Ingegneria del Software

Esercitazione 4

Cron

Si progetti un package che offra un "demone temporale" simile a cron di Unix

***Specifiche**:*

- L'utente del package deve poter creare un demone, registrare presso di lui una serie di coppie <*orario, azione da compiere*>
- Il demone temporale, una volta avviato, deve eseguire le azioni registrate all'orario prestabilito.
- Si supponga che non si possano registrare più di 10 azioni, che ogni azione debba venir eseguita una volta soltanto e che una volta eseguite tutte le azioni cron termini la sua esecuzione.
- Si può interpretare l'orario di esecuzione come "orario indicativo": viene garantito che l'azione viene eseguita *dopo* l'orario specificato

Exceptions

Ask for forgiveness

```
try{
    set.add(new Complex(1.0,1.0));
}catch(FullStackException e){
    System.err.println("Stack is full");
}
```

Ask for permission

```
if (!set.isFull()) {
    set.add(new Complex(1.0, 1.0));
}
```

Stack with Exception (I)

Eccezioni gestite a compile time (checked)

```
public class OutOfDataException extends Exception {}


public class SafeStack extends Stack {

    public int safePop() throws OutOfDataException  {
        if (cur > 0) {
            cur--;
            return data[cur];
        }
        else throw new OutOfElementException();
    }
}
```

Stack with Exception (II)

Eccezioni gestite solamente a runtime

```
public class OutOfDataException extends RuntimeException {}


public class SafeStack extends Stack {

    public int safePop() {
        if (cur > 0) {
            cur--;
            return data[cur];
        }
        else throw new OutOfDataException();
    }
}
```

Stack with Exception (III)

Aggiungere a SafeStack un metodo safePush che gestisca i casi limite dell'inserimento

StringBuffer vs String

Testare la differenza tra la costruzione di stringhe con la classe String e con la classe StringBuffer.

Input/Output

Scrivere un programma che legga da tastiera una frase e una parola, queste devono essere passate ad un metodo statico e che restituisca il numero di occorrenze della parola nella frase.

Collections

Parte I

JCF: Collections in Java

Collezionare oggetti

- **Problema**: raggruppare un insieme di oggetti insieme e accedere ad essi secondo regole particolari (per esempio una Coda).
  - Spesso l'utilizzo degli array non è sufficiente

1. Soluzione 1: Realizzare una propria classe che, utilizzando internamente gli array, fornisce i metodi di accesso opportuni
2. Soluzione 2: Utilizzare classi già pronte fornite da Java, scegliendo quella più opportuna ai propri bisogni

- Java fornisce un insieme molto ampio di classi (concrete) in grado di collezionare oggetti fornendo un interfacce (estese dalle proprie classi) relative a Pile, Code, Insiemi, Liste, Mappe, Insiemi ordinati ecc …
  - **(JCF) Java Collections Framework**

(JCF) Java Collections Framework/1

- Collection:
  - List
    - **ArrayList**
    - **LinkedList**
    - **Vector**
  - Set
    - SortedSet
      - **TreeSet**
    - **HashSet**
    - **LinkedHashSet**
- Altre interfacce disponibili
  - Queue,Dequ, Stack, Map, SortedMap …

(JCF) Java Collections Framework/2

- **Collection**
  - Group of objects, known as its elements. Some collections allow duplicate elements and others do not. Some are ordered and others unordered.
    - ***boolean add(Object e)***
    - ***void clear()***
    - ***boolean contains(Object o)***
    - ***Iterator iterator()***
    - ***boolean remove(Object o) int size()***
- **List**
  - An ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted
    - ***E get(int index)***
    - ***E set(int index, E element)***
- **Set**
  - A collection that contains no duplicate elements
- **SortedSet**
  - A collection that contains sorted elements

Tipi degli oggetti nelle collezioni

- Tutti gli oggetti in java estendono da Object
  - E' corretto scrivere: **`Object o=new Integer(10);`**
- Le collection di java gestiscono elementi di tipo Object
  - Esse possono contenere elementi di tipo object
    - quindi un qualunque oggetto java può essere aggiunto in una collezione
    - NB: Gli oggetti vengono ritornati come Object e non del loro tipo specifico

```
ArrayList a=new ArrayList();
a.add(new Integer(10));//aggiungo un intero
Object elem=a.get(0); //oggetto di tipo Object
```

  - Per ottenere il tipo originario è necessario il cast esplicito

```
Integer i=(Integer) a.get(0);//cast esplicito
```

ArrayList

public class ArrayList extends …

- **boolean add(Object e)**
  - Appends the specified element to the end of this list.
- **void add(int index, Object element)**
  - Inserts the specified element at the specified position in this list.
- **Object get(int index)**
  - Returns the element at the specified position in this list.
- **Object set(int index, Object element)**
  - Replaces the element at the specified position in this list with the specified element.
- **void clear()**
  - Removes all of the elements from this list.
- **Iterator iterator()**
  - Returns an iterator over the elements in this list in proper sequence.
- **int size()**
  - Returns the number of elements in this list.

