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

Esercitazioni di Ingegneria del Software

02: Java "in the large" & Information Hiding

1. Ereditarietà

```
public abstract class Animale {

   private String name;

   public Animale() {
        name = "no name";
        System.out.println("Sono un animale senza nome!");
   }

   public Animale(String name) {
        this.name = name;
        System.out.println("Sono un animale, e il mio nome è " +
   name);
   }

   public String getName() { return name; }

   public abstract String saluta();
   public abstract String saluta(String nomePersona);
}
```

1. Ereditarietà

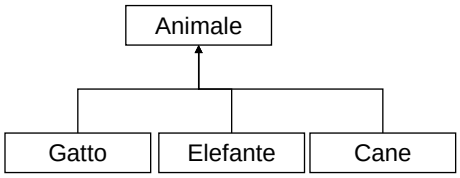
```
public class Gatto extends Animale {

    public Gatto() {
          super();
          System.out.println("Mi correggo, sono un gatto!");
    }

    public Gatto(String name) {
          super(name);
          System.out.println("Mi correggo, sono un gatto!");
    }

    @Override
    public String saluta() {
          return "Miao! Il mio nome è " + getName();
    }

    @Override
    public String saluta(String persona) {
          return "Miao! Ti saluto, " + persona + ", io sono " + getName();
    }
}
```


1. Ereditarietà

```
public class Cane extends Animale {

    public Cane() {
          super();
          System.out.println("Mi correggo, sono un cane!");
    }

    public Cane(String name) {
          super(name);
          System.out.println("Mi correggo, sono un cane!");
    }

    @Override
    public String saluta() {
          return "Bau! Il mio nome è " + getName();
    }

    @Override
    public String saluta(String persona) {
          return "Bau! Ti saluto, " + persona + ", io sono " + getName();
    }
}
```

1. Ereditarietà

- Gerarchia delle classi:



1. Ereditarietà

```
public class Foo {
    public static void main(String[] args) {
1       Animale a;

2       Gatto g1 = new Gatto();
        Gatto g2 = new Gatto("Felix");
3       Cane c = new Cane("Pluto");

4       a = g2;
        System.out.println(a.saluta());

        a = c;
        System.out.println(a.saluta());
    }}
```

Output:

1	Sono un animale senza nome! Mi correggo, sono un gatto!
2	Sono un animale, e il mio nome è Felix! Mi correggo, sono un gatto!
3	Sono un animale, e il mio nome è Pluto! Mi correggo, sono un cane!
4	Miao! Il mio nome è Felix Bau! Il mio nome è Pluto

```
/* costruttore di Gatto */
public Gatto(){
  super();
  syso("Mi correggo, sono un gatto!");
}

 /* costruttore di Animale*/
public Animale(){
  name = "no Name";
  syso("Sono un animale senza nome!");
}
```

1. Ereditarietà

```
class Foo {

   public static void incontraAnimale(Animale a, String uomo) {
        System.out.println("Animale: " + a.saluta());
        System.out.println("Uomo   : Il mio nome è " + uomo);
        System.out.println("Animale: " + a.saluta(uomo));
   }

}
```

- Overloading:
  - Quale metodo usiamo? (compile-time)
    - "saluta()" o "saluta(String)" ?
- Overriding:
  - Quale implementazione del metodo viene chiamata? (run-time)
    - Cane o Gatto? (o Elefante?)

