Dipartimento di Informatica e Sistemistica
Antonio Ruberti

“Sapienza” Università di Roma

Esercitazione 2

Corso di Tecniche di Programmazione

Laurea in Ingegneria Informatica

(Canale di Ingegneria delle Reti e dei Sistemi Informatici - Polo di Rieti)

Anno Accademico 2007/2008

Tutor: Ing. Diego Rughetti

Si ringrazia il Prof. Enrico Denti per aver reso
disponibile il materiale didattico sul quale si basano queste slides

Argomenti di riferimento

- Ereditarietà
- Classi Astratte
- Polimorfismo
- Eccezioni

Esercizio

GESTIONE DI VEICOLI

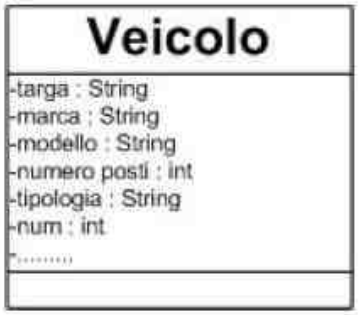
- Si vogliono realizzare una o più classi che permettano di modellare diverse tipologie di **veicolo**: **autocarri** e **autoveicoli**.
- Tutti i veicoli sono caratterizzati da:
  - targa
  - marca
  - modello
  - numero posti
- gli autocarri sono caratterizzati inoltre da:
  - la capacità massima di carico in quintali
- mentre gli autoveicoli da:
  - il numero delle porte

Soluzione (1)

PRIMA SOLUZIONE: UN'UNICA CLASSE

La classe deve contenere:

- La tipologia del veicolo → autoveicolo o autocarro
- La capacità di carico o il numero delle porte, in base alla tipologia del veicolo.

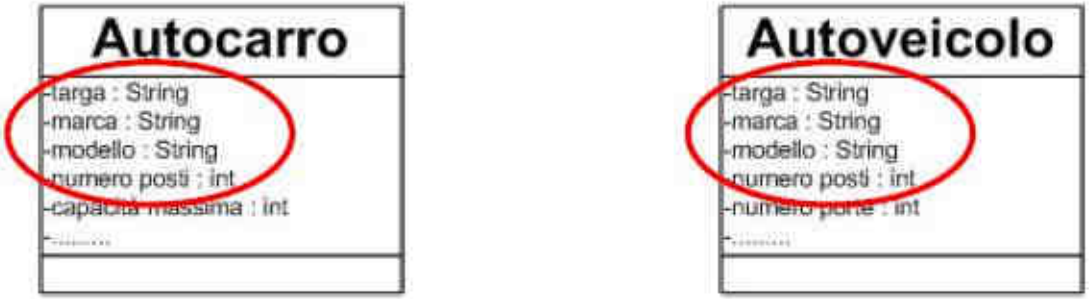


E se volessimo aggiungere un nuovo tipo di veicolo?

Soluzione (2)

SECONDA SOLUZIONE: DUE CLASSI

Si potrebbe realizzare una classe per ogni tipologia di veicolo.



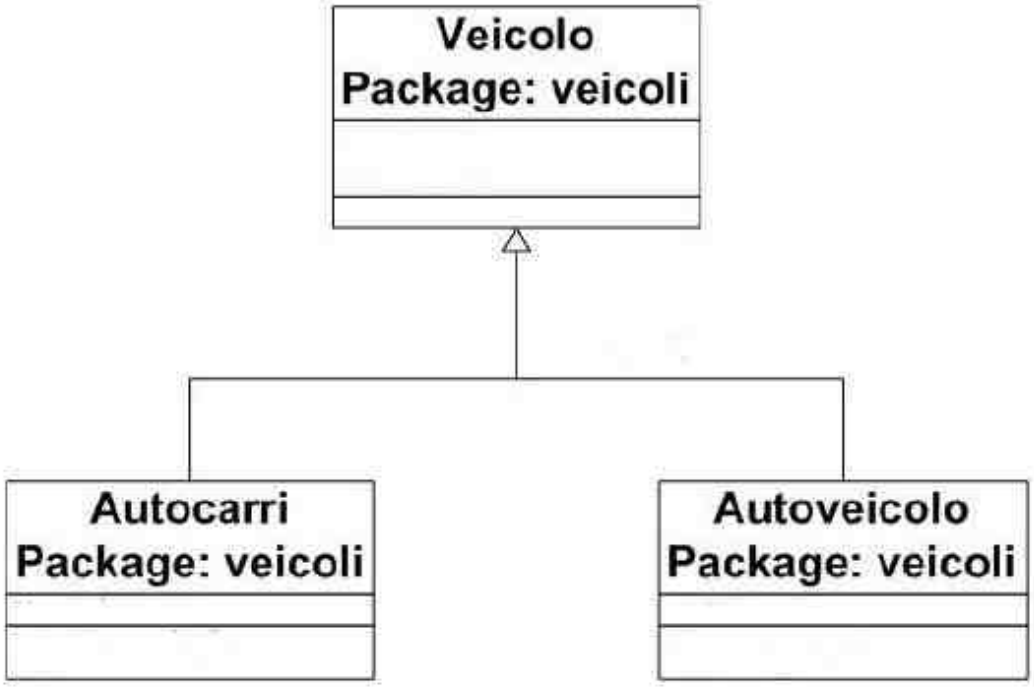
Le diverse tipologie di veicolo hanno caratteristiche comuni

