Programmazione II

12. Astrazione Iterazione (Implementazione, Specifica, Nested Classes) (PDJ 6.5-)

Implementazione di un Iterator

Implementazione Iterator mediante Nested Classes

L'idea è scrivere la classe che implementa Iterator (Classe contenuta) **all'interno della classe che contiene il dato** che voglio iterare (Classe contenitore)

- Così **ha accesso alla rappresentazione** e quindi al dato da iterare
  - Perché **gli attributi della classe sono visibili da tutto il codice della classe**
  - Devo nascondere la rappresentazione da altri, non da me stesso
- Vedremo le:
  - **static nested classes** - classe contenuta **non associata a una specifica istanza** del contenitore
  - **inner classes** - classe contenuta **associata all'istanza** del contenitore
  - **inner classes locali** - classe contenuta **interna a un blocco** del contenitoro
  - **inner classes anonime** - classe contenuta definita direttamente come istanza nel contenitore

Static Nested Classes

Cosa sono le Static Nested Classes?

Una **Static Nested Class** è una **classe interna** ad un'altra **classe esterna**

- Può accedere **direttamente** alla **rappresentazione** della **classe esterna**
  - Anche agli **attributi privati**
  - **senza usare metodi** creati appositamente
  - Questo perché **fa parte del codice della classe** esterna
- Può **esistere indipendentemente** dalla classe esterna
  - Possibile creare **istanze** della Static Nested Class **indipendenti** dalle istanze della classe esterna
  - La Static Nested Class fa parte della classe esterna per **naming** (come se fosse in un package)
- Per accedere alla rappresentazione di un'istanza della classe esterna:
  - La Static Nested Class deve avere un **riferimento** esplicito all'**istanza** della classe esterna
  - Questo riferimento deve essere passato ad un metodo della Static Nested Class che poi lo userà

Static Nested Classes

Esempio Nested Class

```java
public class OuterClass {
  private int n;

  public OuterClass(int n) {
    this.n = n;
  }

  public static class StaticNestedClass { //DENTRO OuterClass per accedere ai suoi attributi
    OuterClass o; //Necessario riferimento a ISTANZA della OuterClass collegata

    private StaticNestedClass(OuterClass o) {
      this.o = o; //Modo per passare la OuterClass collegata
    }

    public String toString() {
      return "n is: " + this.o.n; //accesso diretto all'attributo n di o
    }
  }

  public static void main(String[] args) {
    OuterClass o = new OuterClass(5);
    OuterClass.StaticNestedClass i = new OuterClass.StaticNestedClass(o);
    System.out.println(i); //toString di i si riferisce a n in o
  }
}
```

Static Nested Classes

Esempio Iterable con Static Nested Class

```java
public class Studente implements Iterable<String> { //ha un iterator()
  private ArrayList<String> corsiStudente;

  public Studente() {
    this.corsiStudente = new ArrayList<String>();
    this.corsiStudente.add("corso1");
    this.corsiStudente.add("corso2");
  }

  @Override
  public Iterator<String> iterator() { //restituisce un Iterator<String>
    return new IteratorCorsi(this);
  }

  public static class IteratorCorsi implements Iterator<String> { //avra' hasNext() e next()
    ...
  }

  public static void main(String[] args) {
    Studente t = new Studente();

    for(String s : t) // essendo Studente Iterable, posso usare foreach
      System.out.println(s);
  }
}
```

Static Nested Classes

Esempio Iterator con Static Nested Class

```
1  public static class IteratorCorsi implements Iterator<String> { //avra' hasNext() e next()
2    private Studente s;
3    private int index; //riferimento all'indice dell'ultimo elemento iterato
4
5    private IteratorCorsi(Studente s) {
6      this.s = s; //nel costruttore si prende il riferimento a un'istanza di Studente
7    }
8
9    @Override //override metodo d'interfaccia
10   public boolean hasNext() {
11     return index < s.corsiStudente.size(); //se ci sono ancora elementi da iterare
12   }
13
14   @Override //override metodo d'interfaccia
15   public String next() {
16     if(!(this.hasNext())) //se chiamo next() e non ci sono altri elementi
17       throw new NoSuchElementException();
18
19     return s.corsiStudente.get(index++); //restituisci elemento e avanza l'indice
20   }
21 }
```

Inner Classes

Cosa sono le Inner Classes?