Esercizio 1

Scrivere un metodo java

che somma gli elementi del vettore di interi.

- Somma: 22

Suggerimenti: utilizzare i metodi get per ottenere gli elementi del vettore e size per conoscere il numero di elementi totali.

- Usando il cast si possono ottenere gli oggetti del tipo opportuno
- Il metodo intValue() della classe Integer permette di ricavare il tipo int

Esercizio 1 - Soluzione

```
public static int somma(ArrayList a){
   int somma=0;
   for (int i=0;i<a.size();i++){
      Integer elem=(Integer)a.get(i)
      somma=somma+elem.intValue();
              //tipo primitivo
   }
   return somma;
}
```

Iterator

- Modo “universale” per scorrere collezioni di elementi, indipendentemente dalla particolare disposizione degli elementi
  - Il metodo **`Iterator iterator()`** è disponibile in tutte classi che estendono da Collection

- **`Iterator`**
  - **`boolean hasNext()`**
    - `Returns true if the iteration has more elements.`
  - **`Object next()`**
    - `Returns the next element in the iteration.`
  - **`void remove()`**
    - `Removes from the underlying collection the last element returned by the iterator.`

Esercizio 2

Scrivere un metodo java

```
int somma(ArrayList a)
```

che sommi gli elementi del vettore di interi utilizzando gli iteratori.

Esercizio 2 - Soluzione

```
public static int somma(ArrayList a){
   int somma=0;
   Iterator i=a.iterator();
   while (i.hasNext()){
      Integer val=(Integer)i.next();
      somma=somma+val.intValue();
   }
   return somma;
}
```

Collection

public interface Collection

- **boolean add(Object e)**
  - Appends the specified element to the end of this list.
- **void add(int index, Object element)**
  - Inserts the specified element at the specified position in this list.
- **void clear()**
  - Removes all of the elements from this list.
- **Iterator iterator()**
  - Returns an iterator over the elements in this list in proper sequence.
- **int size()**
  - Returns the number of elements in this list.
- **boolean isEmpty()**
  - Returns true if this collection contains no elements.
- … ***altri metodi***

List

public interface List extends Collection

- ***Ereditate da Collection***
  - *boolean add(Object e)*
  - *void add(int index, Object element)*
  - *void clear()*
  - *Iterator iterator()*
  - *int size()*
  - *boolean isEmpty()*
- **E get(int index)**
  - Returns the element at the specified position in this list.
- **E remove(int index)**
  - Removes the element at the specified position in this list (optional operation).
- **E set(int index, E element)**
  - Replaces the element at the specified position in this list with the specified element (optional operation).
- … ***altri metodi***

ArrayList

public class ArrayList …**implements List**

- ***Ereditate da List***
  - *boolean add(Object e)*
  - *void add(int index, Object element)*
  - *void clear()*
  - *Iterator iterator()*
  - *int size()*
  - *boolean isEmpty()*
  - E get(int index)
  - E remove(int index)
  - E set(int index, E element)
- … ***altri metodi***

Vector

`public class Vector` **`…implements List`**

- ***Ereditate da List***
  - *`boolean add(Object e)`*
  - *`void add(int index, Object element)`*
  - *`void clear()`*
  - *`Iterator iterator()`*
  - *`int size()`*
  - *`boolean isEmpty()`*
  - `E get(int index)`
  - `E remove(int index)`
  - `E set(int index, E element)`
- *…* ***altri metodi***

- Analogo ad ArrayList ma l'implementazione è thread-safe ed è in genere più lenta

LinkedList

public class LinkedList …**implements List**

- ***Ereditate da List***
  - *boolean add(Object e)*
  - *void add(int index, Object element)*
  - *void clear()*
  - *Iterator iterator()*
  - *int size()*
  - *boolean isEmpty()*
  - E get(int index)
  - E remove(int index)
  - E set(int index, E element)
- **E getFirst()**
  - Returns the first element in this list.
- **E getLast()**
  - Returns the last element in this list.
- … ***altri metodi***

- NB: L'implementazione di get e set hanno costo O(n)

Possibili problemi con le collection

- La mancanza di un controllo sui tipi porta ad alcuni problemi:
  - Necessità di ricorrere al cast degli elementi anche quando il tipo di elementi è noto

```
ArrayList a=new ArrayList();
a.add(new Integer(10));
...
Integer i=(Integer)a.get(0);
```

