Lezione 6

Costrutti Iterativi

Lezione 6 - Obiettivi

Al termine di questa lezione saremo in grado di

- Realizzare costrutti iterativi in un programma utilizzando l'istruzione while.
- Realizzare costrutti iterativi in un programma utilizzando la coppia di istruzioni do–while.
- Realizzare costrutti iterativi in un programma utilizzando l'istruzione for.
- Annidare costrutti iterativi.
- Scegliere il costrutto iterativo più appropriato a seconda del contesto.
- Utilizzare la classe ResponseBox del package javabook.
- Visualizzare dati utilizzando la classe Format del package javabook.

Costrutto while

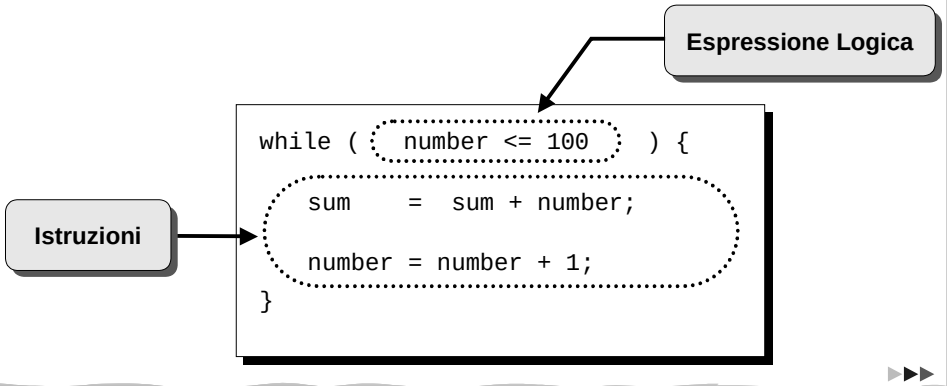
```
int sum = 0, number = 1;

while ( number <= 100 ) {
   sum    =  sum + number;

   number = number + 1;
}
```

Queste istruzioni vengono eseguite fintanto che number è minore o uguale a 100.

 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 3

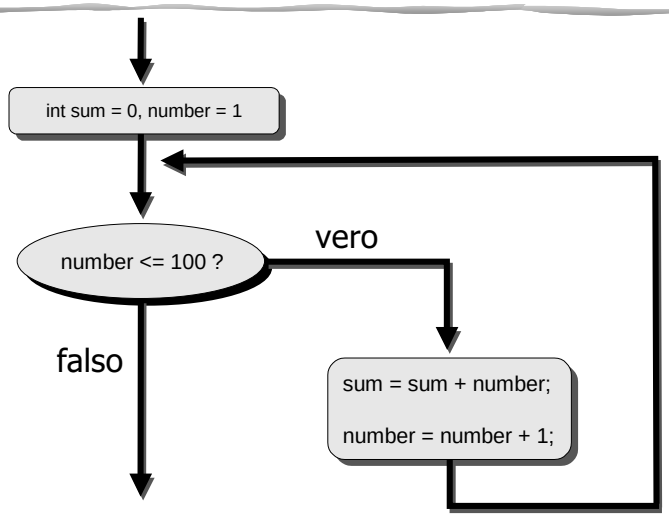
Sintassi del Costrutto while

```
while ( <espressione logica> )

      <corpo del ciclo>
```



 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 4

Flusso di Controllo while



 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 5

Altri Esempi

1

```
int sum = 0, number = 1;

while ( sum <= 1000000 ) {
    sum    =  sum + number;
    number = number + 1;
}
```

Continua ad aggiungere i numeri 1, 2, 3, … sino a che la somma diventa maggiore di 1.000.000.

2

```
int product =  1, number = 1,
    count   = 20, lastNumber;

lastNumber = 2 * count - 1;

while (number <= lastNumber) {
    product = product * number;
    number  = number + 2;
}
```

Calcola il prodotto dei primi 20 interi dispari.

