Programmazione Orientata agli Oggetti in Linguaggio Java

Tecniche di Programmazione: Clonazione e Serializzazione

versione 1.0

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Clonazione
  - ⇨Clonazione Superficiale
  - ⇨Clonazione Profonda
- Serializzazione
  - ⇨Dettagli sulla Serializzazione

Clonazione

- Un problema frequente
  - ⇨creare copie degli oggetti
  - ⇨ovvero “clonare” gli oggetti
- Clonare un oggetto
  - ⇨creare un oggetto equivalente rispetto a quello originale
  - ⇨ma con un OID diverso

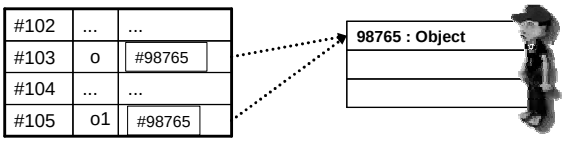
Clonazione

>> it.unibas.appuntamenti2

- Un esempio
  - ⇨duplicare un appuntamento creandone una o più copie in giorni diversi
- Appuntamenti 2
  - ⇨una variante della versione originale
  - ⇨basata sull’uso di generici
  - ⇨e che aggiunge la funzionalità di clonazione

Clonazione

- Attenzione alla differenza
  - creare una copia del riferimento non vuol dire creare una copia dell'oggetto
  - Object o = new Object();
  - Object o1 = o; // nessuna clonazione

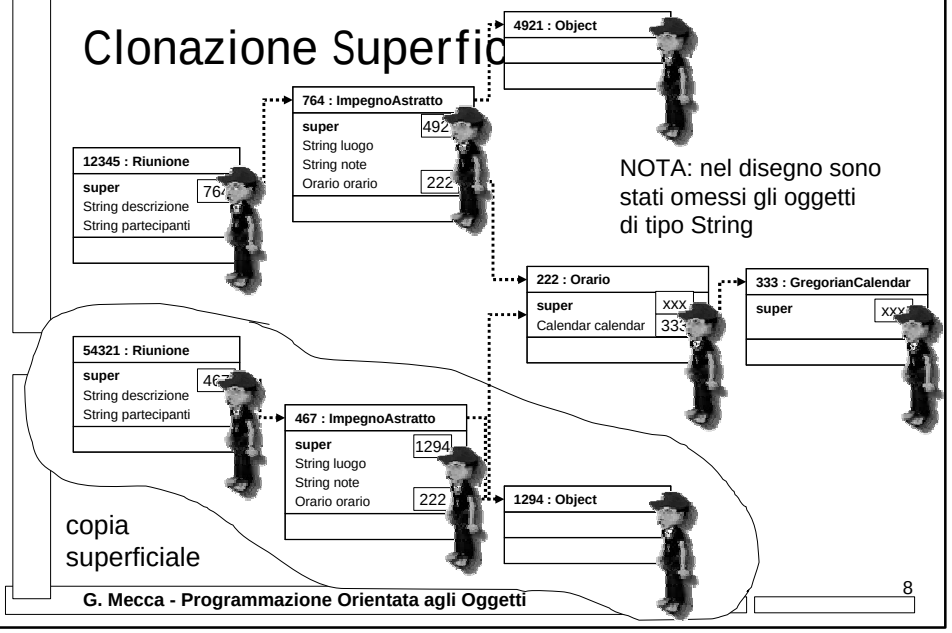


Clonazione

- In effetti
  - il problema della relazione tra copie e riferimenti è molto più complicato di così
  - tanto che esistono due strategie di clonazione
- Strategie di clonazione
  - clonazione superficiale
  - clonazione profonda

Clonazione Superficiale

- Clonazione superficiale (“shallow copy”)
  - ⇨consiste nel creare una copia dell’oggetto principale
  - ⇨ma senza clonare gli oggetti in associazione con quello principale
  - ⇨in altri termini, l’oggetto originale e la copia condividono i riferimenti e quindi collaborano con gli stessi oggetti



Clonazione Superficiale

- Realizzare la clonazione superficiale
  - è una funzionalità fornita dalla piattaforma
  - attraverso il metodo Object clone() ereditato da Object
  - crea una copia bit a bit nello heap dell'oggetto originale
- Ma...
  - ci sono una serie di sottigliezze che bisogna tenere in considerazione

