Metodologie di Programmazione

Lezione 17: Esercizi su ereditarietà e polimorfismo

Lezione 17: Sommario

- Esercizi su ereditarietà e polimorfismo

Esercizio: la LibreriaComponibile

- Progettare una libreria componibile, costituita da una serie di scaffali
- Ogni scaffale contiene una sequenza di libri e prevede una capienza massima
- Un libro è rappresentato dal titolo e dall'autore
- Una libreria permette di aggiungere ed eliminare scaffali, di accedere al k-esimo scaffale e di ottenere il numero di scaffali
- Uno scaffale permette di aggiungere libri, eliminare libri per titolo e cercare libri per titolo
- La libreria inoltre permette di aggiungere un libro nel primo scaffale libero

Esercizio: RiproduttoreMusicale (1)

- Progettare una classe RiproduttoreMusicale che rappresenti un generico riproduttore
- La classe deve realizzare i seguenti metodi:
  - inserisciSupporto(): permette di inserire un supporto musicale (es. CD, nastro, ecc.)
  - espelliSupporto(): espelle il supporto
  - getBrano(): restituisce l'attuale brano in esecuzione (null se non sta eseguendo)
  - play(): esegue il brano attualmente selezionato
  - stop(): interrompe l'esecuzione
  - next(): seleziona il prossimo brano
  - prev(): seleziona il brano precedente
  - toString(): visualizza le informazioni del brano attualmente in esecuzione

Esercizio: RiproduttoreMusicale (2)

- Realizzare quindi i seguenti riproduttori
  - Giradischi
  - Mangianastri
  - Lettore CD
  - Lettore Mp3
- Realizzare inoltre diversi tipi di supporto:
  - Disco a 33 giri (14 brani)
  - Disco a 45 giri (2 brani)
  - Compact Disc (20 brani)
  - Nastro (con un numero specificato di minuti, numero di brani pari al numero di minuti / 5)
  - Memoria USB (1024 brani)
- Gli ultimi due supporti permettono di registrare/inserire brani nella posizione specificata

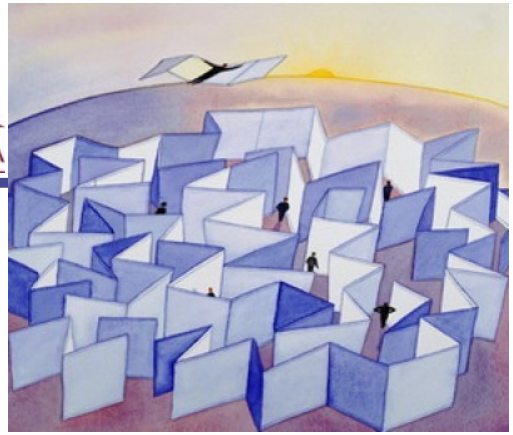
Esercizio: RiproduttoreMusicale (3)

- Ciascun supporto può contenere il numero massimo di brani specificati tra parentesi nella precedente diapositiva e viene costruito con una data sequenza (eventualmente vuota) di brani

- Modellare inoltre la classe Brano che contenga l'informazione sul nome del brano e il cantante

Esercizio: Il labirinto

UNITELMA SAPIENZA



- Progettare una classe Labirinto che rappresenti un labirinto
- La classe ha un unico punto di accesso a un singolo corridoio che viene reso disponibile al giocatore
- Ogni corridoio ha un numero variabile di punti di accesso ad altrettanti corridoi
- Un corridoio implementa un metodo che determina se il giocatore ha raggiunto il corridoio d'uscita del labirinto
- A ogni passo, il giocatore deve scegliere uno dei possibili corridoi accessibili dall'attuale posizione

Labirinto: Soluzione (1)


```
public class Corridoio
{
    private List<Corridoio> corridoi;

    public Corridoio()
    {
        corridoi = new ArrayList<Corridoio>();
    }

    public void addCorridoio(Corridoio c)
    {
        this.corridoi.add(c);
    }

    /**
     * Calcola il numero di corridoi (ovvero bivii) che e' possibile accedere da questo
     * corridoio
     * @return
     */
    public int getNumeroCorridoi() { return corridoi.size(); }

    public Corridoio getCorridoio(int k) { return corridoi.get(k-1); }

    /**
     * Consente di verificare se questo corridoio e' un'uscita o meno.
     * @return true se e solo se questo corridoio e' un'uscita.
     */
    public boolean isUscita() { return false; }
}
```