 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 6

Esempio: Controllo dei Valori in Ingresso

- Un esempio realistico di utilizzo del costrutto while per filtrare i valori in ingresso non validi.
- Accetta età tra 0 e 130, estremi esclusi.

Questa istruzione deve essere ripetuta.

```
age = inputBox.getInteger("Età (tra 0 e 130):");

while (age < 0 || age > 130) {
    messageBox.show("Età non valida." +
                    "Si prega riprovare.");

    age = inputBox.getInteger( "Età (tra 0 e 130):" );

}
```


Possibili Errori con Costrutti while - 1

1

```
int product = 0;

while ( product < 500000 ) {
    product = product * 5;
}
```

2

```
int count = 0;

while ( count != 10 ) {
    count = count + 2;
}
```

Cicli Infiniti
Entrambe i cicli non termineranno perchè le espressioni logiche non diventeranno mai false.

Possibili Errori con Costrutti while - 2

1
```
float count = 0.0f;

while ( count != 1.0f ) {
    count = count + 0.3333333f;
}       //sette cifre decimali
```

2
```
float count = 0.0f;

while ( count != 1.0f ) {
    count = count + 0.33333333f;
}       //otto cifre decimali
```

Utilizzo di Real
Il ciclo 2 termina, ma non il ciclo 1, in quanto i numeri reali sono necessariamente approssimati per essere contenuti nella memoria del calcolatore.

 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 9

Possibili Errori con il Costrutto while - 3

- Obiettivo: eseguire il corpo del ciclo 10 volte.

1
```
count = 1;
while ( count < 10 )
{
    . . .
    count++;
}
```
✗

2
```
count = 1;
while ( count <= 10 )
{
    . . .
    count++;
}
```
✓

3
```
count = 0;
while ( count <= 10 )
{
    . . .
    count++;
}
```
✗

4
```
count = 0;
while ( count < 10 )
{
    . . .
    count++;
}
```
✓

① e ③ contengono l'errore off-by-one (fuori di uno).

 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 10

Controlli sui Costrutti Iterativi

1. Attenzione all'errore off-by-one.
2. Assicurarsi che il ciclo contenga un'istruzione che consentirà al ciclo stesso di terminare.
3. Assicurarsi che il ciclo sia ripetuto il numero corretto di volte.
4. Volendo eseguire il corpo di un ciclo N volte è necessario inizializzare una variabile contatore a 0 e utilizzare il test "contatore < N", oppure inizializzare una variabile contatore a 1 e utilizzare il test "contatore <= N".

 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 11

Abbreviazione degli Operatori

```
sum = sum + number;
```

```
sum += number;
```

Operatore	Utilizzo	Significato
+=	a += b;	a = a + b;
-=	a -= b;	a = a – b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 12

Il Costrutto do-while

```
int sum = 0, number = 1;

do {
   sum += number;
   number++;
} while ( sum <= 1000000 );
```

Queste istruzioni sono eseguite fintanto che la somma è minore uguale a 1.000.000.

 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 13

Sintassi del Costrutto do-while

```
do
      <istruzioni>

while ( <espressione logica> ) ;
```

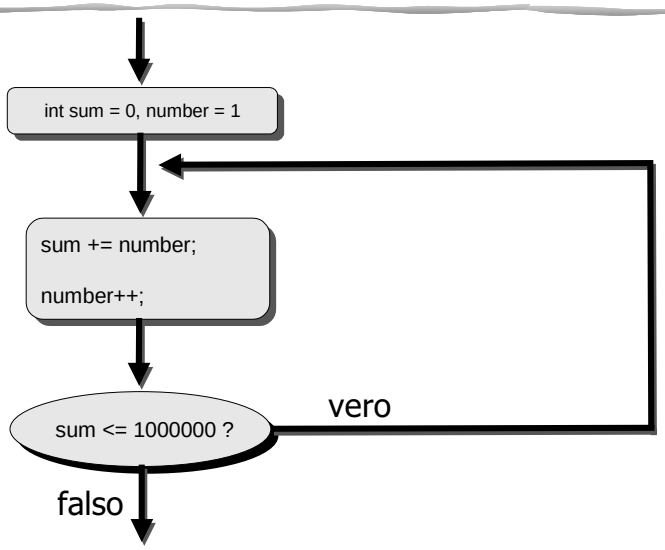
```
do   {
   sum += number;
   number++;
}   while (   sum <= 1000000   );
```

