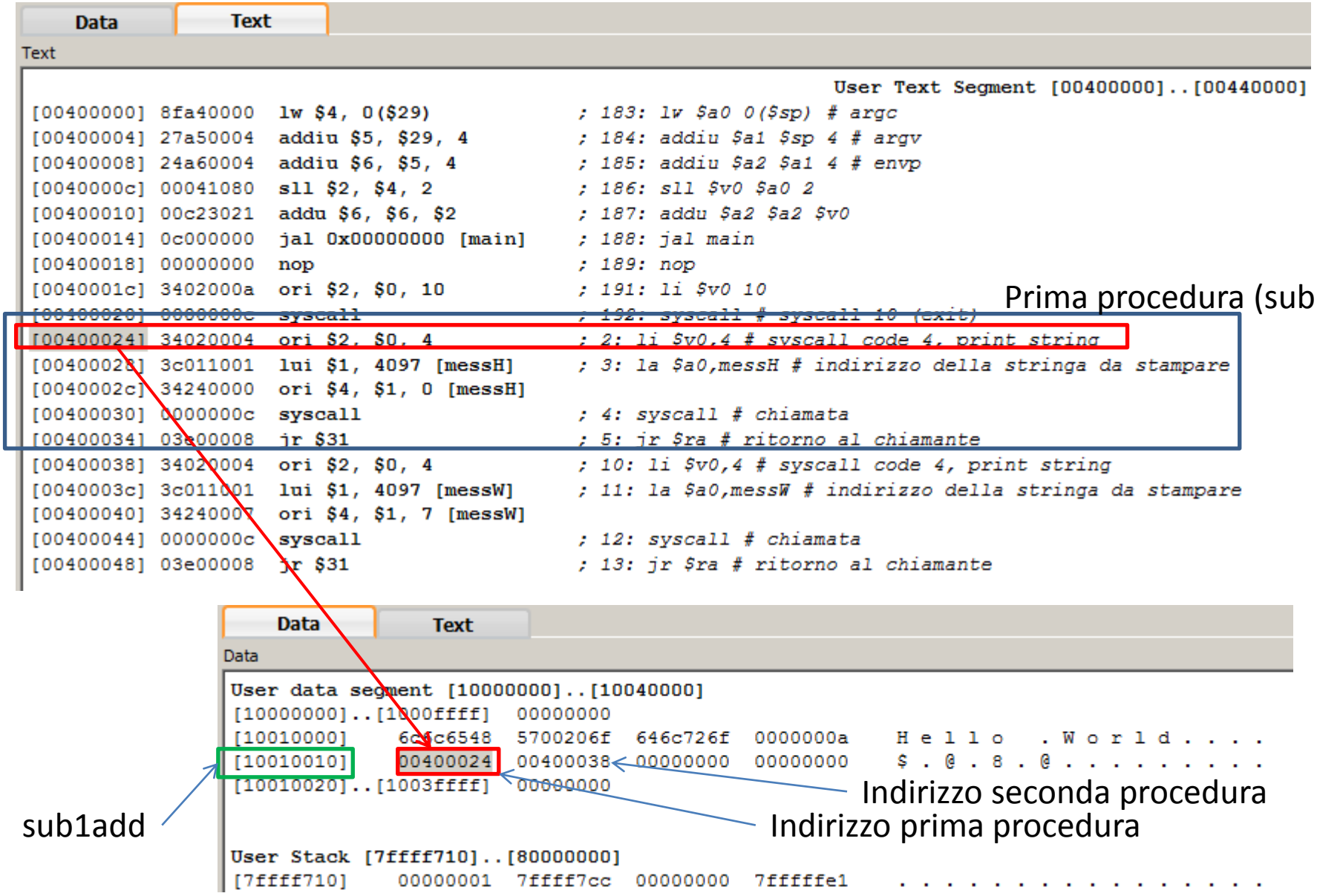
Indirizzi in memoria

```
.text
sub1:   li      $v0,4
        la      $a0,messH
        syscall
        jr      $ra
        .data
messH:  .asciiz "Hello "

        .text
sub2:   li      $v0,4
        la      $a0,messW
        syscall
        jr      $ra
        .data
messW:  .asciiz "World\n"

.data
sub1add:  .word sub1
sub2add:  .word sub2
```

L'indirizzo simbolico sub1 indica il primo byte dell'indirizzo a cui viene caricata la subroutine 1 a runtime.