Labirinto: Soluzione (2)


```java
/**
 * Implementa il corridoio di uscita (visto come particolare tipo di corridoio).
 * <p>
 * Ridefinendo opportunamente il metodo che determina se vi e'
 * un'uscita, riusciamo a codificare la vittoria del giocatore
 * all'interno del labirinto.
 * @author flati
 *
 */
public class CorridoioUscita extends Corridoio
{
    @Override
    public boolean isUscita()
    {
        System.out.println("Ho trovato la via di uscita!");
        return true;
    }
}
```

Labirinto: Soluzione (3)


```
public class Labirinto
{
    private Corridoio c;

    public Labirinto(Corridoio c)
    {
        this.c = c;
    }

    /**
     * Da' accesso all'unico corridoio disponibile, consentendo
     * della visita all'interno del labirinto
     * @return il riferimento al corridoio di accesso
     */
    public Corridoio getEntrata()
    {
        return c;
    }

    public static void main(String[] args)
    {
        Corridoio entrata = new Corridoio();
        Corridoio strada1 = new Corridoio();
        Corridoio strada1a = new Corridoio();
        Corridoio strada1b = new Corridoio();

        Corridoio strada2 = new Corridoio();
        Corridoio strada2a = new Corridoio();
        Corridoio strada2b = new Corridoio();
        Corridoio strada2c = new Corridoio();

        Corridoio strada3 = new Corridoio();
        Corridoio strada3a = new Corridoio();
        Corridoio strada3b = new Corridoio();

        Corridoio uscita = new CorridoioUscita();

        // Creiamo il labirinto
        Labirinto labirinto = new Labirinto(entrata);
        entrata.addCorridoio(strada1);
        strada1.addCorridoio(strada1a);
        strada1.addCorridoio(strada1b);

        strada1b.addCorridoio(strada2);
        strada2.addCorridoio(strada2a);
        strada2.addCorridoio(strada2b);
        strada2.addCorridoio(strada2c);

        strada2a.addCorridoio(strada3);
        strada3.addCorridoio(strada3a);
        strada3.addCorridoio(strada3b);

        strada3a.addCorridoio(uscita);

        // Giochiamo
        GiocatoreLabirinto giocatore = new GiocatoreLabirinto("Marco");
        giocatore.entra(labirinto);
    }
}
```

Labirinto: Soluzione (4)


```
import java.util.Scanner;

/**
 * Modella il giocatore di un {@code Labirinto}. La logica di visita del labirinto si trova dentro
 * questa classe ed e' contenuta nel metodo prossimaMossa(Corridoio c).
 * <p>
 * Il giocatore di un {@code Labirinto} puo' entrare dal labirinto ed effettuare una prossima
 * mossa scegliendo un corridoio tra quelli che gli si parano davanti durante la visita.
 * @author flati
 *
 */
public class GiocatoreLabirinto
{
    private String nome;

    public GiocatoreLabirinto(String name)
    {
        this.nome = nome;
    }

    public void entra(Labirinto l)
    {
        Corridoio entrata = l.getEntrata();

        prossimaMossa(entrata);
    }
```

Labirinto: Soluzione (5)


```
public boolean prossimaMossa(Corridoio corridoio)
{
    while (!corridoio.isUscita())
    {
        if (corridoio.getNumeroCorridoi() == 0)
        {
            System.out.println(this + " ha trovato un vicolo cieco. Tornando indietro...");
            return false;
        }
        int i = scegliCorridoio(corridoio.getNumeroCorridoi());
        if (i <= 0) return false;

        Corridoio c = corridoio.getCorridoio(i);
        if (prossimaMossa(c)) return true;
    }

    System.out.println(this + " è uscito/a dal labirinto!");
    return true;
}

public int scegliCorridoio(int k)
{
    System.out.println("Scegli un corridoio!");
    System.out.println("k<=0 : torna indietro nel labirinto.");
    System.out.println("k>0 : percorri il k-esimo corridoio.");
    System.out.println("Massimo numero di corridoi percorribili: " + k);

    int i;
    do { i = new Scanner(System.in).nextInt(); } while (i > k);

    return i;
}
```

Esercizio: Esseri viventi (1)

- Progettare una classe EssereVivente dotata di età, sesso (M, F) e nome, con i seguenti metodi:
  - getEta(): restituisce l'età dell'essere
  - getSesso(): restituisce il sesso dell'essere
  - getNome(): restituisce il nome
  - cresce(): fa crescere l'essere vivente di 1 anno e con una certa probabilità lo fa morire
  - mangia(): fa mangiare l'essere vivente
  - siRiproduceCon(EssereVivente e): dato in ingresso un altro EssereVivente, se di sesso opposto fa riprodurre i due esseri e ne crea altri, altrimenti emette eccezione
  - muore(): causa la morte dell'essere vivente; a seguito di questa operazione, l'essere non può eseguire nessun'altra operazione