Istruzioni

Espressione Logica

 Introduction to Object-Oriented Programming with Java--Wu Chapter 7 - 14

Flusso di Controllo do-while



Utilizzo di Variabili Booleane nei Cicli -1

```
sum = 0;
do {
    num = inputBox.getInteger();

    if (num == 0)                          //sentinella
        messageBox.show("Somma = " + sum);
    else if (num % 2 == 0)                 //dato non valido
        messageBox.show("Errore: numero pari!");
    else {
        sum += num;
        if (sum > 1000)                    //soglia
           messageBox.show(“La somma è maggiore di 1000");
    }
} while ( !(num % 2 == 0 || num == 0 || sum > 1000) );
```


Utilizzo di Variabili Booleane nei Cicli - 2

```
boolean repeat = true;  sum = 0;
do {
     num = inputBox.getInteger();
     if (num == 0)  {                //sentinella
          messageBox.show("Somma = " + sum);
          repeat = false;
     }
     else if (num % 2 == 0)  {       //dato non valido
          messageBox.show("Errore: numero pari!");
          repeat = false;
     }
     else {
          sum += num;
          if (sum > 1000) {          //soglia
              messageBox.show("La somma è maggiore di 1000");
              repeat = false;
          }
     }
} while ( repeat );
```

Termina il ciclo assegnando repeat a falso.

ResponseBox

- La classe ResponseBox può essere utilizzata per sollecitare una particolare risposta (affermativa o negativa) dall'utente.

```
MainWindow   mainWindow = new MainWindow( );
ResponseBox  yesNoBox
                  = new ResponseBox( mainWindow );

yesNoBox.prompt( "Do you love Java?");
```

ResponseBox – Gestire la Selezione

- Per determinare quale bottone è stato premuto:

```
int selection = yesNoBox.prompt(“Premi un bottone");

switch (selection) {

   case ResponseBox.YES:
         messageBox.show(“Hai premuto Yes");
         break;

   case ResponseBox.NO:
         messageBox.show(“Hai premuto No“);
         break;
}
```


ResponseBox – Esempio

- Utilizzo tipico di ResponseBox:

```
choice = yesNoBox.prompt
         (“Vuoi iniziare ad eseguire i calcoli?");

while (choice == ResponseBox.YES) {

   //qui vengono eseguiti i calcoli

   choice = yesNoBox.prompt
               (“Altri calcoli? ");
}
```