Nella sezione .data l'indirizzo rappresentato da sub1 viene salvato in memoria, all'indirizzo rappresentato dall'indirizzo simbolico (o, se preferite, etichetta) sub1add

Indirizzi in memoria



All'indirizzo **sub1add** è presente l'indirizzo del punto di ingresso in **sub1**

Torniamo a guardare il sorgente di esempio …

```
.text
sub1:       li        $v0,4
            la        $a0,messH
            syscall
            jr        $ra
            .data
messH:      .asciiz   "Hello "

            .text
sub2:       li        $v0,4
            la        $a0,messW
            syscall
            jr        $ra
            .data
messW:      .asciiz   "World\n"

.data
sub1add:    .word sub1
sub2add:    .word sub2
```

Una jump table non è altro che una lista di indirizzi di punti di ingresso (di procedure) a cui è possibile saltare.

Normalmente sarebbe presente una procedura **main** dalla quale chiamare le altre procedure ma, al momento, essa non è ancora presente nel sorgente di esempio.

Domanda: Qual è l'istruzione che si utilizza per passare il controllo ad una procedura/funzione?
jal (jump and link)

Torniamo a guardare il sorgente di esempio …

```
.text
sub1:       li        $v0,4
            la        $a0,messH
            syscall
            jr        $ra
            .data
messH:      .asciiz   "Hello "

            .text
sub2:       li        $v0,4
            la        $a0,messW
            syscall
            jr        $ra
            .data
messW:      .asciiz   "World\n"

.data
sub1add:    .word sub1
sub2add:    .word sub2
```

Una jump table non è altro che una lista di indirizzi di punti di ingresso (di procedure) a cui è possibile saltare.

Normalmente sarebbe presente una procedura **main** dalla quale chiamare le altre procedure ma, al momento, essa non è ancora presente nel sorgente di esempio.

Domanda: Qual è l'istruzione che si utilizza per passare il controllo ad una procedura/funzione?
jal (jump and link)

Istruzione jalr

```
.text
sub1:       li        $v0,4
            la        $a0,messH
            syscall
            jr        $ra
            .data
messH:      .asciiz   "Hello "

            .text
sub2:       li        $v0,4
            la        $a0,messW
            syscall
            jr        $ra
            .data
messW:      .asciiz   "World\n"

.data
sub1add:    .word sub1
sub2add:    .word sub2
```

Jump table

In un programma una specifica istruzione jal passa il controllo ad una specifica funzione/procedura.

Ma come possiamo fare se vogliamo che il salto sia verso procedure differenti a seconda delle circostanze? Questa è esattamente la situazione in cui tornano utili le jump table.

La jump table contiene una lista di entry point di procedure.

Per chiamare una procedura:

1) Copiamo il suo entry point in un registro (r)
2) Usiamo **jalr r** # $ra <- PC +4
   # PC <- $r

Indirizzo di ritorno

Carico in PC l'indirizzo contenuto in $r

jalr funziona come jal ma l'indirizzo è prelevato da un registro

Istruzione jalr

```
.text
main:
          lw      $t0, .......    # preleva il primo entry point dalla
                                  # Jump Table

          jalr    ........       # passa il controllo a sub1

          li      $v0,10          # syscall exit
          syscall

.data
sub1add:  .word   sub1             # Jump Table
sub2add:  .word   sub2
```

Jump table

Istruzione jalr

```
.text
main:
        lw      $t0, sub1add    # preleva il primo entry point dalla
                                # Jump Table

        jalr    $t0             # passa il controllo a sub1

        li      $v0,10          # syscall exit
        syscall

.data
sub1add:  .word   sub1            # Jump Table
sub2add:  .word   sub2
```

