Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Dipartimento di Elettronica e Informazione

Esercitazioni di Ingegneria del Software

03: Polimorfismo

1. Completare il codice

Visibilità

Completare il codice con gli opportuni modificatori di visibilità (usare la visibilità più ristretta compatibilmente con il codice del programma).

I confini delle compilation unit non vengono riportati (dichiarazioni facenti parte di uno stesso package potrebbero appartenere a compilation unit diverse).

1. Completare il codice

Visibilità

```
package a;
... class First {
  ...int x;  ... int y;
  ... void h() { y = -1; }
}
... class Second extends First {
  ... void f(int x) { this.x = x; h(); }
}
package b;
imports a.*;
...class Third {
  ... static void main(String[] s) {
    Second z = new Second();
    z.f(3);
}
class Fourth extends First {   void g() { h(); }  }
```


1. Soluzione

Visibilità

```
package a;
public class First {
  /* friendly o protect */ int x;  private int y;
  protected void h() { y = -1; }
}
public class Second extends First {
  public void f(int x) { this.x = x; h(); }
}
package b;
imports a.*;
public class Third {
  public static void main(String[] s) {
    Second z = new Second();
    z.f(3);
}
class Fourth extends First {   void g() { h(); }  }
```

2. Cosa stampa? (costruttori)

```
public abstract class Persona {
    protected String nome;
    abstract public void stampa();

    Persona(String nome) {
          this.nome = nome;
          System.out.println("Sono in Persona prima di stampa.");
          stampa();
          System.out.println("Sono in Persona dopo stampa.");
    }
}
public class Studente extends Persona {
    protected int matricola;

    Studente(String nome, int matricola) {
          super(nome);
          this.matricola    =matricola;
    }

    public void stampa() {
          System.out.printf("Nome: %s Matricola: %d\n", nome, matricola);
    }
}
```

Output:
Sono in Persona prima di stampa.
Nome: Marco Matricola: 0
Sono in Persona dopo stampa.
Nome: Marco Matricola: 12345

```
Studente s = new Studente("Marco", 12345);
s.stampa();
```


2(bis). Cosa stampa? (instanceOf)

```
class Father { }
class Son extends Father { }
class Test {
  public static void main(String[] s) {
    Father f = new Son(); Father f2 = new Father();
    if (f instanceof Father) System.out.println("True");
    else System.out.println("False");
    if (f.getClass() = = f2.getClass())
          System.out.println("True");
    else System.out.println("False");
  }
}
```

2(bis). Cosa stampa? (instanceOf)

- Risposta: stampa
  True
  False
- L'esempio mostra la differenza tra l'uso di instanceof e quello del confronto diretto degli oggetti Class per verificare a runtime che classe ha l'oggetto. L'operatore instanceof ritorna true se c'è compatibilità di assegnamento. Nel primo caso, dal momento che il tipo dinamico di f è Son, e che questo è un sottotipo di Father, instanceof ritorna true. Il metodo getClass, invece, ritorna la classe runtime dell'oggetto al quale si fa riferimento. Dal momento che il tipo dinamico di f è Son, e quello dinamico di f2 è Father, i due oggetti Class saranno diversi, e quindi viene stampato False. Il secondo confronto, quindi, viene fatto confrontando direttamente i tipi dinamici di f e di f2.

3. Polimorfismo

```
public interface Persona {
  String getNome();
}

public interface Studente extends Persona {
  int getMatricola();
}

public interface Lavoratore extends Persona{
  float getSalario();
}
```

3. Polimorfismo

```
public class PersonaImpl implements Persona{
  private String nome;

  public PersonaImpl(String nome) {
    this.nome = nome;
  }

  public String getNome() {
    return nome;
  }
}
```


3. Polimorfismo

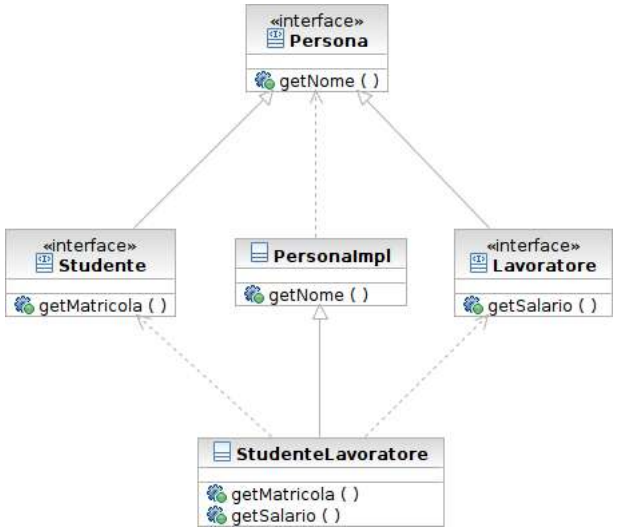
```
public class StudenteLavoratore
  extends PersonaImpl
  implements Studente, Lavoratore {

  private int matricola;
  private float salario;

  public StudenteLavoratore(String nome,
                    int matricola, float salario) {
    super(nome);
    this.matricola = matricola;
    this.salario = salario;
  }

  public int getMatricola() { return matricola; }
  public float getSalario() { return salario; }
}
```

3. Polimorfismo



3. Polimorfismo

```
public class Launcher {
  public static void main(String[] args) {
    Persona p; Studente s; Lavoratore l;

    p = new StudenteLavoratore("Gianni", 123, 100);
    System.out.println(p.getNome());
    System.out.println(p.getSalario());

    s = new PersonaImpl("Mario");

    l = p;
    l = (Lavoratore) p;
    System.out.println(l.getNome());
    System.out.println(l.getSalario());
    System.out.println(l.getMatricola());

    s = p;
    s = (Studente) p;
    s = (StudenteLavoratore) p;
    System.out.println(s.getNome());
    System.out.println(s.getSalario());
    System.out.println(s.getMatricola());
```