ResponseBox – Altri Usi

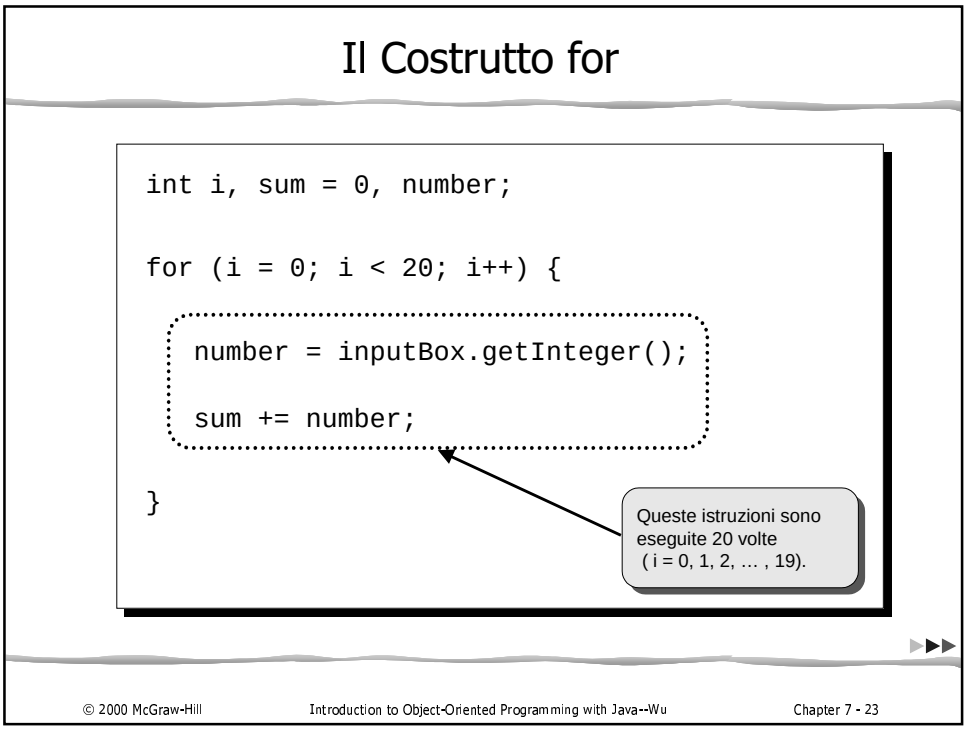
- ResponseBox può avere sino a tre bottoni con etichette designate dall'utente.

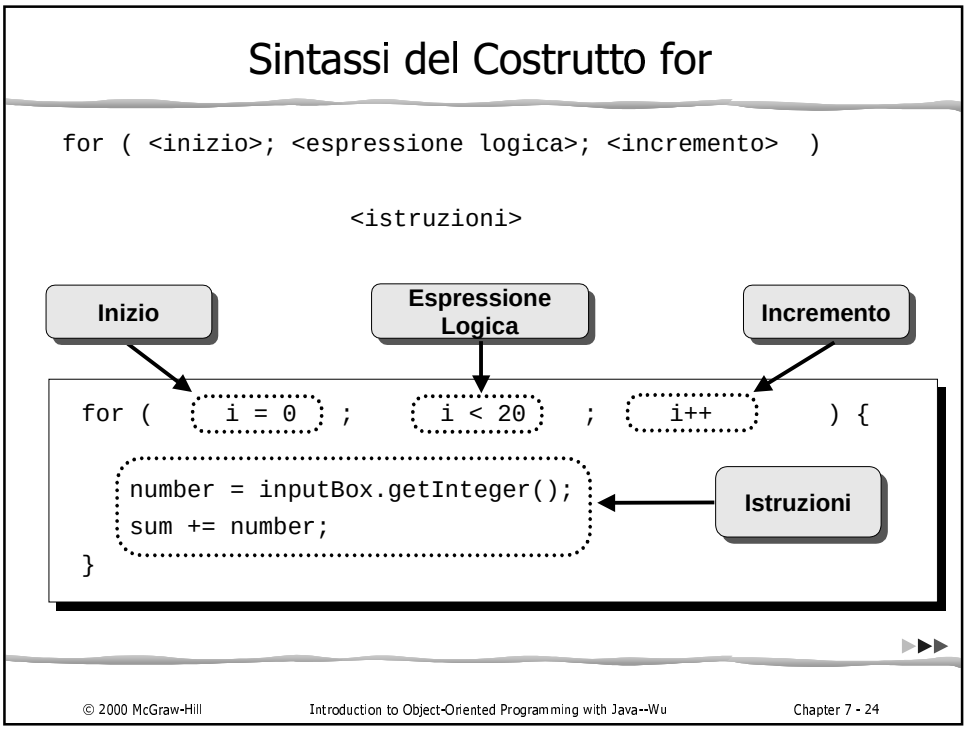
```
ResponseBox   threeButtonBox;

threeButtonBox = new ResponseBox(mainWindow,3);

threeButtonBox.setLabel( ResponseBox.BUTTON1, "OK"     );
threeButtonBox.setLabel( ResponseBox.BUTTON2, "Cancel" );
threeButtonBox.setLabel( ResponseBox.BUTTON3, "Help"   );
```