  - Possibili cast degli elementi a tipi non corretti

```
ArrayList a=new ArrayList();
a.add(new Integer(10));
...
String i=(String)a.get(0);//eccezione!!!
```

Possibili problemi con le collection/2

- Nessun controllo sui tipi di dati inseriti all'interno di un vettore

```
ArrayList a=new ArrayList();
a.add(new Integer(10));
...
a.add(new String("apple"));//?
Integer i=(Integer)a.get(0);//OK
Integer j=(Integer)a.get(1);//cast exception
```

- Poca chiarezza sul tipo di dati trattati
  - `public static ArrayList calcolaQualcosa()` //che tipi contiene?

NB: In tutti i casi, il codice risulta sintatticamente corretto e non viene segnalato alcun errore dal compilatore. L'errore viene scoperto solo a Runtime!

Parte II

Generics

Generics /1

- Programmazione generica: creazione di costrutti di programmazione che possano essere utilizzati con tipi di dati diversi.
  - In Java si può raggiungere l'obiettivo della programmazione generica usando l'ereditarietà oppure le variabili di tipo.
  - Esempio: ArrayList di Java → **ArrayList<String>**
- **Classe generica**: è stata dichiarata usando una variabile di tipo **E**. La variabile di tipo rappresenta il tipo degli elementi:

```
public class ArrayList<E>
  // si può usare "ElementType" invece di E
  {
     public ArrayList() { . . . }
     public void add(E element) { . . . }
     . . .
  }
```

Generics /2

- Le variabili di tipo possono essere sostituite, all'atto della creazione di esemplari, con nomi di classe o di interfacce
  - **`ArrayList<String>`**
    `ArrayList<MyClass>`
- Non si può usare come sostituto uno degli otto tipi di dati primitivi
  - **`ArrayList<double> // Sbagliato!`**
  - Usate un esemplare di ArrayList<Double>

Generics /3

- Il tipo di dato che indicate va a sostituire la variabile di tipo utilizzata nella definizione dell'interfaccia o della classe generica
- Esempio: nel metodo **`add`** di un oggetto di tipo **`ArrayList<String>`** la variabile di tipo E viene sostituita dal tipo String
  - **`public void add(String element)`**

Generics /4

- Le variabili di tipo rendono più sicuro e di più facile comprensione il codice generico.
  - E' impossibile aggiungere un oggetto di tipo **`Integer`** ad un esemplare di **`ArrayList<String>`**
  - È invece possibile aggiungere un oggetto di tipo **`Integer`** ad un esemplare di **`ArrayList`** (senza generics) che sia stato creato con l'intenzione di usarlo per contenere stringhe

Generics /5

```
• ArrayList<String> a1 = new ArrayList<String>();
• Integer numero=new Integer(30);

// Dovrebbe contenere oggetti di tipo String
• ArrayList a2 = new ArrayList();

// errore di compilazione
• a1.add(numero);

// errore non individuato dal compilatore
• a2.add(numero);

//errore di esecuzione
• String account = (String) a2.get(0);
```

ArrayList<E> e Iterator<E>

- **ArrayList<E>**
  - **boolean add(E e)**
  - **void add(int index, E element)**
  - **E get(int index)**
  - **E set(int index, E element)**
  - **void clear()**
  - **Iterator<E> iterator()**
  - **int size()**

- **Iterator<E>**
  - **boolean hasNext()**
  - **E next()**
  - **void remove()**

Esercizio 3

Scrivere un metodo java
`int somma(ArrayList<Integer> a)`
che somma gli elementi del vettore di interi.

- Somma: 22

Esercizio 3 - Soluzione

```
public static int somma(ArrayList<Integer> a){
   int somma=0;
   for (int i=0;i<a.size();i++){
      Integer elem=a.get(i)
      somma=somma+elem.intValue();
   }
   return somma;
}
```

Esercizio

- Data la porzione di codice:

```
ArrayList<String> s=new ArrayList<String>();

ArrayList p=new ArrayList ();
```

Indicare quali, tra le istruzioni seguenti, genera un errore in compilazione o in esecuzione (eccezione):

```
1.  s.add("apple");
2.  s.add(new Integer(10));
3.  String q=s.get(0);
4.  Object o=s.get(0);
5.  Integer i=s.get(0);
6.  s.add(s.get(0));

7.  p.add("apple");
8.  p.add(new Integer(10));
9.  String q=p.get(0);
10. Object o=p.get(0);
11. Integer i=p.get(0);
12. p.add(p.get(0));
```

Esercizi

Stack

Implementare la classe Stack con i Generics

Stack (II)

Aggiungere un Iteratore alla classe Stack