→ ripetizione di codice

Soluzione (3)

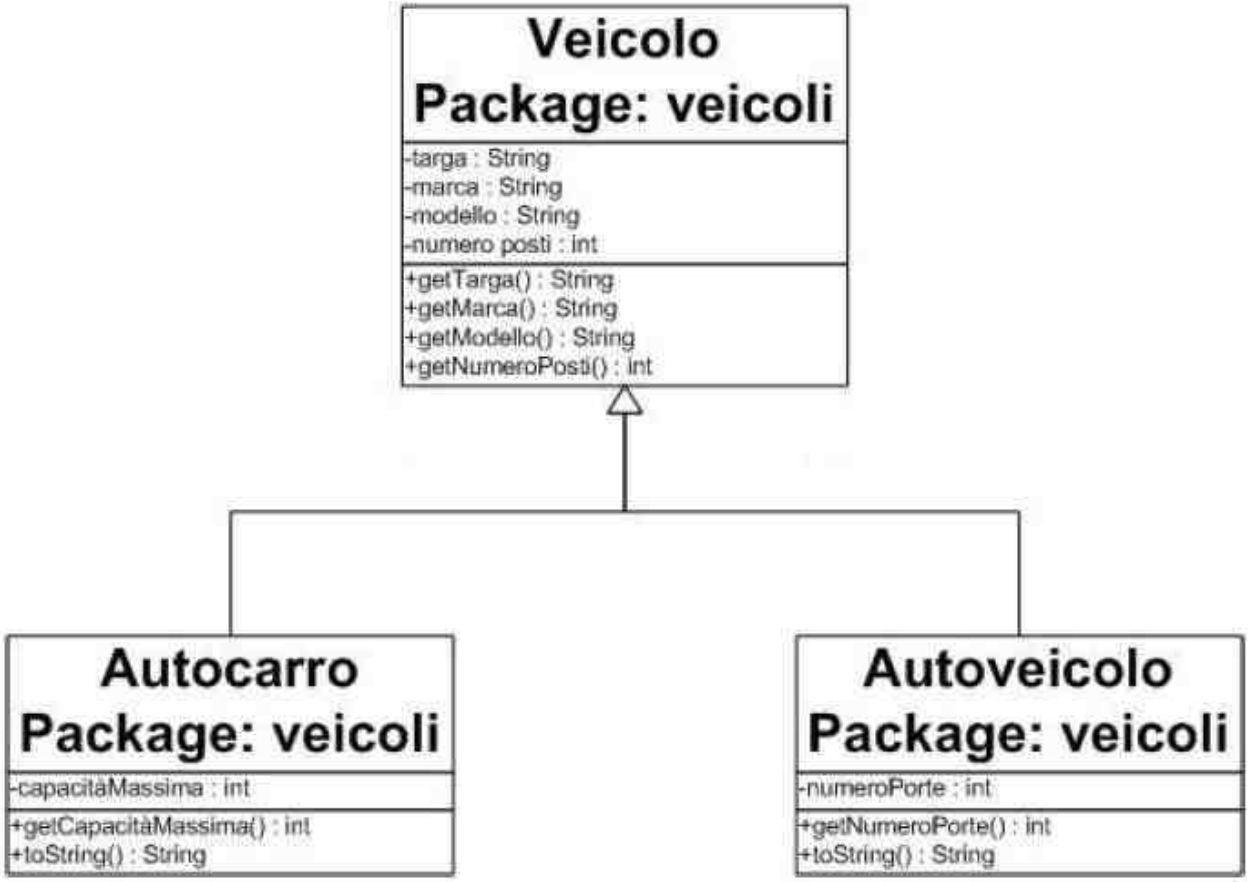
TERZA SOLUZIONE: UNA GERARCHIA DI CLASSI

- Una classe `Veicolo` per rappresentare un generico veicolo
- Una classe `Autocarro` rappresentare un veicolo di tipo autocarro
- Una classe `Autovettura` rappresentare un veicolo di tipo autoveicolo



Soluzione (4)

CLASSI NEL DETTAGLIO



Soluzione (5)

```
package veicoli;

public class Veicolo{
  protected String targa; protected String marca;
  protected String modello; protected int numeroPosti;

  public Veicolo(String t, String m, String mo, int n){
                this.targa = t;
                this.marca = m;
                this.modello = mo;
                this.numeroPosti = n;
  }
  public String getTarga(){
                return targa;
  }
  public String getMarca(){
                return marca;
  }
  public String getModello(){
                return modello;
  }

  public int getNumeroPosti(){
                return numeroPosti;
  }

  public String toString(){
                return "targa = " + targa + " marca = " + marca + "
                                modello = " + modello + "
                                numero posti = " + numeroPosti + " ";
  }
}
```

Soluzione (6)

```
package veicoli;

public class Autocarro extends Veicolo{
		private int capacaitàMassima;

		public Autocarro(String targa, String marca, String modello, int numeroPosti, int capacaitàMassima){
			super(targa, marca, modello, numeroPosti);
			this.capacaitàMassima = capacaitàMassima;
		}

		public int getCapacaitàMassima(){
			return capacaitàMassima;
		}

		public String toString(){
			return super.toString() + " capacità massima = " + capacaitàMassima;
		}
}
```