Jump table

Esempio completo utilizzo Jump Table

```
.globl   main
.text
main:

          lw       $t0,sub1add        # prelevo primo entry point
          jalr     $t0                # passo controllo a sub1

          lw       $t0,sub2add        # prelevo secondo entry point
          jalr     $t0                # passo controllo a sub2

          li      $v0,10              # syscall exit
          syscall

          .data
sub1add:  .word sub1                 # Jump Table
sub2add:  .word sub2

          .text
sub1:     li       $v0,4
          la       $a0,messH
          syscall
          jr       $ra
          .data
messH:    .asciiz  "Hello "

          .text
sub2:     li       $v0,4
          la       $a0,messW
          syscall
          jr       $ra
          .data
messW:    .asciiz  "World\n"
```

(esempio2.asm)

Domanda: quanty byte ci sono in ogni entry della Jump Table? 4

Jump Table dinamiche (simulazione)

Spesso i s.o. supportano il caricamento dinamico, in cui il caricamento in memoria del codice macchina di una procedura avviene solo quando un programma lo richiede.

Immediatamente dopo che il codice della procedura è stato caricato in memoria il suo punto di ingresso viene aggiunto ad una Jump Table e la procedura può essere chiamata.

Collezioni più o meno estese di codice macchina di procedure e/o funzioni che possono essere rese disponibili in questo modo costituiscono librerie che possono essere caricate dinamicamente.

Jump Table dinamiche (simulazione)

```
.globl   main
.text
main:

         # Assumiamo che quando il programma è caricato in moemoria sub1 e sub2 non siano caricate
         # insieme ad esso. Ad un certo punto della sua esecuzione il programma necessita di sub1
         # Solo a questo punto sub1 è caricata in memoria e il suo entry point è aggiunto alla jump table.

         lw       $t0,jtable          # chiamata a sub1
         jalr     $t0                 # passa controllo a sub1

         . . . .
         # Il programma continua ad eseguire ma, ad un certo punto gli serve sub2. Anche in questo caso
         # sub2 viene caricata in memoria e il suo entry point è aggiunto alla jump table.

         lw       $t0,jtable+4        # chiamata a sub2
         jalr     $t0                 # passa controllo a sub2

         li      $v0,10               # syscall exit
         syscall


.data
jtable:
         .word sub1                   # Jump Table (immaginiamo che gli
         .word sub2                   # indirizzi vengano aggiunti a run time).

         .globl   sub1
.text
sub1:    li       $v0,4
         la       $a0,messH
         syscall
         jr       $ra
.data
messH:   .asciiz  "Hello "

         .globl   sub2
.text
sub2:    li       $v0,4
         la       $a0,messW
         syscall
         jr       $ra
.data
messW:   .asciiz  "World\n"
```

NBB: diversi programmi possono condividere la medesima libreria caricata dinamicamente. Questo permette a s.o. di risparmiare memoria

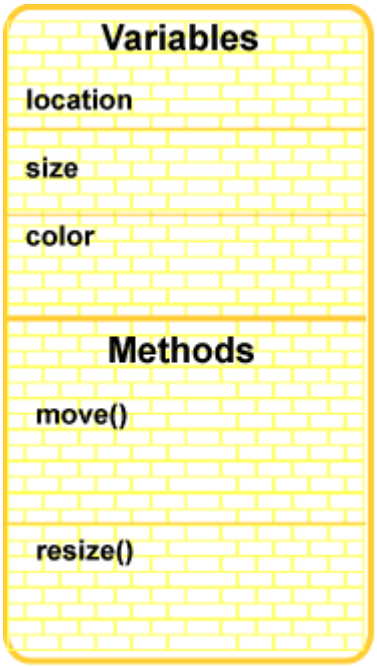
Oggetti

Gli oggetti sono spesso descritti in questo modo:

- Un oggetto ha **identità** (nel senso che agisce cone un tutt'uno)
- Un oggetto ha uno **stato** (nel senso che ha variabili interne, proprietà che possono cambiare)
- Un oggetto ha un **comportamento** (può decidere di agire su altri oggetti o di permettere che altri oggetti agiscano su di esso)

Può essere utile (anche se non è molto accurato) pensare ad un oggetto come a un unico e contiguo blocco di memoria contenente sia dati che codice eseguibile, come mostrato in figura.

In programmazione orientata agli oggetti le procedure interne ad un oggetto sono chiamate metodi. Quindi un oggetto è «composto» da dati e da metodi.

