D04
Materiale Integrativo

Luca Grilli

Programmazione di Interfacce Grafiche e Dispositivi Mobili

A.A. 2020-2021

$$(x, y) \rightarrow x^2 + y^2$$

Cenni alle Lambda Expression in Java

Concetti di base sulla programmazione funzionale (in Java)

Cenni al *lambda calcolo* (*λ-calcolo*)

- In Java 8 è stato inserito il supporto alla ***programmazione funzionale***, e in particolare al cosiddetto ***lambda calcolo***, o in breve ***λ-calcolo***
- L'idea di base della programmazione funzionale è quella di ricondurre una generica computazione alla valutazione di determinate **funzioni matematiche anonime** opportunamente combinate
- Intuitivamente, una ***funzione matematica anonima*** è una funzione descrivibile da una sintassi del tipo $x \rightarrow f(x)$
- Ad esempio, $x \rightarrow x^2$ rappresenta una **funzione quadratica** anonima di una sola variabile
- Invece $(x, y) \rightarrow x^2 + y^2$ rappresenta una funzione anonima di due variabili

λ-calcolo - Funzioni anonime

- Nel λ-calcolo si combinano in diversi modi delle **funzioni anonime**, un'operazione tipica è la composizione di funzioni
- Non è necessario che le funzioni siano identificabili da un nome apposito, ma è sufficiente una sintassi che stabilisca qual è l'input e qual è il tipo di trasformazione da applicare all'input per ottenere l'output
- Esempio: $(x, y) \rightarrow x^2 + y^2$

- Una trattazione anche semplificata della programmazione funzionale va oltre gli scopi di questo insegnamento, ci si limiterà soltanto ad alcuni concetti fondamentali spiegati in modo intuitivo

Lambda Expressions

- In Java 8 (e successive versioni), il supporto alla programmazione funzionale è offerto dalla cosiddette ***Lambda Expressions***
- Una ***lambda expression*** è una **funzione** (in senso matematico) **anonima**, priva cioè di una dichiarazione che le assegni un nome, che permette di scrivere codice funzionale
- Cioè, è possibile passare una funzione ad un'altra funzione (nel senso di composizione di funzioni), restituire funzioni da funzioni e, in generale, combinare due o più funzioni per realizzare una nuova funzione
- Per quanto concerne lo sviluppo di GUI, la principale **utilità** delle lambda expression è quella di poter scrivere un **codice snello** e facile da **manutenere**

Quando utilizzare una lambda expression 1/2

- L'utilizzo delle lambda expression è consigliabile in tutte quelle situazioni in cui la logica dell'applicazione richiede il **passaggio** di un «**comportamento**» ad un **metodo**, piuttosto che il passaggio di dati generici

- Ad esempio, facendo riferimento al codice della classe SwingApplication, in particolare al metodo addActionListener(), ciò che in realtà si desidera passare all'argomento di questo metodo è il «comportamento» che deve essere innescato in risposta all'evento di click del pulsante
- **Se** tale comportamento è **esprimibile** tramite una **funzione anonima**, allora è possibile utilizzare una lambda expression

Quando utilizzare una lambda expression 2/2

- Come **regola** pratica, possiamo dire che una lambda expression è **utilizzabile** tutte quelle volte che
  - viene passato un «**comportamento**» ad un metodo; e
  - tale comportamento è **incapsulato** all'interno di un oggetto di una **classe anonima** che implementa un'***interfaccia funzionale***;
- dove per ***interfaccia funzionale*** si intende un'interfaccia che presenta soltanto **uno ed un solo** metodo astratto

- Si osservi che si tratta esattamente della situazione esaminata nella gestione decentralizzata degli eventi dei pulsanti in SwingApplication

Lambda expression in SwingApplication

- In SwingApplication, il comportamento descritto dal metodo actionPerformed() è **incapsulato** in un **oggetto** di una classe (interna) **anonima** che implementa l'interfaccia ***ActionListener***
- ***ActionListener*** è a tutti gli effetti un'**interfaccia funzionale**, in quanto presenta **un solo metodo** astratto: il metodo ***actionPerformed()***

- Si noti inoltre che il nome del metodo (unico) dell'interfaccia (nel nostro esempio *actionPerformed()*) può anche essere **omesso**, in quanto è **deducibile in modo implicito**

Lambda expression – Sintassi 1/N

- In definitiva, stiamo dicendo che la seguente sintassi «ingombrante» vista per la gestione degli eventi dei pulsanti in SwingApplication

```
jbutton.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      handleBut1();
   }
});

...

jbut2.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      handleBut2();
   }
});
```

Lambda expression – Sintassi 2/N

- È sostituibile dalla seguente, molto più snella, che fa uso delle lambda expression

```
jbutton.addActionListener(
    (ActionEvent e) -> handleBut1()
);

...

jbut2.addActionListener(
    (ActionEvent e) -> handleBut2()
);
```

Lambda expression – Sintassi 3/N

- Essendo implicito anche il **tipo** del **parametro** formale, o i tipi dei **parametri formali**, del metodo dell'interfaccia funzionale, la sintassi può esser ulteriormente snellita

```
jbutton.addActionListener(
   e -> handleBut1()
);

...

jbut2.addActionListener(
   e -> handleBut2()
);
```

Lambda expression – Sintassi 4/N

- Ovviamente il **nome** del **parametro** può essere qualsiasi nome di variabile accettato da Java, non necessariamente quello riportato nella definizione dell'interfaccia funzionale

```
jbutton.addActionListener(
   event -> handleBut1()
);

...

jbut2.addActionListener(
   event -> handleBut2()
);
```

Lambda expression – Sintassi 5/N

- Inoltre, non si è obbligati ad utilizzare dei metodi di appoggio privati, cioè handleBut1() e handleBut2(), ma la loro **implementazione** può essere direttamente inserita nella definizione della lambda expression

```
jbutton.addActionListener(
   event -> {
      this.numClicks++;
      this.jlabel.setText(labelPrefix + this.numClicks);
   }
);

...

jbut2.addActionListener(
   event -> {
      slowMethod(10);
      this.numSlowClicks++;
      this.jlab2.setText(lab2Prefix + this.numSlowClicks);
   }
);
```

The End