Soluzione (7)

```
package veicoli;

public class Autoveicolo extends Veicolo{
		private int numeroPorte;

		public Autoveicolo(String targa, String marca, String modello, int numeroPosti, int numeroPorte){
			super(targa, marca, modello, numeroPosti);
			this.numeroPorte = numeroPorte;
		}

		public int getNumeroPorte(){
			return numeroPorte;
		}

		public String toString(){
			return super.toString() + " numero porte = " + numeroPorte;
		}
}
```

Soluzione (8)

POSSIBILE TEST

Test.java

```
import veicoli.*;
public class Test
{
   public static void main(String args[])
   {
      Veicolo v[] = {new Autoveicolo("FC 23423","Fiat","Panda",5,5),
      new Autocarro("BO 23023","Fiat","IVECO 109.14",5,3),
      new Veicolo("FC 23423","Audi","A3",5)};
      ……
   }
}
```

È possibile fare ciò perché `Autocarro` e `Autoveicolo` sono anche oggetti di tipo `Veicolo`

(segue)

Soluzione (9)

(segue)

POSSIBILE TEST

Test.java

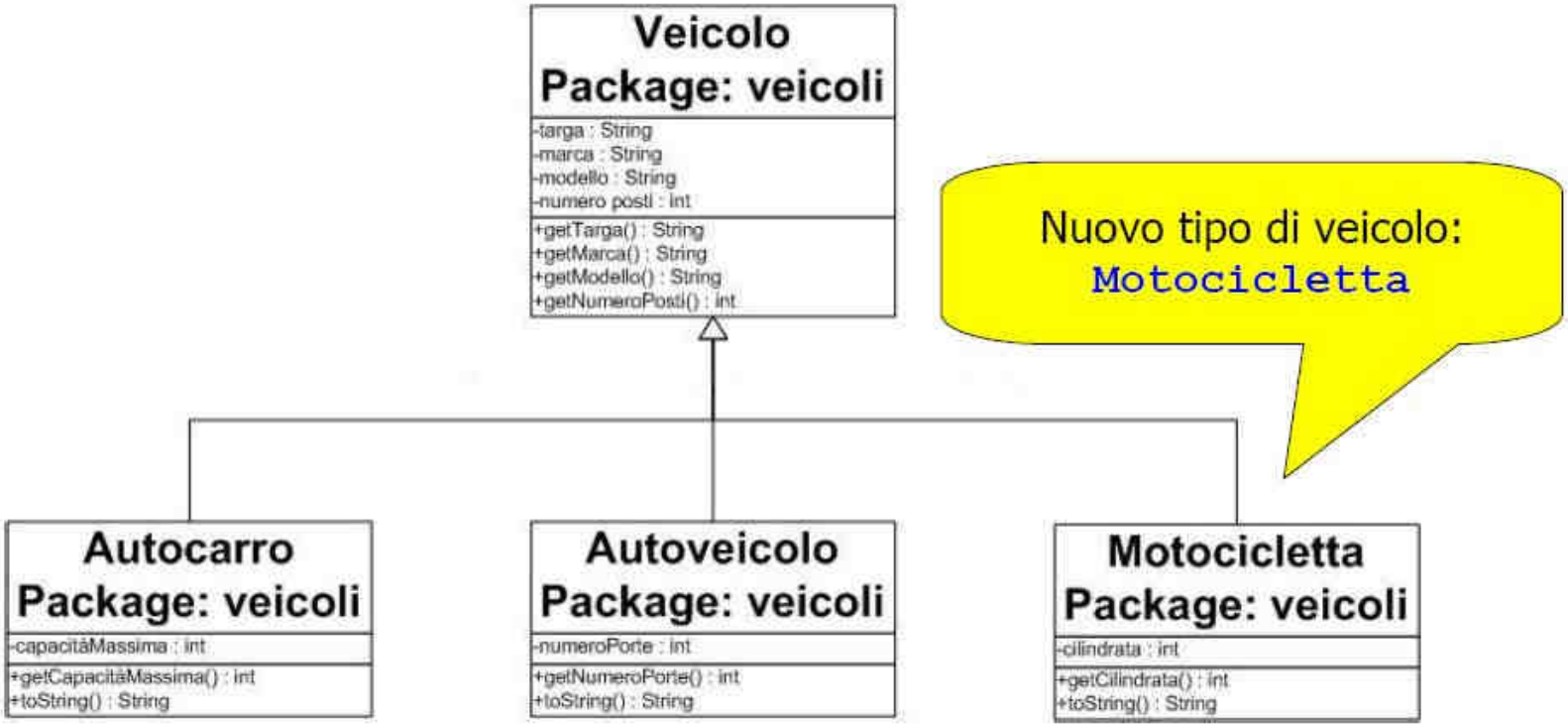
```
import veicoli.*;
public class Test
{
   public static void main(String args[])
   {
      ……..
      for(int i=0;i<v.length;i++)
          System.out.println(v[i]);
   }
}
```

Viene richiamata la toString() corretta
→ **Polimorfismo**

Estensione esercizio 1

Se si vuole aggiungere un nuovo tipo di veicolo, grazie al meccanismo dell'ereditarietà, è necessario solamente realizzare una nuova classe che erediti da `Veicolo` e che aggiunga nuove variabili e operazioni