Metodi di ResponseBox

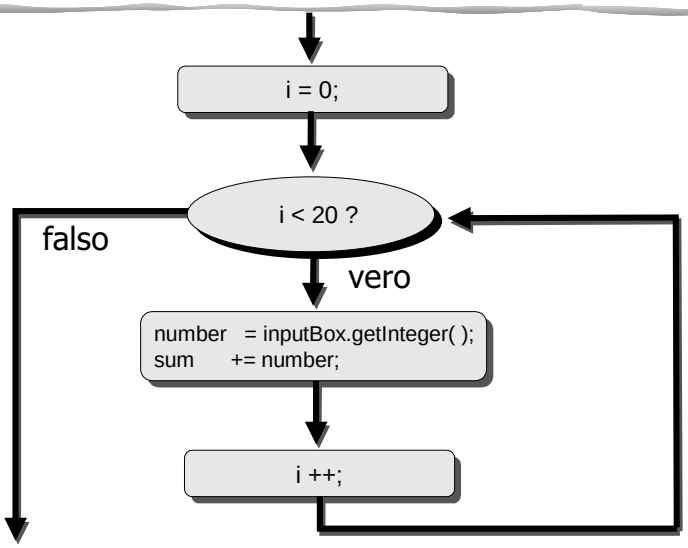
CLASS:	ResponseBox	
Method	Argument	Description
`<constructor>`	`MainWindow`	Creates a ResponseBox object.
`<constructor>`	`MainWindow, int`	Creates a ResponseBox object with N (the second argument) buttons, 1 <= N <= 3. If an invalid N is passed, then the object will include one button.
`prompt`	`String`	Prompts the user with the text passed as an argument. Returns an integer that identifies the clicked button. See the explanation of the class constants.
`setLabel`	`int, String`	Sets the label of the designated button with the passed String. The first argument identifies the button. See the explanation of the class constants.

Class Constant	Description
`YES`	This value identifies the Yes button.
`NO`	This value identifies the No button.
`BUTTON1`	This value identifies the leftmost button. The value of BUTTON1 is equal to the value of YES.
`BUTTON2`	This value identifies the middle button. Note: the middle button becomes the rightmost button if there are only two buttons. The value of BUTTON2 is equal to the value of NO.
`BUTTON3`	This value identifies the rightmost button when the ResponseBox includes three buttons.


Il Costrutto for
int i, sum = 0, number;
for (i = 0; i < 20; i++) {
number = inputBox.getInteger();
sum += number;
}
Queste istruzioni sono eseguite 20 volte (i = 0, 1, 2, … , 19).
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 7 - 23


Sintassi del Costrutto for
for (<inizio>; <espressione logica>; <incremento>)
<istruzioni>
Inizio
Espressione Logica
Incremento
for (i = 0 ; i < 20 ; i++) {
number = inputBox.getInteger();
sum += number;
}
Istruzioni
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 7 - 24

Flusso di Controllo for



Altri Esempi

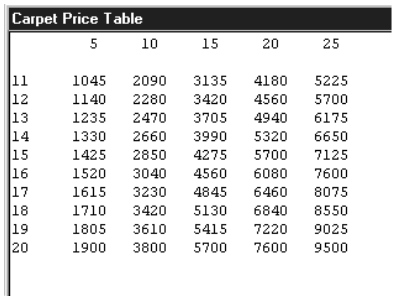
1.
```
for (int i = 0; i < 100; i += 5)
```
i = 0, 5, 10, ... , 95

2.
```
for (int j = 2; j < 40; j *= 2)
```
j = 2, 4, 8, 16, 32

3.
```
for (int k = 100; k > 0; k--) )
```
k = 100, 99, 98, 97, ..., 1

Costrutti for Annidati

- I cicli for annidati sono un costrutto comunemente utilizzato in programmazione.
- Si supponga di voler generare la seguente tabella...