Clonazione Superficiale

- I sottigliezza
  - il metodo clone() è un metodo protetto
  - di conseguenza è visibile nel codice delle sottoclassi, me non nel codice del client
  - per rendere gli oggetti di una classe clonabili utilizzando il metodo clone() bisogna ridefinirlo nella sottoclasse come pubblico
  - chiamando nel corpo super.clone() per sfruttarne le funzionalità

Clonazione Superficiale

- II sottigliezza
  - ⇨il metodo Object.clone() lancia un'eccezione controllata CloneNotSupportedException

```
// in ImpegnoAstratto.java
public Object clone() {
    try {
        return super.clone();
    } catch (CloneNotSupportedException e) {
        return null;
    }
}
```


Clonazione Superficiale

```
public interface Cloneable {}
```

- III sottigliezza
  - ⇨non basta rendere pubblico clone(), ma bisogna anche implementare l'interfaccia java.lang.Cloneable
- Attenzione
  - ⇨a differenza di quanto si può pensare, Cloneable NON contiene il metodo clone() (che appartiene all'interfaccia di tutti gli oggetti perchè è ereditato)

Clonazione Superficiale

- Viceversa
  - ⇨l'interfaccia Cloneable è un'interfaccia vuota
  - ⇨detta anche "tagging interface"
- Tagging interface
  - ⇨interfaccia che serve esclusivamente ad attribuire ad un oggetto un tipo
  - ⇨verificabile usando la riflessione
  - ⇨il metodo clone() di Object verifica che l'oggetto da clonare sia un Cloneable, altrimenti lancia CloneNotSupportedException

Clonazione Superficiale

- Riassumendo
  - ⇨perchè i suoi oggetti siano clonabili usando clone() una classe deve ridefinire clone() rendendolo pubblico
  - ⇨deve catturare l'eventuale CloneNotSupportedException lanciata da super.clone()
  - ⇨e dichiarare di implementare java.lang.Cloneable

Clonazione Superficiale

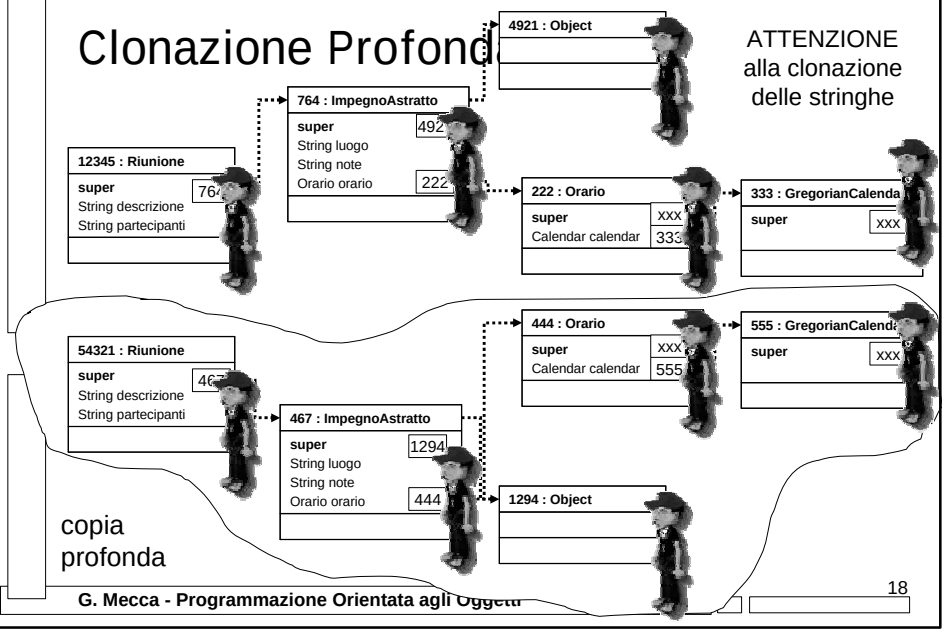
- Il perchè di tutte queste restrizioni
  - ⇨il metodo clone() è un metodo pericoloso
  - ⇨non è opportuno poter clonare tutti gli oggetti
  - ⇨di conseguenza è stato reso protetto
  - ⇨l'utente deve esplicitamente ridefinirlo come pubblico
  - ⇨e, per garantire che non sia stato ridefinito per errore, in aggiunta implementare un'interfaccia specifica