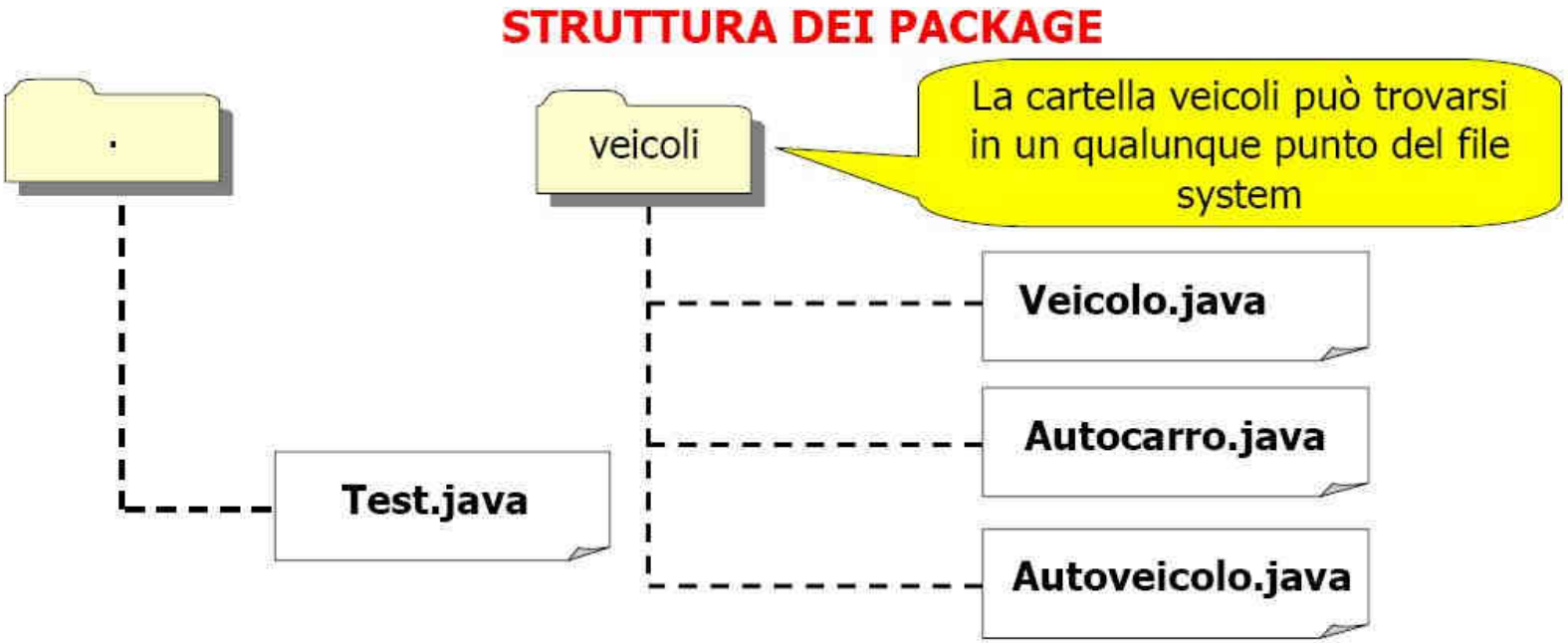
NUOVA TASSONOMIA



Soluzione estensione

```
package veicoli;

public class Motocicletta extends Veicolo{
		private int cilindrata;

		public Motocicletta(String targa, String marca, String modello, int numeroPosti, int cilindrata){
				super(targa, marca, modello, numeroPosti);
				this.cilindrata = cilindrata;
		}

		public int getCilindrata(){
				return cilindrata;
		}

		public String toString(){
				return super.toString() + " cilindrata = " + cilindrata;
		}
}
```

Package (1)



Se la **cartella veicoli** e il file **Test.java** si trovano nella cartella di lavoro vengono utilizzati i seguenti comandi (non c'è bisogno di indicare il **path** tramite **–cp**):

Compilazione: C:……\cartelladilavoro> `javac *.java`

Esecuzione: C:……\cartelladilavoro> `java Test`

(segue)

Package (2)

(segue)

Se invece ad esempio la **cartella veicoli** non si trova nella cartella di lavoro, vengono utilizzati i seguenti comandi (c'è bisogno di indicare il **path** tramite **–cp**):

Compilazione:

```
C:.....\cartelladilavoro>javac -cp .;percorsocartellaveicoli *.java
```

Esecuzione:

```
C:.....\cartelladilavoro>java -cp .;percorsocartellaveicoli Test
```

Dato che la cartella veicoli è associata ad un package, **percorsocartellaveicoli** deve riportare il percorso solo fino alla cartella superiore a questa. Nel caso la cartella sia "normale", è necessario invece indicarla nel percorso.

Proposta di esercizio

- Realizzare un insieme di classi in relazione gerarchica che modellino diversi tipi di **dipendenti** di un'azienda
- Ogni dipendente è caratterizzato da: un identificativo, un nome, un cognome
- I dipendenti possono essere:
  - **Operai**. Ogni operaio appartiene ad un settore
  - **Impiegato**. Ogni impiegato lavora in un ufficio ed ha un numero di telefono
  - **Dirigente**. Ogni dirigente ha un ufficio, un numero di telefono e una macchina aziendale
  - Ogni dipendente ha uno stipendio che dipende dalla sua tipologia
- Si realizzi una classe contenente un **main** di prova e si crei il **jar** dell'applicazione

Esercizio 2

Si vuole realizzare un insieme di classi per la gestione del personale di un’istituzione. Ogni persona che lavora nell’istituzione è caratterizzata da un nome, un indirizzo ed un numero di telefono. Deve essere possibile in qualsiasi momento ottenere una rappresentazione di un membro del personale sotto forma di stringa o calcolare la sua paga.