Una **Inner Class** è una **classe interna ad un'istanza** di una **classe esterna**

- Un'istanza della Inner Class è **dipendente** da un'istanza della classe esterna
  - Le istanze della classe interna devono essere **create dall'istanza della classe esterna**
  - O dentro i **metodi** della classe interna
  - O con la sintassi `` `istanzaClasseEsterna.new ClasseInterna()` ``
- Essendo **parte di un'istanza** della classe esterna, **può accedere direttamente** ai suoi attributi
  - **Non serve passare un riferimento** all'istanza della classe esterna alla classe interna
  - La parola chiave `` `this` `` dentro la classe interna si riferisce agli attributi della classe interna
  - Mentre per la classe esterna si può usare `` `ClasseEsterna.this` `` per disambiguare

Inner Classes

Esempio Inner Class

```
1  public class OuterClass {
2    private int n;
3
4    public OuterClass(int n) {
5      this.n = n;
6    }
7
8    public InnerClass returnInnerClass() { //metodo sara' chiamato su un'istanza di OuterClass
9      return new InnerClass(); //istanza di InnerClass legata a questa istanza di OuterClass
10   }
11
12   public class InnerClass { //ha riferimento alla SUA istanza della OuterClass
13     public String toString() {
14       return "n is: " + n; //accesso alla rep della SUA istanza della OuterClass
15     }
16   }
17
18   public static void main(String[] args) {
19     OuterClass o = new OuterClass(5); //nuova istanza OuterClass
20     InnerClass i = o.returnInnerClass(); //istanza InnerClass creata dentro la OuterClass
21     System.out.println(i); //toString di i si riferisce a n in o
22   }
23 }
```

Inner Classes

Esempio Iterable con Inner Class

```
1  public class Studente implements Iterable<String> { //ha un iterator()
2    private ArrayList<String> corsiStudente;
3
4    public Studente() {
5      this.corsiStudente = new ArrayList<String>();
6      this.corsiStudente.add("corso1");
7      this.corsiStudente.add("corso2");
8    }
9
10   @Override //override di metodo d'interfaccia
11   public Iterator<String> iterator() { //restituisce un Iterator(String)
12     return new IteratorCorsi();
13   }
14
15   class IteratorCorsi implements Iterator<String> { //avra' hasNext() e next()
16     ...
17   }
18
19   public static void main(String[] args) {
20     Studente t = new Studente();
21
22     for(String s : t) // essendo Studente Iterable, posso usare foreach
23       System.out.println(s);
24   }
25 }
```

Inner Classes

Esempio Iterator con Inner Class

```
1  class IteratorCorsi implements Iterator<String> { //avra' hasNext() e next()
2    private int index = 0; //riferimento all'indice dell'ultimo elemento iterato
3
4    @Override //override metodo d'interfaccia
5    public boolean hasNext() { //corsiStudente, ovvero Studente.this.corsiStudente
6      return index < corsiStudente.size(); //se ci sono ancora elementi da iterare
7    }
8
9    @Override //override metodo d'interfaccia
10   public String next() {
11     if(!(this.hasNext())) //se chiamo next() e non ci sono altri elementi
12       throw new NoSuchElementException();
13
14     return corsiStudente.get(index++); //restituisci elemento e avanza l'indice
15   }
16 }
```

Local Inner Classes

Cosa sono le Local Inner Classes?