Esercizio: Esseri viventi (2)

- Implementare i seguenti tipi di esseri viventi:
  - EssereUmano: rappresenta un essere umano: si riproduce in un numero di 1 o 2 esemplari
  - Coniglio: rappresenta un coniglio, che si riproduce in al più 10 esemplari
  - Fenice: rappresenta l'uccello mitologico che resuscita dalle ceneri dopo la morte; si preveda quindi un metodo risorgi() che ha l'effetto di far rinascere la fenice
  - Gatto: rappresenta un gatto, che si riproduce in al più 5 esemplari, e che possiede 7 vite (ogni volta che muore, può accedere alla vita successiva)
  - Pesce: rappresenta un pesce, che si riproduce tra 30 e 100 esemplari
  - PescePagliaccio: rappresenta una specie di pesce che ha la capacità di cambiare sesso spontaneamente: si preveda quindi un metodo cambiaSesso() che provoca il cambio di sesso dell'oggetto; il sesso viene cambiato casualmente all'interno del metodo cresce()

EssereVivente: Soluzione (1)


```
abstract public class EssereVivente
{
    public enum Sesso { M, F }

    /**
     * Contiene il nome dell'essere
     */
    private String nome;

    /**
     * Mantiene l'informazione sull'eta'
     */
    protected int eta;

    /**
     * Mantiene l'informazione sul sesso dell'essere (M o F)
     */
    protected Sesso sesso;

    /**
     * Rappresentano rispettivamente il numero minimo e massimo di esseri che questa specie puo'
     * a seguito dell'invocazione siRiproduceCon()
     */
    protected int minFigli = 0;
    protected int maxFigli = 1;

    public EssereVivente(Sesso sex, String nome)
    {
        this.sesso = sex;
        this.nome = nome;
    }
```

EssereVivente: Soluzione (2)


```
public String getNome() { return nome; }
public int getEta() { return eta; }
public Sesso getSesso() { return sesso; }

/**
 * Causa la crescita di 1 anno dell'essere.
 * Con una certa probabilita' l'essere muore.
 * @throws EssereMortoException
 */
public void cresce() throws EssereMortoException
{
    if (isMorto()) throw new EssereMortoException();

    // Aumentiamo l'età  di 1
    eta++;

    // Calcolo probabilità  di morte
    int xshift = 50;
    double scale = 5.0;
    double probabilitaMorte = 1.0/(1+ Math.exp(-(eta-xshift)/scale));
    if (new Random().nextDouble() <= probabilitaMorte) muore();
}

public void mangia() throws EssereMortoException
{
    if (isMorto()) throw new EssereMortoException();

    System.out.println("Mangiando...");
}
```

EssereVivente: Soluzione (3)


```
public class EssereMortoException extends Exception
{

}


public class RiproduzioneException extends Exception
{

}
```

EssereVivente: Soluzione (4)


```
/**
 * Scatena la riproduzione di due esseri viventi.
 *
 *
 * @param e l'essere vivente con cui far riprodurre questo essere vivente
 * @return la lista degli essere viventi creati a seguito della riproduzione
 * @throws RiproduzioneException
 */
public ArrayList<EssereVivente> siRiproduceCon(EssereVivente e)
        throws RiproduzioneException, EssereMortoException
{

    if (isMorto()) throw new EssereMortoException();




}
```

EssereVivente: Soluzione (4)


```java
/**
 * Scatena la riproduzione di due esseri viventi.
 * Se il sesso dei due essere e' lo stesso, viene lanciata un'eccezione di tipo {@code HomosexualityException}.
 *
 * @param e l'essere vivente con cui far riprodurre questo essere vivente
 * @return la lista degli essere viventi creati a seguito della riproduzione
 * @throws RiproduzioneException
 */
public ArrayList<EssereVivente> siRiproduceCon(EssereVivente e)
        throws RiproduzioneException, EssereMortoException
{

    if (isMorto()) throw new EssereMortoException();

    // controlliamo che gli esseri viventi siano di sesso opposto
    if (getSesso() == e.getSesso()) throw new HomosexualityException();

    return genera(getNumeroRandomFigli());
}

/**
 * Delega la generazione alle sottoclassi concrete
 * @param numFigli numero di figli da generare
 * @return la lista di figli generati
 */
abstract protected ArrayList<EssereVivente> genera(int numFigli);
```