Nell’istituzione esistono due tipologie di personale, i volontari e i dipendenti. I dipendenti sono caratterizzati, oltre che dalle informazioni precedentemente indicate per tutto il personale, anche da un codice fiscale e da una base di retribuzione. Un volontario percepisce una paga pari a 0, un dipendente percepisce una paga pari alla base di retribuzione.

I dipendenti a loro volta possono essere, giornalieri o impiegati. I giornalieri sono caratterizzati anche dalle giornate lavorative svolte e la loro paga è calcolata moltiplicando la loro base di retribuzione per il numero di giorni in cui hanno lavorato. Gli impiegati sono caratterizzati dal numero di bonus maturati e la loro paga è calcolata sommando i bonus maturati alla base di retribuzione

Infine aggiungiamo che se necessario si può organizzare il personale in gruppi o staffs.

Nella realizzazione delle classi in Java gestire le condizioni di errore lanciando le opportune eccezioni.

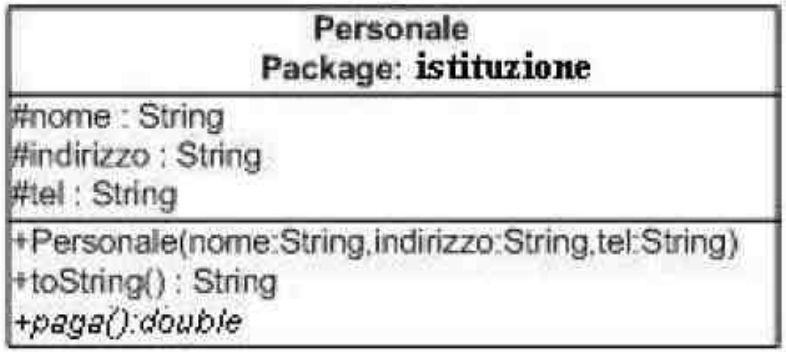
Soluzione (1)


Classe astratta
ESERCIZIO "ISTITUZIONE"
Personale
Package: istituzione
Staff
Package: istituzione
-lista:Personale[]
+Staff()
+giornoDiPaga()
Istituzione
+static void main(args:String[])
Volontario
Package: istituzione
Dipendente
Package: istituzione
Giornaliero
Package: istituzione
Impiegato
Package: istituzione

Soluzione (2)

CLASSE `Personale`

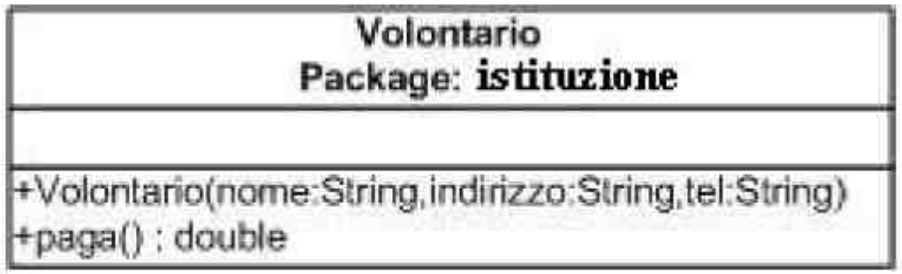
- La suddetta **classe** deve:
  - contenere tre valori (`String`) per rappresentare rispettivamente il nome, l'indirizzo e il telefono associati all'istanza corrente
  - contenere un costruttore con tre parametri, per rappresentare rispettivamente il nome, l'indirizzo e il telefono da associare all'istanza che si vuole creare
  - ridefinire il metodo `toString()`
  - contenere un metodo astratto `paga()`, che servirà per calcolare e restituire la paga associata all'istanza corrente



Soluzione (3)

CLASSE `Volontario`

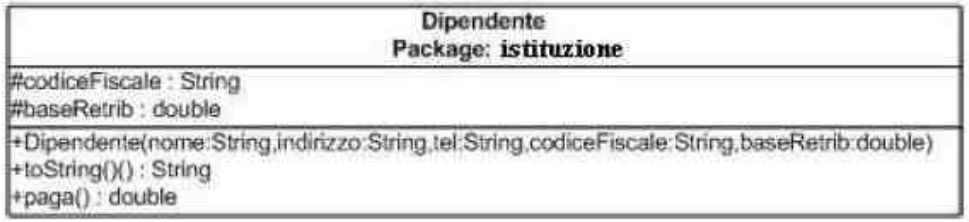
- La suddetta **classe** deve:
  - estendere la classe `Personale`
  - contenere un costruttore con tre parametri, per rappresentare rispettivamente il nome, l'indirizzo e il telefono da associare all'istanza che si vuole creare (**IMPORTANTE:** deve richiamare un costruttore della superclasse)
  - ridefinire il metodo `paga()` → la paga di un volontario è pari a 0,00 euro



Soluzione (4)