Una **Local Inner Class** è una Inner Class dichiarata all'**interno di un blocco** della **classe esterna**

- La Local Inner Class è **visibile solo all'interno del blocco** che la contiene
  - Come tale, può essere **istanziata ed utilizzata** solo all'**interno di quel blocco**
  - Ha **accesso alle variabili presenti nel blocco** in cui si trova (**closure**)
  - Queste devono essere **effettivamente** `final` (non possono essere modificate nella inner)
  - Una **sua istanza** non può essere restituita perchè l'**esterno non è a conoscenza della classe**
- Come si può restituire qualcosa di noto all'esterno?
  - Si può restituire come **istanza di superclasse** visibile dall'esterno (es: Object)
  - Si può restituire come **istanza di una** delle sue **Interface** (es: Iterator)
  - Dall'esterno **sarà visibile** e usabile solo il suo **Tipo Apparente** e non il suo Tipo Concreto
  - Il suo Tipo Concreto può avere anche **altri metodi** ma questi saranno **visibili solo nel blocco**

Local Inner Classes

Esempio Local Inner Class

```
1  public class OuterClass {
2    private int n;
3
4    public OuterClass(int n) {
5      this.n = n;
6    }
7
8    public Object returnLocalInnerClass() { //restituire superclasse o un'interfaccia
9      int l = n + 3; //una local inner class vede il contenuto del blocco in cui si trova
10     class LocalInnerClass { //deve essere dichiarata dentro un blocco
11       @Override
12       public String toString() { //visibili fuori solo metodi Override
13         return "n is: " + n + " l is: " + l; //l puo' essere letta ma non modificata
14       }
15     }
16
17     return new LocalInnerClass();
18   }
19
20   public static void main(String[] args) {
21     OuterClass o = new OuterClass(5);
22     Object i = o.returnLocalInnerClass(); //Object perche' LocalInnerClass e' sconosciuta
23     System.out.println(i);
24   }
25 }
```

Local Inner Classes

Esempio Iterable con Local Inner Class

```java
public class Studente implements Iterable<String> { //ha un iterator()
  private ArrayList<String> corsiStudente;

  public Studente() {
    this.corsiStudente = new ArrayList<String>();
    this.corsiStudente.add("corso1");
    this.corsiStudente.add("corso2");
  }

  @Override //override di metodo d'interfaccia
  public Iterator<String> iterator() { //restituisce un Iterator(String)
    class IteratorCorsi implements Iterator<String> { //avra' hasNext() e next()
      ...
    }

    return new IteratorCorsi(); //da fuori figura solo come Iterator<String>
  }

  public static void main(String[] args) {
    Studente t = new Studente();

    for(String s: t) // essendo Studente Iterable, posso usare foreach
      System.out.println(s);
  }
}
```

Local Inner Classes

Esempio Iterator con Local Inner Class

```
1  class IteratorCorsi implements Iterator<String> { //avra' hasNext() e next()
2    private int index = 0; //riferimento all'indice dell'ultimo elemento iterato
3
4    @Override //override metodo d'interfaccia
5    public boolean hasNext() {
6      return index < corsiStudente.size(); //se ci sono ancora elementi da iterare
7    }
8
9    @Override //override metodo d'interfaccia
10   public String next() {
11     if(!(this.hasNext())) //se chiamo next() e non ci sono altri elementi
12       throw new NoSuchElementException();
13
14     return corsiStudente.get(index++); //restituisci elemento e avanza l'indice
15   }
16 }
```

Anonymous Classes

Cosa sono le Anonymous Classes?