Clonazione Superficiale

- Limiti della clonazione superficiale
  - ⇨se cambio una delle copie (es: modifico l'orario) la modifica ha effetto anche su tutte le altre copie
  - ⇨in alcuni casi questo è accettabile, in altri casi no
  - ⇨quando questo non è accettabile, è necessario sviluppare una forma di clonazione profonda

Clonazione Profonda

- Clonazione profonda (“deep copy”)
  - ⇨consiste nel clonare l’oggetto originale
  - ⇨e ricorsivamente clonare profondamente tutti gli oggetti con cui è in associazione
- In altri termini
  - ⇨clona l’intera rete di oggetti raggiungibili a partire dall’oggetto originale



Clonazione Profonda

- Nel caso di appuntamenti
  - ⇨è indispensabile la clonazione profonda
  - ⇨per rendere i due appuntamenti realmente indipendenti
- Strategia
  - ⇨ridefinire clone() come pubblico
  - ⇨chiamare super.clone() per effettuare la clonazione superficiale
  - ⇨clonare l'orario e correggere il riferimento

```
package it.unibas.appuntamenti2.modello;

public abstract class ImpegnoAstratto implements Impegno, Cloneable {

    protected String luogo;
    protected String note;
    protected Orario orario;

    public Object clone() {
        Object clone = null;
        try {
            clone = super.clone();
            clone.orario = (Orario)this.orario.clone();
        } catch (CloneNotSupportedException e) {
            System.err.println("Errore: oggetto non clonabile " + e);
        }
        return clone;
    }
    ...
}
```

Clonazione Profonda

- Perchè il tutto funzioni
  - ⇨orario deve essere clonabile (problema risolvibile perchè è una classe dell'applicazione)
  - ⇨ma soprattutto deve essere clonabile in modo profondo
  - ⇨questo richiede che sia clonabile il calendario sottostante
  - ⇨fortunatamente la maggior parte delle classi delle APU sono clonabili

```
package it.unibas.appuntamenti2.modello;

import java.util.GregorianCalendar;

public class Orario implements Cloneable {

    private GregorianCalendar calendar;

    public Object clone() {
        Object clone = null;
        try {
            clone = super.clone();
            clone.calendar = (GregorianCalendar)this.calendar.clone();
        } catch (CloneNotSupportedException e) {
            System.err.println("Errore: oggetto non clonabile " + e);
        }
        return clone;
    }
    ...
}
```

Serializzazione

- Un modo alternativo
  - ⇨per clonare in modo profondo un oggetto
  - ⇨salvarlo su disco e poi ricaricarlo
- Serializzazione
  - ⇨processo secondo il quale lo stato di un oggetto viene trasformato in un flusso (stream) di bit
  - ⇨il processo contrario si chiama deserializzazione

Serializzazione

- Utilizzi della serializzazione
  - ⇨forma molto semplificata di persistenza: consente di salvare il contenuto degli oggetti su disco e successivamente ricaricarli
  - ⇨comunicazione via rete: consente di inviare oggetti da una macchina virtuale all'altra utilizzando la rete

Serializzazione

- Gli strumenti per la serializzazione
  - ⇨java.io.ObjectOutputStream
  - ⇨metodo writeObject(Object o)
- Gli strumenti per la deserializzazione
  - ⇨java.io.ObjectInputStream
  - ⇨metodo Object readObject()

Serializzazione

- Semantica dei metodi
  - ⇨producono una serializzazione e deserializzazione profonda
  - ⇨ovvero lavorano sull'oggetto e su tutto il grafo di oggetti in associazione
- Esempio: salvataggio su disco
  - ⇨nel file viene salvato lo stato di tutta la rete di oggetti raggiungibili dall'oggetto principale
  - ⇨in modo che poi siano recuperabili tutti

Serializzazione

- Le regole della serializzazione
  - ⇨analoghe per impostazione a quelle sulla clonazione
- I regola
  - ⇨perchè gli oggetti di una classe siano serializzabili/deserializzabili la classe deve avere un costruttore no-arg
  - ⇨in modo da poter deserializzare gli oggetti

Serializzazione

- II regola
  - ⇨perchè gli oggetti di una classe siano serializzabili, la classe deve implementare java.lang.Serializable (interfaccia vuota)
  - ⇨public interface Serializable {}
  - ⇨il metodo writeObject() lancia NotSerializableException nel caso in cui l'oggetto non è di tipo Serializable