CLASSE `Dipendente`

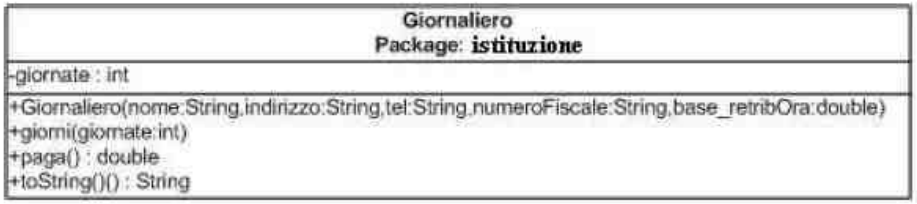
- La suddetta **classe** deve:
  - estendere la classe `Personale`
  - contenere due valori, per rappresentare rispettivamente il codice fiscale (`String`) e la base di retribuzione (`double`) associati all'istanza corrente
  - contenere un costruttore con cinque parametri, per rappresentare rispettivamente il nome, l'indirizzo, il telefono, il codice fiscale e la base di retribuzione da associare all'istanza che si vuole creare (**IMPORTANTE:** deve richiamare un costruttore della superclasse)
  - ridefinire il metodo `toString()`
  - ridefinire il metodo `paga()` → la paga di un dipendente è rappresentata dalla base di retribuzione



Soluzione (5)

CLASSE `Giornaliero`

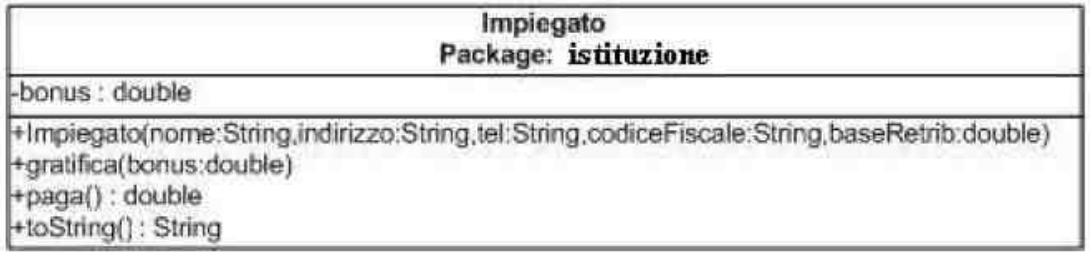
- La suddetta **classe** deve:
  - estendere la classe `Dipendente`
  - contenere un valore (`int`), per rappresentare il numero di giornate lavorative associate all'istanza corrente
  - contenere un costruttore con cinque parametri, che rappresentano rispettivamente il nome, l'indirizzo, il telefono, il codice fiscale e la base di retribuzione oraria da associare all'istanza che si vuole creare (**IMPORTANTE:** deve richiamare un costruttore della superclasse)
  - contenere un metodo `giorni()` che permetta di aggiungere il numero di giornate lavorative effettuate
  - ridefinire il metodo `toString()` (**SUGGERIMENTO:** sfruttare quello della superclasse)
    ridefinire il metodo `paga()` → la paga di un dipendente giornaliero è rappresentata dalla base di retribuzione (oraria) moltiplicata per i giorni lavorativi. Una volta restituita la paga, il numero di giornate lavorative va azzerato



Soluzione (6)

CLASSE `Impiegato`

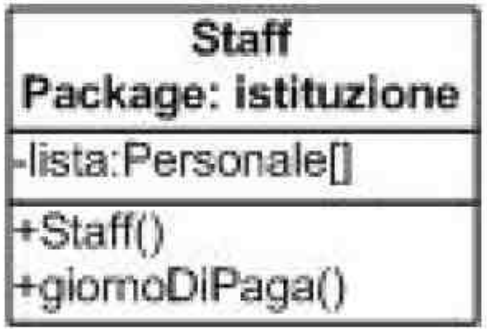
- La suddetta **classe** deve:
  - estendere la classe `Dipendente`
  - contenere un valore (`double`), per rappresentare il numero di bonus ottenuti
  - contenere un costruttore con cinque parametri, che rappresentano rispettivamente il nome, l'indirizzo, il telefono, il codice fiscale e la base di retribuzione da associare all'istanza che si vuole creare (**IMPORTANTE:** deve richiamare un costruttore della superclasse)
  - contenere un metodo `gratifica()` che permetta di aggiungere il numero di bonus
  - ridefinire il metodo `toString()` (**SUGGERIMENTO:** sfruttare quello della superclasse)
  - ridefinire il metodo `paga()` → la paga di un dipendente impiegato è rappresentata dalla base di retribuzione sommata ai bonus (**SUGGERIMENTO:** sfruttare il metodo corrispondente della superclasse). Una volta restituita la paga, il numero di bonus va azzerato



Soluzione (7)

CLASSE `Staff`

- La suddetta **classe** deve:
  - contenere un insieme di elementi di tipo `Personale` (`Personale[] lista`), per rappresentare lo staff di un'istituzione
  - contenere un costruttore senza parametri, che si occupi di inizializzare l'`array` con istanze delle sottoclassi di `Personale` (`Volontario`, `Dipendente`, `Giornaliero`, `Impiegato`)
  - contenere un metodo `giornoDiPaga()`, per simulare la giornata in cui viene pagato il personale di un'istituzione



Soluzione (8)

GIORNATA DI PAGA

La giornata di paga viene simulata stampando a video la paga da dare a ciascun membro del personale