Una **Anonymous Class** è una Local Inner Class **senza un nome Classe** proprio

- Si deve **dichiarare ed inizializzare in una singola espressione**
  - Es: nell'assegnamento ad una variabile, come parametro, come oggetto...
  - come local inner, **vede le variabili del blocco** in cui si trova (**closure**)
- **Deve avere una superclasse o un'interfaccia** che sarà il suo **Tipo Apparente**
  - Si può **assegnare** l'istanza a una variabile di quel **Tipo Apparente** e restituirla
  - Il **Tipo Concreto non è visibile** neanche nel blocco nel quale è creata la classe anonima
  - Gli **altri metodi** saranno usabili **solo internamente alla classe anonima**
- Limiti delle Classi Anonime
  - Può esserci **una sola istanza** di una classe anonima, può estendere **una sola superclasse o interfaccia**
  - La classe anonima **non si può estendere**, **non ha variabili di classe** e **non ha costruttori** (espliciti)
  - La **closure** può sopperire alla **mancanza del costruttore** (es: posso fare li il sort di una lista da iterare)

Anonymous Classes

Esempio Anonymous Class

```
1  public class OuterClass {
2    private int n;
3
4    public OuterClass(int n) {
5      this.n = n;
6    }
7
8    public Object returnAnonymousClass() {
9      int l = n + 3; //una anonymous class vede il contenuto del blocco in cui si trova
10
11     return new Object() { //posso restituire una superclasse o un'interfaccia
12       public String toString() {
13         return "n is: " + n + " l is: " + l; //l puo' essere letta ma non modificata
14       }
15     };
16   }
17
18   public static void main(String[] args) {
19     OuterClass o = new OuterClass(5);
20     Object i = o.returnAnonymousClass();
21     System.out.println(i);
22   }
23 }
```

Anonymous Classes

Esempio Iterable e Iterator con Anonymous Class

```
1  public class Studente implements Iterable<String> { //ha un iterator()
2    private ArrayList<String> corsiStudente;
3
4    public Studente() {
5      this.corsiStudente = new ArrayList<String>();
6      this.corsiStudente.add("corso1");
7      this.corsiStudente.add("corso2");
8    }
9
10   @Override //override di metodo d'interfaccia
11   public Iterator<String> iterator() { //restituisce un Iterator(String)
12     return new Iterator<String>() { //avra' hasNext() e next()
13       private int index = 0; //riferimento all'indice dell'ultimo elemento iterato
14
15       @Override //override metodo d'interfaccia
16       public boolean hasNext() {
17         return index < corsiStudente.size(); //se ci sono ancora elementi da iterare
18       }
19
20       @Override //override metodo d'interfaccia
21       public String next() {
22         if(!(this.hasNext())) //se chiamo next() e non ci sono altri elementi
23           throw new NoSuchElementException();
24
25         return corsiStudente.get(index++); //restituisci elemento e avanza l'indice
26       }
27     };
28   }
29
30   public static void main(String[] args) {
31     Studente t = new Studente();
32
33     for(String s : t) // essendo Studente Iterable, posso usare foreach
34       System.out.println(s);
35   }
36 }
```

AF & RI

Funzione di Astrazione di un Iteratore

AF associa le istanze di una classe all'oggetto astratto che rappresentano

- Le istanze di un iteratore sono **tutti i possibili stati di iterazione**
  - Cioè in un dato momento cosa è stato già iterato e cosa manca ancora da iterare
  - Questo in molti casi è difficile (o impossibile nelle iterazioni infinite) da implementare
  - Dovrei fare un toString() che stampa tutti gli elementi e dov'è il puntatore adesso

Invariante di Rappresentazione di un Iteratore

RI di un iteratore è definita **in termini** della rappresentazione **del dato iterato**

- Quindi dovrà considerare sia lo **stato della classe base**
- Sia lo stato dell'iteratore stesso (es: lista dei prossimi elementi da iterare)
  - Come per la Funzione di Astrazione questo potrebbe essere difficile o impossibile da calcolare
  - Per questo Liskov non chiede di implementare tali funzioni
  - È comunque utile pensarle per validare la correttezza della rappresentazione

Programmazione II

12. Astrazione Iterazione (Implementazione, Specifica, Nested Classes) (PDJ 6.5-)

Dragan Ahmetovic