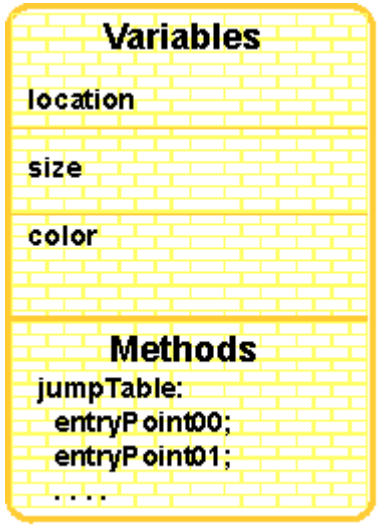


Domanda: Se un programma costruisse 1000 copie di un oggetto sarebbe efficiente dublicare 1000 volte il codice dell'oggetto (quello in .text)?

Oggetti

Piuttosto che duplicare 1000 volte il codice macchina di un oggetto il sistema che si occupa di istanziare gli oggetti carica un'unica copia in memoria del codice macchina comune a tutti gli oggetti di un determinato tipo e poi assegna ad ogni oggetto memoria solo per i suoi dati (che sono personali per ogni istanza dell'oggetto). In seguito ogni oggetto istanziato ottiene anche una Jump Table contenente gli entry point di tutti i suoi metodi.

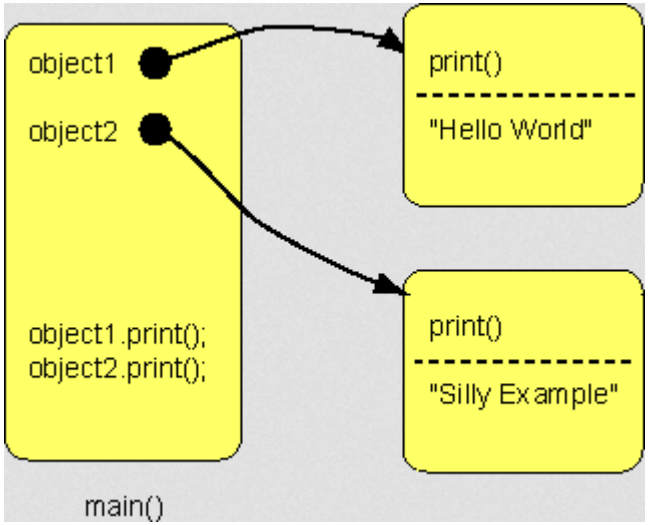
Come risultato si verifica un consistente risparmio di memoria indipendentemente dal numero degli oggetti dello stesso tipo da istanziare.



Esempio programma contenente oggetti

Scriviamo un semplice programma contenente due oggetti. Concettualmente ogni oggetto consiste di una stringa contenente un messaggio e di un metodo print() che stampa il messaggio. Nell'implementazione il codice del metodo sarà fuori dall'oggetto. La figura mostra il programma.

La main() contiene gli indirizzi di ogni oggetto. Questo è rappresentato dalle frecce che puntano ad ogni oggetto. Il programma chiamerà prima il metodo print() del primo oggetto e poi il metodo print() del secondo oggetto.



Questa parte del codice, se fosse scritta ad esempio in Java, potrebbe essere così:

```
object1.print();
object2.print();
```

Esempio programma contenente oggetti

```
.globl   main
.text
main:                                   # object1.print();
          la       $a0,object1          #   indirizzo primo oggetto
          lw       $t0,0($a0)           #   indirizzo del metodo del primo oggetto
          jalr     $t0                  #   chiamata del metodo dell'oggetto

                                        # object2.print();
          la       $a0,object2          #   indirizzo secondo oggetto
          lw       $t0,0($a0)           #   indirizzo del metodo del secondo
oggetto
          jalr     $t0                  #   chiamata del metodo dell'oggetto

          li      $v0,10                # ritorna il controllo al s.o.
          syscall

# codice per il metodo print
          . . .

.data
object1:  .word    print                    # Jump Table
          .asciiz  "Hello World\n"          # dati oggetto

object2:  .word    print                    # Jump Table
          .asciiz  "Silly Example\n"        # dati oggetto
```

In java gli oggetti sono costruiti dinamicamente in memoria. Per semplificare esempio (e codice) usiamo memoria statica ….