Soluzione (9)

```
package istituzione;
abstract class Personale{
  protected String nome;   protected String indirizzo;   protected String tel;

  public Personale (String nome, String indirizzo, String tel)  throws IstituzioneException{
    if(nome == null || indirizzo == null || tel == null)
                throw new IstituzioneException(“uno o più argomenti non validi!”);
    this.nome = nome;
    this.indirizzo = indirizzo;
    this.tel = tel;
  }
  public String toString (){
    String riporta = "Nome: " + nome + "\n";
    riporta += "Indirizzo: " + indirizzo + "\n";
    riporta += "Telefono: " + tel;
    return riporta;
  }
  public abstract double paga();
}
```

Soluzione (10)

```
package istituzione;

class Volontario extends Personale{
   public Volontario (String nome, String indirizzo, String tel)  throws IstituzioneException{
      super (nome, indirizzo, tel);
   }

   public double paga(){
      return 0.0;
   }
}
```

Soluzione (11)

```
package istituzione;

class Dipendente extends Personale{
  protected String numeroFiscale;   protected double base_retribuzione;

  public Dipendente (String nome, String indirizzo, String tel, String numeroFiscale, double base_retribuzione) throws IstituzioneException{
    if(numeroFiscale == null || base_retribuzione == null)
                throw new IstituzioneException(“uno o più argomenti non validi!”);
    super(nome, indirizzo, tel);
    this.numeroFiscale = numeroFiscale;
    this.base_retribuzione = base_retribuzione;
  }
  public String toString(){
    String riporta = super.toString();
    riporta += "\nCodice fiscale: " + numeroFiscale;
    return riporta;
  }
  public double paga(){
    return base_retribuzione;
  }
}
```

Soluzione (12)

```
Package istituzione;
class Giornaliero extends Dipendente{
  private int giornate;
  public Giornaliero (String nome, String indirizzo, String tel, String numeroFiscale, double base_retribuzione_ora) throws IstituzioneException{
    super (nome, indirizzo, tel, numeroFiscale, base_retribuzione_ora);
    giornate = 0;
  }
  public void giorni (int extraGiornate){
    if(!(extraGiornate >= 1 && extraGiornate <= 31) )
                throw new IstituzioneException(“Argomento passato non valido”);
    giornate += extraGiornate;
  }
  public double paga (){
    double pagamento = base_retribuzione * giornate;
    giornate = 0;      return pagamento;
  }
  public String toString (){
    String riporta = super.toString();
    riporta += "\nGiornate lavorate: " + giornate;
    return riporta;
  }
}
```

Soluzione (13)

```
package istituzione;
class Impiegato extends Dipendente{
  private double bonus;
  public Impiegato(String nome,String indirizzo,String tel,String codiceFiscale,double baseRetrib) throws IstituzioneException{
    super(nome,indirizzo,tel,codiceFiscale,baseRetrib);
    bonus = 0;
  }
  public void gratifica(double bonus)   {
    if(bonus <= 0 )
                throw new IstituzioneException(“Argomento passato non valido, deve essere > 0”);
    this.bonus = bonus;
  }
  public double paga()   {
    double pagamento = super.paga() + bonus;
    bonus = 0;      return pagamento;
  }
  public String toString()   {
    String riporta = super.toString();
    riporta += "\nNumero bonus: " + bonus;
    return riporta;
  }
}
```

Soluzione (14)

```
package istituzione;


public class Staff{
  Personale[] lista;
  public Staff()   {
    try{
        lista = new Personale[6];
        lista[0] = new Dipendente ("Carlo Rossi", "Via dei Gracchi 0", “555-0469", "CRLRSS53B12F205T", 1923.07);
        lista[1] = new Dipendente ("Amintore Fanfani", "Piazza del Gesu' 15","555-0101", "MNRFNF11C13G243F", 846.15);
        lista[2] = new Impiegato ("Palmiro Togliatti", "Via Botteghe Oscure 1","555-0000", "PLRTGT17F05F205A", 769.23);
        ((Impiegato)(lista[2])).gratifica(50.00);
        lista[3] = new Giornaliero ("Pierino Porcospino", "Via Anfiteatro 4","555-0690", "PRNPRP65C10F205R", 18.55);
        ((Giornaliero)lista[3]).giorni(40);
        lista[4] = new Volontario ("San Camillo", "4 Ciclo", "555-8374");
        lista[5] = new Volontario ("San Callisto", "5 Ciclo","555-7282");
    }catch(IstituzioneException e){
        System.out.println(“Catturata un’eccezione IstituzioneException”);
        e.printStackTrace();
    }
}
```

Soluzione (14-2)

```
public void giornoDiPaga()   {
        double totale;
        System.out.println();
        System.out.println("*** calcolo delle paghe del personale ***");
        System.out.println();

        for (int i=0; i < lista.length; i++)      {
                System.out.println (lista[i]);
                totale = lista[i].paga();

                if (totale == 0.0)
                        System.out.println ("Grazie per il vostro contributo!");
                else
                        System.out.println ("Pagamento totale dovuto: " + totale);
                System.out.println ("-----------------------------------");
        }
 }
}
```

Soluzione (15)

```
package istituzione;

import java.util.*;

public class IstituzioneException extends Exception{

        public IstituzioneException()       {
                super();
        }

        public IstituzioneException(String s){
                super(s);
        }
}
```

Soluzione (16)

CLASSE `Istituzione`

Si vuole realizzare una **classe** `Istituzione` che permetta di testare l'applicazione.