EssereVivente: Soluzione (5)


```
    public boolean isMorto()
    {
        return getEta() < 0;
    }

    public void muore()
    {
        System.out.println("Morendo...");
        eta = -1;
    }

    /**
     * Restituisce una stringa simbolica della razza a cui questo essere appartiene
     * @return
     */
    public String getRazza()
    {
        return getClass().getSimpleName();
    }

    @Override
    public String toString()
    {
        return getRazza() + " " + getNome() + " (sesso: "+getSesso()+" età : "+getEta()+")";
    }
}
```

Coniglio: Soluzione

```
public class Coniglio extends EssereVivente
{
    public static final int MAX_RABBIT_NUMBER = 10;

    public Coniglio(Sesso sesso, String nome)
    {
        super(sesso, nome);
        this.maxFigli = MAX_RABBIT_NUMBER;
    }

    protected ArrayList<EssereVivente> genera(int numFigli)
    {
        ArrayList<EssereVivente> figli = new ArrayList<EssereVivente>();

        for (int k = 0; k < numFigli; k++)
        {
            Sesso s = Sesso.values()[new Random().nextInt(2)];
            figli.add(new Coniglio(s, null));
        }

        return figli;
    }
}
```

Gatto: Soluzione

UNITELMA SAPIENZA

```
public class Gatto extends EssereVivente
{
    public static final int MAX_KITTEN_NUMBER = 5;

    public static final int MAX_VITE_GATTO = 7;
    private int viteResidue = MAX_VITE_GATTO;

    public Gatto(Sesso sex, String nome)
    {
        super(sex, nome);

        this.maxFigli = MAX_KITTEN_NUMBER;
    }

    @Override
    public void muore()
    {
        super.muore();

        viteResidue--;
        if (viteResidue > 0)
        {
            System.out.println("Ho ancora " + viteResidue + " vita/e davanti....Miao!");
            eta = 0;
        }
    }

    public int getViteResidue()
    {
        return viteResidue;
    }
}
```

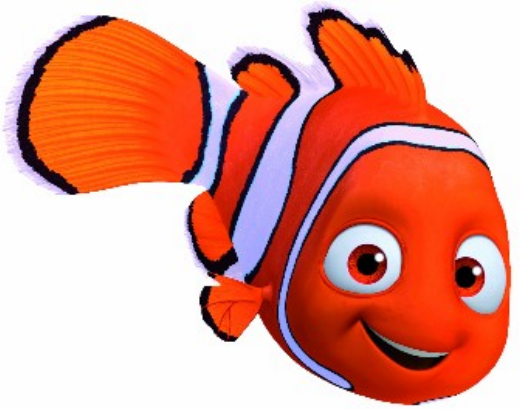
Pesce: Soluzione

```
public abstract class Pesce extends EssereVivente
{
    public static final int MIN_CHILDREN_NUMBER = 30;
    public static final int MAX_CHILDREN_NUMBER = 100;

    public Pesce(Sesso sesso, String nome)
    {
        super(sesso, nome);

        this.minFigli = MIN_CHILDREN_NUMBER;
        this.maxFigli = MAX_CHILDREN_NUMBER;
    }
}
```

PescePagliaccio: Soluzione



```
public class PescePagliaccio extends Pesce
{
    public PescePagliaccio(Sesso sesso, String nome)
    {
        super(sesso, nome);
    }

    /**
     * Con una probabilita' pari a 1/2, il PescePagliaccio crescendo cambia sesso in modo spontaneo
     * @throws EssereMortoException
     */
    @Override
    public void cresce() throws EssereMortoException
    {
        super.cresce();

        if (new Random().nextBoolean()) cambiaSesso();
    }

    public void cambiaSesso()
    {
        if (isMorto()) return;

        System.out.println(this + ": cambiando sesso...");
        this.sesso = this.sesso == Sesso.M ? Sesso.F : Sesso.M;

    }
}
```

Fenice: Soluzione

```
public class Fenice extends EssereVivente
{
    public Fenice(Sesso sex, String nome)
    {
        super(sex, nome);
    }

    @Override
    public void muore()
    {
        super.muore();
        this.risorge();
    }

    public void risorge()
    {
        System.out.println("La fenice " + getNome() + " risorge dalle proprie ceneri!");
        eta = 0;
    }
}
```