Carpet Price Table

	5	10	15	20	25
11	1045	2090	3135	4180	5225
12	1140	2280	3420	4560	5700
13	1235	2470	3705	4940	6175
14	1330	2660	3990	5320	6650
15	1425	2850	4275	5700	7125
16	1520	3040	4560	6080	7600
17	1615	3230	4845	6460	8075
18	1710	3420	5130	6840	8550
19	1805	3610	5415	7220	9025
20	1900	3800	5700	7600	9500

Generazione di una Tabella

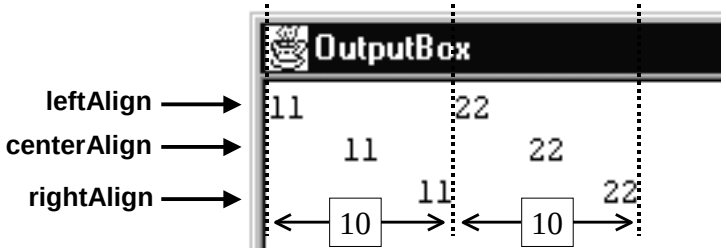
```
int          price;
MainWindow   mainWindow  = new MainWindow();
OutputBox    outputBox   = new OutputBox(mainWindow);

mainWindow.setVisible( true );
outputBox.setTitle("Carpet Price Table");
outputBox.setVisible( true );

for (int width = 11; width <= 20; width++) {
        for (int length = 5; length <= 25; length += 5) {
                price = width * length * 19; //$19 per sq ft.
                outputBox.print("  " + price);
        }

    outputBox.skipLine(1); // un spazio al termine di ogni riga
}
```

ESTERNO

INTERNO

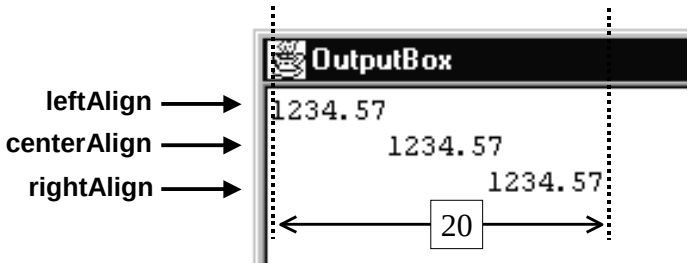
Impaginazione di Numeri Interi

```
int x = 11, y = 22;
outputBox.printLine( Format.leftAlign  ( 10, x ) +
                     Format.leftAlign  ( 10, y )  );
outputBox.printLine( Format.centerAlign( 10, x ) +
                     Format.centerAlign( 10, y )  );
outputBox.printLine( Format.rightAlign ( 10, x ) +
                     Format.rightAlign ( 10, y )  );
```



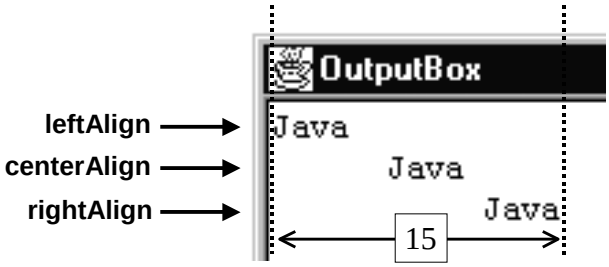
Impaginazione di Numeri Reali

```
double w = 1234.5678;
outputBox.printLine( Format.leftAlign  ( 20, 2, w ));
outputBox.printLine( Format.centerAlign( 20, 2, w ));
outputBox.printLine( Format.rightAlign ( 20, 2, w ));
```



Impaginazione di Testi (Stringhe)

```
String s = “Java”;
outputBox.printLine( Format.leftAlign  ( 15, s ));
outputBox.printLine( Format.centerAlign( 15, s ));
outputBox.printLine( Format.rightAlign ( 15, s ));
```



