Nascondere dettagli implementativi

Partiamo dal progetto realizzato AtletiGUI (http://amplio00.belluzzi.scuole.bo.it/mod/resource/view.php?id=76549)

In questo pregetto la classe model SquadraDiAtleti ha il compito di gestire i dati degli atleti di una squadra. Sappiamo che internamente gli atleti sono organizzati in un array, e questo traspare anche al di fuori perché i due metodi

```
public void setAtleta(int i, Atleta a){...}
public Atleta getAtleta(int i){...}
```

richiedono l'esplicitazione dell'indice del dato a cui riferirsi.

Analizziamo il contesto d'uso del metodo getter possiamo osservare che viene utilizzato durante una iterazione su tutti i dati in squadra:

```
//scrivi su file tutti gli atleti della squadra
for (int i=0;i<atleti.getNumeroAtleti();i++){
    scrittore.setPettorale(atleti.getAtleta(i).getPettorale());

scrittore.setCognomeNome(atleti.getAtleta(i).getCognomeNome());
    scrittore.setPeso(atleti.getAtleta(i).getPeso());
}
```

Possiamo raggiungere allo stesso obiettivo senza esplicitare l'uso di un indice se disponiamo di un oggetto che permette di “scorrere” (iterare) lungo tutti i dati garantendoci di poter passare ciclicamente su di tutti senza saltarne alcune né mai ripeterne.

Un oggetto di questo tipo prende il nome di iteratore e permette di raggiungere l'obiettivo prefissato se dispone di due metodi:

```
public boolean hasNext()
```

che ritorna true se sono disponibili altri dati, e

```
public Atleta next()
```

che restituisce il 'prossimo' atleta.

Ecco una possibile implementazione:

```
public class SquadraIterator {
  private SquadraDiAtleti squadra;
  int indice=0;

  public SquadraIterator(SquadraDiAtleti squadra) {
    this.squadra = squadra;
  }



  public boolean hasNext(){
```

```
    return indice<squadra.getNumeroAtleti();
  }
  public Atleta next(){
    if (hasNext()){
      Atleta a=squadra.getAtleta(indice);
      indice++;
      return a;
    }
    else return null;
  }
}
```

e quindi il segmento di codice del precedente esempio diventerebbe:

```
//scrivi su file tutti gli atleti della squadra
SquadraIterator iteratore= new SquadraIterator(atleti);
while (iteratore.hasNext()){
  Atleta a=iteratore.next();
  scrittore.setPettorale(a.getPettorale());
  scrittore.setCognomeNome(a.getCognomeNome());
  scrittore.setPeso(a.getPeso());
}
```

come si vede non è stato più necessario esplicitare l'uso dell'indice.

Rimane il fatto che fino a che la classe SquadraDiAtleti rende pubblico l'uso dell'indice non avremo reso totalmente invisibili gli elementi implementativi, ma d'altra parte la classe SquadraIterator deve poter accedere all'indice per poter svolgere il suo compito. Se potessimo rendere la classe SquadraIterator un elemento interno alla classe SquadraDiAtleti allora non sarebbe più necessario rendere visibile l'uso dell'indice.

Questo è possibile definendo SquadraIterator come **inner class** della classe SquadraDiAtleti, cioè scrivendo fisicamente la sua definizione all'interno della classe squadra di Atleti.

```
public class SquadraDiAtleti {
  private Atleta[] squadra;

//...

  /**
   * getter dell'atleta di indice i
   * @param i indice, non viene testato il fuori range
   * @return atleta
   */
  private Atleta getAtleta(int i){
    return squadra[i];
  }
  // ...
```

```
  public class SquadraIterator {
    private SquadraDiAtleti squadra;
    int indice=0;

    public SquadraIterator(SquadraDiAtleti squadra) {
      this.squadra = squadra;
    }



    public boolean hasNext(){
      return indice<squadra.getNumeroAtleti();
    }
    public Atleta next(){
      if (hasNext()){
        Atleta a=squadra.getAtleta(indice);
        indice++;
        return a;
      }
      else return null;
    }
  }

}
```

Si noti che il metodo getAtleta(indice) è divento privato.

La sintassi d'uso di questa classe diventa:

```
//scrivi su file tutti gli atleti della squadra
SquadraDiAtleti.SquadraIterator iteratore= atleti.new
SquadraIterator(atleti);
while (iteratore.hasNext()){
  Atleta a=iteratore.next();
  scrittore.setPettorale(a.getPettorale());
  scrittore.setCognomeNome(a.getCognomeNome());
  scrittore.setPeso(a.getPeso());
}
```

Si osservi ora che in realtà non c'è più bisogno che l'iteratore abbia in sé un riferimento all'oggetto SquadraDiAtleti; dal momento che ne fa parte internametne può accedere direttamente al suo array di dati:

```
public class SquadraDiAtleti {
  private Atleta[] squadra;

//...
```

```
  public class SquadraIterator {
    int indice=0;

    public boolean hasNext(){
//       return indice<squadra.getNumeroAtleti();
      return indice<squadra.length;
    }
    public Atleta next(){
      if (hasNext()){
        Atleta a=squadra[indice];
        indice++;
        return a;
      }
      else return null;
    }
  }
}
```

si noti che ora il metodo getAtleta può essere rimosso.