Esempio programma contenente oggetti

```
.globl   main
.text
main:
          la        $a0,object1
          lw        $t0,0($a0)
          jalr      $t0


          la        $a0,object2
          lw        $t0,0($a0)
          jalr      $t0

          li       $v0,10
          syscall

# codice per il metodo print
          . . .

.data
object1:  .word     print
          .asciiz   "Hello World\n"

object2:  .word     print
          .asciiz   "Silly Example\n"
```

Ogni oggetto ha questa struttura:

byte 0-3 indirizzo di un metodo
(single entry Jump Table)
byte 4- stringa
(dimensione variabile)

La Jump Table consiste di un solo indirizzo: l'entry point di print() che si trova all'inizio dell'oggetto. Per copiare questo indirizzo da object1 a $t0 si utilizza il seguente codice:

```
la      $a0,object1      # ottieni indirizzo di
                         # object1
lw      $t0,0($a0)       # ottieni indirizzo del
                         # metodo di object1
```

Ora che l'entry point del metodo di object1 è in $t0 il metodo può essere chiamato

Codice metodo print()

```
# Singola copia del metodo print()
# Parametro: $a0 == indirizzo dell'oggetto
.text
print:
          li       $v0,4                    # syscall print string
          addu     $a0,$a0,4                # indirizzo della stringa dell'oggetto
          syscall                           #
          jr       $ra                      # ritorno al chiamante
```

Esiste solo una copia di questo metodo … ma ci aspettiamo che essa si comporti come se facesse parte di tutti gli oggetti. Questo si ottiene copiando in $a0 l'indirizzo dell'oggetto che deve usare il metodo print. Questa operazione viene effettuata in main() come segue:

```
                                    # object1.print();
          la       $a0,object1      #   indirizzo del primo oggetto
          lw       $t0,0($a0)       #   indirizzo del metodo del primo oggetto
          jalr     $t0              #   chiamata del metodo dell'oggetto
                                    #   L'indirizzo dell'oggetto è
                                    #   in $a0.
```

In print() l'indirizzo della stringa all'interno dell'oggetto che sta utilizzando print è calcolato come segue:

```
          addu     $a0,$a0,4           # indirizzo della stringa dell'oggetto
```

Questo carica l'indirizzo della stringa in $a0 dove il metodo print() si aspetta che sia.

Esempio completo

```
.globl   main
.text
main:                                   # object1.print();
          la        $a0,object1         #   indirizzo del primo oggetto
          lw        $t0,0($a0)          #   indirizzo del metodo del primo oggetto
          jalr      $t0                 #   chiamata metodo

                                        # object2.print();
          la        $a0,object2         #   indirizzo del secondo oggetto
          lw        $t0,0($a0)          #   indirizzo del metodo del secondo oggetto
          jalr      $t0                 #   chiamata metodo

          li       $v0,10               # ritorna controllo a s.o.
          syscall

# Oggetto costruito in memoria statica. Un oggetto consiste di dati
# per ogni oggetto e una Jump Table di entry point comune a tutti gli oggetti
# del medesimo tipo.
.data
object1:
          .word    print                  # Jump Table
          .asciiz  "Hello World\n"        # dati di questo oggetto

object2:
          .word    print                  # Jump Table
          .asciiz  "Silly Example\n"      # dati di questo oggetto

# Singola copia del metodo print()
# Parametro: $a0 == indirizzo oggetto
.text
print:
          li        $v0,4                 # metodo print string
          addu      $a0,$a0,4             # indirizzo stringa oggetto
          syscall                         #
          jr        $ra
```

Esercizio 1

Aggiungere un secondo metodo agli oggetti dell'esempio precedente. Per riuscirci dovete:

Aggiungere un elemento alla Jump Table di ogni oggetto
Aggiungere il codice per il nuovo metodo

Questo cambierà il layout dell'oggetto e dovrete introdurre altri (piccoli) cambiamenti nel codice dell'esempio.

Esercizio 2

Aggiungere un terzo metodo agli oggetti dell'esempio precedente. Il metodo dovrà essere in grado di creare e popolare dinamicamente una lista concatenata.