Metodi nella Classe Format

CLASS: Format		
Class Method	Argument	Description
leftAlign	int, long or int or String	The first argument designates the field width. The second argument is left aligned in the given field. The method returns the formatted value as a String.
leftAlign	int, int, double or float	The first argument designates the field width. The second argument designates the decimal places. The third argument is left aligned in the given field. The method returns the formatted value as a String.
centerAlign	int, long or int or String	Same as the first version of leftAlign, but with the center alignment.
centerAlign	int, int, double or float	Same as the second version of leftAlign, but with the center alignment.
rightAlign	int, long or int or String	Same as the first version of leftAlign, but with the right alignment.
rightAlign	int, int, double or float	Same as the second version of leftAlign, but with the right alignment.

Esempio: Gioco Hi-Lo

- Problema

  Scrivere un'applicazione per il gioco Hi-Lo. Lo scopo del gioco è quello di indovinare un numero segreto generato dal calcolatore nel minor numero di tentativi possibile. Il numero segreto è un intero tra 1 e 100 (estremi inclusi). Quando il giocatore fa un tentativo, il programma risponde HI oppure LO a seconda che il tentativo sia più alto o più basso del numero segreto. Il massimo numero di tentativi è fissato a sei per ogni giocata. Il giocatore può ripetere il gioco quante volte vuole.

- Schema del programma

  do {

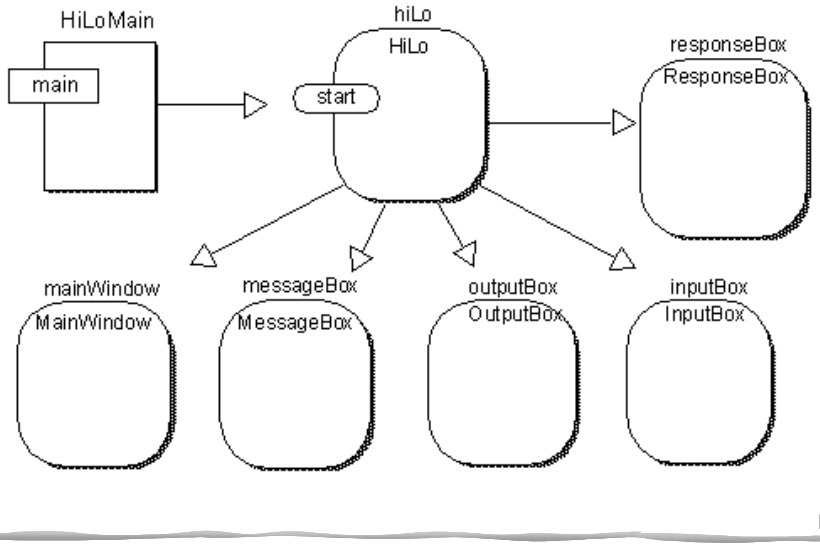
  1. Generare un numero segreto;
  2. Giocare (6 tentativi);

  } while (l'utente desidera continuare a giocare);

HiLo – Progetto

Design Document: HiLo	
Class	**Purpose**
`HiLoMain`	The main class of the program.
`HiLo`	The top-level control object that manages other objects in the program.
`MessageBox`	A MessageBox object is used to show the hint HI or LO after each guess. This class is from javabook.
`InputBox`	An InputBox object is used to get a guess from the player. This class is from javabook.
`OutputBox`	An OutputBox object is used to display the rules of the Hi-Lo game. This class is from javabook.
`ResponseBox`	A ResponseBox object is used to get the user's instruction to play another game or not. This class is from javabook.
`MainWindow`	The main window of the application. This class is from java-book.

HiLo – Diagramma



Gioco HiLo – Passi di Sviluppo

1. Iniziare con uno scheletro del programma. Definire le classi HiLoMain e HiLo.
2. Aggiungere istruzioni alla classe HiLo per giocare usando un numero segreto fittizio.
3. Aggiungere istruzioni alla classe HiLo per generare un numero (pseudo)casuale.
4. Finalizzare il programma nei dettagli.

The End