3. Polimorfismo

```
public class Launcher {
  public static void main(String[] args) {
    Persona p;

    p = new PersonaImpl("Mario");
    stampaDettagli(p);

    p = new StudenteLavoratore("Gianni", 123, 100);
    stampaDettagli(p);
}

  private static void stampaDettagli(Persona p) {
    System.out.println("Persona: " + p.getNome());

    if(p instanceof Studente)
      System.out.println(((Studente)p).getMatricola());

    if(p instanceof Lavoratore)
      System.out.println(((Lavoratore)p).getSalario());
  }
}
```


Visibilità

- Problema:
  - Abbiamo un'applicazione che fa da
    - Agenda
    - Calendario
    - …
  - Come facciamo a gestire i permessi sui vari dati?
    - Es. Chi può cambiare il nome di una persona?

Visibilità

```
public class Persona {
  private String nome;
  private Indirizzo ind;

  public String getNome() { return nome; }
  public void setNome(String n) { this.nome = n; }

  public Indirizzo getIndirizzo() { return ind; }
  public void setIndirizzo(Indirizzo ind) {
    this.ind = ind;
  }
}

public class Indirizzo {
  private String via;

  public String getVia() { return via; }
  public void setVia(String via) { this.via = via;}

}
```


Visibilità

```
public class Agenda {
  private Persona[] persone;

  public Persona cerca(String nome) {
    int i;
    //... cerca la persona con il nome indicato
    return persone[i];
  }
}

public class Calendario {
  private Agenda agenda;

  public void meeting(String nome1, String nome2) {
    Persona p1 = agenda.cerca(nome1);
    Persona p2 = agenda.cerca(nome2);
    //
    p1.setNome("Pippo");
    p1.getIndirizzo().setVia("noWhere");
  }
}
```

Visibilità

```
public interface PersonaReader {
  public String getNome();
  public IndirizzoReader getIndirizzo();
}

public interface IndirizzoReader {
  public String getVia();
}


public class Persona
  implements PersonaReader {
  […] // come prima!
}

public class Indirizzo
  implements IndirizzoReader {
  […] // come prima!
}
```


Visibilità

```
public class AgendaSafe {
  private Persona[] persone;

  public PersonaReader cerca(String nome) {
    int i;
    //... cerca la persona con il nome indicato
    return persone[i];
  }
}

public class CalendarioSafe {
  private AgendaSafe agenda;

  public void meeting(String nome1, String nome2) {
    PersonaReader p1 = agenda.cerca(nome1);
    PersonaReader p2 = agenda.cerca(nome2);
    // ...
    p1.setNome("Pippo");
    p1.getIndirizzo().setVia("noWhere");
  }
}
```

Visibilità

- L'applicazione è "sicura" ?
  - Siamo sicuri che il calendario non possa cambiare il nome?

```
Persona p1 = (Persona) agenda.cerca(nome1);
p1.setNome("Pippo");
```

  - In questo caso però è il calendario che sta barando…
- Conclusioni:
  - "giocare" in questo modo con le interfacce
    - permette di essere avvisati quando "qualcosa non va"
    - È comunque possibile "aggirarlo" (casting)
    - … e se la persona avesse un numero di carta di credito?

4. Polimorfismo

- Vogliamo realizzare un semplice sistema di scambio notizie
  - Utenti: sono interessati a ricevere Notizie
  - Giornalisti: producono Notizie
- Problema:
  - Ci possono essere nuovi giornalisti
  - Ci possono essere nuovi utenti
- Come progettare un sistema disaccoppiato?

4. Polimorfismo

```
public interface Utente {
  public void segnala(String notizia);
}

public class Giornale {
  private Utente[] utenti = new Utente[10];
  private int numUtenti = 0;

  public void registra(Utente u) {
    if(numUtenti < utenti.length) {
      utenti[numUtenti] = u;
      numUtenti++;
    }
  }

  public void pubblica(String notizia) {
    for (int i = 0; i < numUtenti; i++)
      utenti[i].segnala(notizia);
  }
}

public class Giornalista {
  private Giornale giornale;
  public Giornalista(Giornale giornale) { this.giornale = giornale; }

  public void pubblica() {
    giornale.pubblica("Notizia delle " + new java.util.Date());
} }
```

```
class UtenteTestuale implements Utente {
 public void segnala(String notizia) {
    System.out.println(notizia);
} }
```


4. Polimorfismo

```
public class Launcher {
  public static void main(String[] args) {
    Giornale giornale = new Giornale();

    Giornalista g1 = new Giornalista(giornale);
    Giornalista g2 = new Giornalista(giornale);

    Utente u1 = new UtenteTestuale();
    giornale.registra(u1);

    Utente u2 = new UtenteTestuale();
    giornale.registra(u2);

    g1.pubblica();
    g2.pubblica();
    g1.pubblica();
  }
}
```

5. Polimorfismo

```
public interface Intero {
  public void meth(int i);
}

public interface VirgolaMobile {
  public void meth(float f);
}

public class Tricky implements Intero, VirgolaMobile {

  public void meth(int i) {
    System.out.println("Versione parametro int: " + i);
  }

  public void meth(float f) {
    System.out.println("Versione parametro float: " + f);
  }
}
```


5. Polimorfismo

```
public class Launcher {
  public static void main(String[] args) {
    Tricky t = new Tricky();
    VirgolaMobile v = t;
    Intero i = t;

    t.meth((int)   1   );
    t.meth((float) 1.0);

    v.meth((int)   1   );
    v.meth((float) 1.0);

    i.meth((int)   1   );
    i.meth((float) 1.0);
  }
}
```