Per rimuovere consideriamo che per la funzione di modifica del dato abbiamo già visto che non serve, rimane l'uso che ne abbiamo fatto al momento di caricare i dati letti da file:

```
  public static SquadraDiAtleti popolaAtletiViaFile(File f,
AtletiConsoleView console) throws Exception {
    SquadraDiAtleti atleti;

    // viene aperto il file e determinato numAtleti

    atleti=new SquadraDiAtleti(numAtleti);
    //controllo la lettura con ciclo sul numero di atleti
    for (int i =0; i< numAtleti ; i++ ){
      // ...
      //leggo da file pettorale, cn, peso
      // ...
      atleti.setAtleta(i, new Atleta(pettorale,cn,peso));
    }
    //lettura completa
    return atleti;
  }
```

Qui di fatto si stanno di volta in volta aggiungendo nuovi elementi, quindi esplicitiamo questo con un metodo aggiungi, questo comporta la necessità di avere un contatore di dati effettivamente presenti nell'array.

```
/**
 * Gestione dei dati di una squadra di atleti
 * @author Gianni
 */
public class SquadraDiAtleti {
  private Atleta[] squadra;
  private int numeroAtleti;
```

```
  /**
   * Costruttore, istanzia la squadra su un numero fornito di
atleti
   * @param numAtleti il numero di altleti su cui dimensionare
   */
  public SquadraDiAtleti(int numAtleti) {
    squadra = new Atleta[numAtleti];
    numeroAtleti=0;
  }

  /**
   * Costruttore, referenzia un array di atleti passato
   * @param squadra array da referenziare
   */
  public SquadraDiAtleti(Atleta[] squadra) {
    this.squadra=squadra;
    numeroAtleti=squadra.length;
  }

  /**
   * Conversione a string dei dati di un atleta
   * @return dati dell'atleta
   */
  @Override
  public String toString() {
    String out="";

    out += "SquadraDiAtleti{" + "\nlunghezza=" + squadra.length;
    for (int i = 0; i<squadra.length ; i++){
      out += "\natleta["+i+"]="+squadra[i]+";";
    }
    out+=  "\n}\n";
    return out;
  }

  /**
   * Aggiunge un atleta
   * @param a il dato da agggiungere
   * @throws Exception spazio esaurito
   */
  public void aggiungi(Atleta a)throws Exception {
    if (numeroAtleti<squadra.length){
      squadra[numeroAtleti]=a;
      numeroAtleti++;
    }
    else {
      throw new Exception("Spazio esaurito in squadra");
    }
  }
```

```
  /**
   * Ricerca nei dati l'atleta corrispondente al numero di
pettorale passato
   * @param pettorale da ricercare
   * @return atleta trovato
   * @throws Exception non esiste il pettorale ricercato
   */
  public Atleta ricercaPettorale(int pettorale) throws Exception{
    for (int i = 0; i<squadra.length ; i++){
      if (squadra[i].getPettorale()==pettorale){
        return squadra[i];
      }
    }
    throw new Exception();
  }

  /**
   * Il numero dei dati effettivamente presenti
   * @return
   */
  public int getNumeroAtleti(){
      return numeroAtleti;
  }

  /**
   * Classe iteratore sulla squadra
   */
  public class SquadraIterator {
    int indice=0;

    /**
     * ritorna tru ese il prossimo next() avrà successo
     * @return
     */
    public boolean hasNext(){
      return indice<numeroAtleti;
    }

    /**
     * ritorna il prossimo dato
     * @return
     */
    public Atleta next(){
      if (hasNext()){
        Atleta a=squadra[indice];
        indice++;
        return a;
      }
      else return null;
    }
  }
```

```
}
```

Si noti l'eccezione prevista nel metodo aggiungi quando l'array si satura.

Il metodo di caricamento dati da file diventa:

```
  public static SquadraDiAtleti popolaAtletiViaFile(File f,
AtletiConsoleView console) throws Exception {
    SquadraDiAtleti atleti;

    atleti=new SquadraDiAtleti(numAtleti);
    //leggo i dati da file e memorizzo in atleti
    while (lettoreAtleti.ancoraDati()){
      // ...
      //leggo da file pettorale, cn, peso
      //...
      //dati di atleta completi, istanzio oggetto e lo aggiungo in
squadra
      atleti.aggiungi(new Atleta(pettorale,cn,peso));//l'eventuale
eccezione viene portata al livello superiore
    }
    //lettura completa
    return atleti;
  }
```

cc