1. Ereditarietà

- Problema:
  - dobbiamo "trasferire" gli animali con un TIR
  - rispettando un limite di peso massimo
- Soluzione (triviale):

```
public abstract class Animale {
    private float peso;
    public float getPeso() { return peso; }
    /* ... */
}
public class Tir {
    private float somma = 0;
    public boolean send(Animale a) {
       if(somma + a.getPeso() > 100) return false;
       else { somma += a.getPeso(); return true; }
    }}
```

1. Ereditarietà

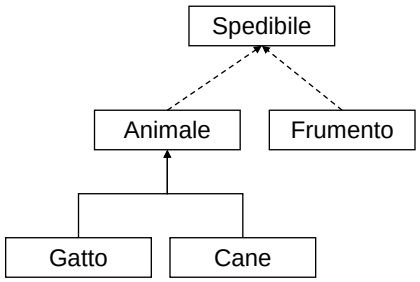
- Analisi della soluzione:
  - Pro: semplice
  - Contro: fatta ad hoc per spedire gli animali…
    - … e se dovessi spedire anche le gabbie? O il frumento?
- Proposta:
  - Definiamo le caratteristiche delle cose che possono essere spedite
    - in questo caso "principalmente" il peso
  - Creiamo un'interfaccia con questi metodi

1. Ereditarietà - Interfacce

```
public interface Spedibile {
   public float getPeso();
}
public abstract class Animale implements Spedibile {
   private float peso;
   public float getPeso() {  return peso; }
   /* ... */
}
public class Tir {
   private float somma = 0;
   public boolean send(Spedibile s) {
       if(somma + s.getPeso() > 100) return false;
       else { somma += s.getPeso(); return true; }
   }
}
```

1. Ereditarietà - Interfacce

- Gerarchia delle classi:



2. Trovare gli errori

Polimorfismo - Overriding

- Quali sono le istruzioni scorrette nel metodo main? Una volta eliminate tali istruzioni, cosa stampa il programma?
- Qual è il tipo statico e dinamico delle variabili al termine dell'esecuzione?

2. Trovare gli errori

Polimorfismo - Overriding

```
class Persona {
   String saluto() { return "Buongiorno";}
}
class PersonaEducata extends Persona {
   String saluto() { return "Buongiorno a lei";}
}
class PersonaMaleducata extends Persona {
   String saluto() { return "Faccia silenzio!"; }
}
class PersonaMaleducatissima extends PersonaMaleducata {
   String saluto() { return "tsé!"; }
}
```


2. Trovare gli errori

Polimorfismo - Overriding

```
class Prova {
 public static void main(String[] args) {
  Persona p = new Persona();
  PersonaEducata pe = new PersonaEducata();
  PersonaMaleducata pm = new PersonaMaleducata();
  PersonaMaleducatissima pmm = new PersonaMaleducatissima();

  p.saluto(); //1
  pe = p;     //2
  p = pe;     //3
  p.saluto(); //4
  pe = pm;    //5
  pe.saluto();//6
  pm.saluto();//7
  p = new PersonaMaleducata(); //8
  p.saluto(); //9
  pm = p;     //10
  pmm = (PersonaMaleducatissima) pm; //11
  pmm.saluto();//12
 }}
```

2. Soluzione

Polimorfismo - Overriding

- Le istruzioni scorrette sono 2 (p non ha come tipo statico una sottoclasse del tipo statico di pe), 5 e 10 (stesso motivo). Questi errori sono individuati a compile time.
- A runtime, invece, l'esecuzione dell'istruzione 11 solleva un'eccezione. Il codice diventa:

```
Persona p = new Persona();
PersonaEducata pe = new PersonaEducata();
PersonaMaleducata pm = new PersonaMaleducata();
PersonaMaleducatissima pmm = new PersonaMaleducatissima();
p.saluto();                          //1
p = pe;                              //3
p.saluto();                          //4
pe.saluto();                         //6
pm.saluto();                         //7
p = new PersonaMaleducata();         //8
p.saluto();                          //9
pmm = (PersonaMaleducatissima) pm; //11
pmm.saluto();                        //12
```


2. Soluzione

Polimorfismo - Overriding

Il programma stampa:
Buongiorno
Buongiorno a lei
Buongiorno a lei
Faccia silenzio!
Faccia silenzio!

- A questo punto l'esecuzione dell'istruzione 11 solleva un'eccezione, dal momento che il tipo dinamico di pm non e' PersonaMaleducatissima, e il programma termina.