Istituzione.java

```
import istituzione.*;
class Istituzione
{
   public static void main (String[] args)
   {
      Staff staff = new Staff();
      staff.giornoDiPaga();
   }
}
```

necessità di importare le classi del package **istituzione**

Esercizio 3

ESERCIZIO "CONTO BANCARIO"

- Si vuole realizzare una **classe** `ContoBancario` che permetta di modellare un generico conto bancario.

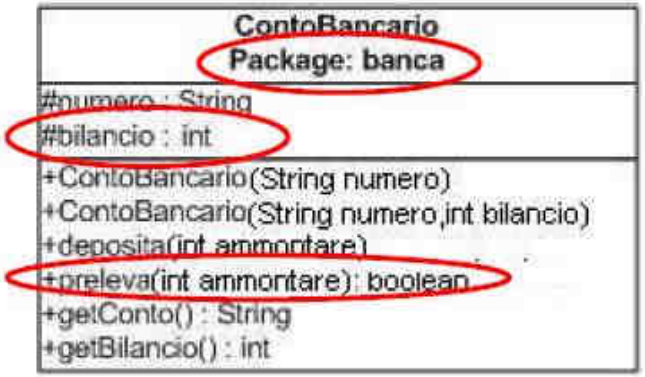
- La suddetta **classe** deve:
  - contenere un valore (`String`), per rappresentare il numero del conto corrente
  - contenere un valore (`int`), per rappresentare il bilancio del conto corrente
  - contenere un costruttore con un parametro che rappresenta il numero del conto corrente. Il bilancio viene inizializzato con il valore 0
  - contenere un secondo costruttore con due parametri, che rappresentano rispettivamente il numero del conto corrente e il bilancio del conto

(segue)

Esercizio 3 (2)

(segue)

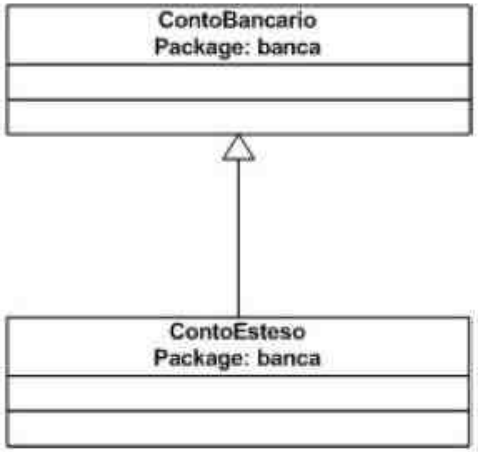
- permettere di conoscere il numero e il bilancio del conto corrente
- permettere di depositare e prelevare somme di denaro dal conto corrente → il prelievo può avvenire solo se il conto corrente presenta un bilancio sufficiente



Soluzione (1)

ESTENSIONE CLASSE `ContoBancario`

- Si vuole realizzare una **classe** `ContoEsteso` che permetta di modellare un conto bancario con fido.
- La suddetta **classe** deve:
  - estendere la classe `ContoBancario`
  - contenere un valore (`int`), per rappresentare il valore del fido
  - contenere un costruttore con un parametro che rappresenta il numero del conto corrente. Il fido viene inizializzato con il valore 1000
  - contenere un secondo costruttore con due parametri che rappresentano rispettivamente il numero del conto corrente e il bilancio del conto. Il fido viene inizializzato con il valore 1000
  - contenere un terzo costruttore con tre parametri che rappresentano rispettivamente il numero del conto corrente, il bilancio e il fido del conto .

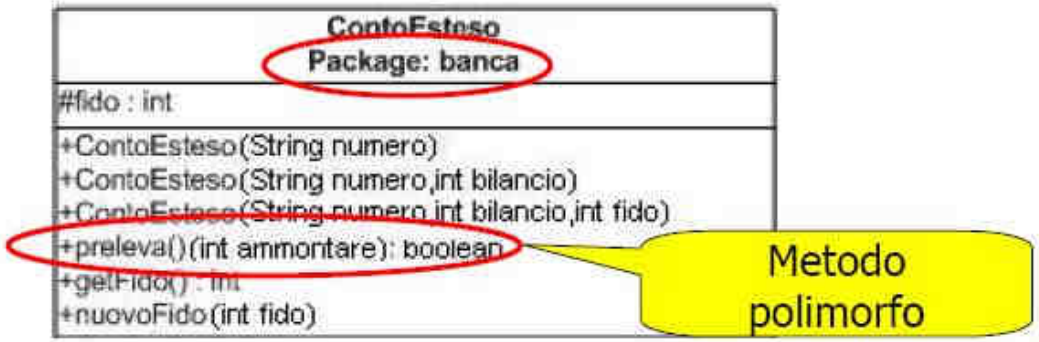


(segue)

Soluzione (2)

(segue)

- permettere di conoscere il valore del fido
- permettere di creare un nuovo fido
- ridefinire il metodo `preleva` della **superclasse** → è possibile prelevare una determinata somma di denaro da un conto con fido solo se questo presenta un bilancio sufficiente e se la radice della differenza fra il bilancio e la somma richiesta è minore uguale al fido associato al conto bancario



Soluzione (3)

TEST DI PROVA

Banca.java

```
import banca.*;

public class Banca{

   public static void main(String args[])
   {
                              ...
   }
}
```

necessità di importare le classi del package **banca**

Verificare il funzionamento del polimorfismo tramite il metodo **preleva**