Serializzazione

>> it.unibas.appuntamenti2

- Un esempio: appuntamenti 2
  - ⇨viene utilizzata la serializzazione come tecnica per gestire la persistenza
  - ⇨le classi coinvolte implementano Serializable
  - ⇨in effetti si tratta di un meccanismo di persistenza molto veloce
- Attenzione
  - ⇨il contenuto è binario e non condivisibile con altre applicazioni


```
public class DAOAgenda {

   public static Agenda carica(String nomeFile) throws DAOException {
      Agenda agenda = new Agenda();
      java.io.ObjectInputStream flusso = null;
      try {
         java.io.FileInputStream flussoFile = new java.io.FileInputStream(nomeFile);
         flusso = new java.io.ObjectInputStream(flussoFile);
         agenda = (Agenda)flusso.readObject();
      } catch (Exception e) { throw new DAOException(e);
      } finally {
         try { if (flusso != null) { flusso.close(); }
         } catch (java.io.IOException ioe) {}
      }
      return agenda;
   }

   public static void salva(Agenda agenda, String nomeFile) throws DAOException {
      java.io.ObjectOutputStream flusso = null;
      try {
         java.io.FileOutputStream flussoFile = new java.io.FileOutputStream(nomeFile);
         flusso = new java.io.ObjectOutputStream(flussoFile);
         flusso.writeObject(agenda);
      } catch (java.io.IOException ioe) { throw new DAOException(ioe);
      } finally {
         try { if (flusso != null) { flusso.close(); }
         } catch (Exception e) {}
      }
   }
}
```

Dettagli sulla Serializzazione

- Quali oggetti vengono serializzati
  - ⇨esclusivamente quelli che hanno uno stato (ovvero delle proprietà)
  - ⇨con la relativa rete di riferimenti
  - ⇨gli oggetti che non hanno proprietà ma solo metodi non vengono serializzati perchè la serializzazione non avrebbe senso

Dettagli sulla Serializzazione

- Proprietà transienti
  - ⇨non è sempre opportuno serializzare tutte le proprietà
  - ⇨I ragione: salvare alcune proprietà potrebbe essere inutile (il loro valore può essere ricostruibile)
  - ⇨II ragione: salvare alcune proprietà può essere controproducente (es: riferimenti a flussi o a connessioni)

Dettagli sulla Serializzazione

- Per impedire la serializzazione
  - ⇨Java fornisce la parola chiave transient
  - ⇨le proprietà statiche e le proprietà transienti non vengono serializzate
  - ⇨alla deserializzazione gli viene attribuito il valore standard
  - ⇨come fare per correggere questi valori ? è necessario personalizzare il processo di serializzazione/deserializzazione

Dettagli sulla Serializzazione

- Personalizzare il processo
  - ⇨è possibile definire in una classe serializzabile versioni private di writeObject() e readObject()
  - ⇨il metodo writeObject() di ObjectOutputStream verifica se l'oggetto ha un proprio metodo writeObject() e, se sì, demanda a queste versioni la serializzazione
  - ⇨utilizzando la riflessione (per questo private)

Dettagli sulla Serializzazione

- Il serial version UID
  - durante la serializzazione, viene salvato anche un hash del codice della classe
  - la ragione: impedire che un oggetto creato con una vecchia versione della classe possa essere deserializzato con una nuova versione (le proprietà potrebbero cambiare)
  - ma in alcuni casi questo non è un problema

Dettagli sulla Serializzazione

- Per risolvere il problema
  - è necessario utilizzare lo strumento serialver.exe per leggere il serial version UID della vecchia versione
  - ridefinire la proprietà serialVersionUID attribuendogli come valore il vecchio valore
  - in questo modo il valore usato dalle due versioni della classe sarà lo stesso

Dettagli sulla Serializzazione

>> materiale\LinkedList

- Un esempio
  - ⇨java.util.LinkedList
- Nota
  - ⇨la stragrande maggioranza delle classi delle API implementano Cloneable e Serializable

Riassumendo

- Clonazione
  - ⇨Clonazione Superficiale
  - ⇨Clonazione Profonda
- Serializzazione
  - ⇨Dettagli sulla Serializzazione

Termini della Licenza