Definizione di classi

- Definire un tipo di oggetto e la semantica che gli si vuole dare tramite
  - campi
    - come i campi di una struttura C
  - metodi
    - non esiste parallelo diretto in C
  - blocchi di inizializzazione
    - eseguiti al caricamento, prima di generare qualunque istanza della classe, una sola volta
- I nomi delle classi cominciano con la maiuscola

Classe DisneyCharacter

```
package it.unige.dist.laser.esempio;
public class DisneyCharacter
{
    int yob = -1;
    String name;
    protected DisneyCharacter(String theName, int theYob)
    {
        name = theName;
        yob = theYob;
    }
```

Classe DisneyCharacter

```
    public String isFunny()
    {
        return “tutti i personaggi Disney sono divertenti”;
    }
    public String toString()
    {
       return new String(name + “ “ + String.valueOf(yob));
    }
    static
    {
        System.out.println(“Hurra!”);
    }
}
```

Campi e tipi di campi

- campo di istanza
  - uno per ogni istanza della classe
  - tipo di default
- campo di classe
  - uno comune a tutte le istanze della classe
  - richiede la keyword “static”

Metodi e tipi di metodi

- metodo di istanza
  - uno per ogni istanza della classe
  - tipo di default
- metodo di classe
  - uno comune a tutte le istanze della classe
  - richiede la keyword “static”

Overloading dei metodi

- Si possono avere piu' metodi con lo stesso nome purche' la signature (numero e/o tipo dei parametri formali) sia diversa
- Il valore di ritorno non e' sufficiente a dstinguere

```
public class Esempio
{
    int value(int a) {};
    int value(long a) {};
    long value(int a, long b) {};
    long value(long a) {}; //errore al tempo di compilazione!
}
```

Ereditarieta’

- Una classe puo’ estendere un’altra classe
- Eredita cio’ che era gia’ definito nella classe madre
  - campi e metodi
- il processo e’ addittivo
  - a estende b estende c
  - implica che a contiene tutto cio’ che e’ in b e in a
- un tipo derivato e’ sempre anche di ogni tipo piu’ astratto, non e’ vero il contrario
  - a e’ di tipo a, di tipo b e di tipo c
  - b e’ di tipo b e di tipo c
  - c e’ solo di tipo c

Esempio classe derivata

```
package it.unige.dist.laser.esempio;
public class Pippo extends DisneyCharacter
{
	public pippo()
	{
		super(“Pippo”, 1932) // 25 maggio, per essere precisi
	}
	public String isFunny()
	{
	    return “tra i personaggi Disney Pippo e’ uno dei piu’ divertenti”;
	}
	static
	{
		System.out.println(“Yuk yuk!”);
	}
}
```

Classi astratte

- Una classe puo’ essere “astratta”, cioe’ definire la necessita’ di avere un tipo di comportamento al fine di appartenere ad una categoria logica

```
public abstract class Astratta
{
    public abstract int quanti();
}
```

  - non puo’ essere instanziata direttamente
  - deve essere estesa e la classe derivata deve implementare tutti i metodi astratti della classe base

Polimorfismo

- La capacita’ identificare solo a runtime l’effettivo comportamento del programma

```
Public static void main(String[] argv)
{
    Astratta a = new Concreta();
    Sstem.out.println(String.valueOf(a.quanti()));
}
```

Overriding dei metodi

- Si usa per implementare il polimorfismo
- I metodi in Java sono virtuali per default
- il metodo invocato sara’ sempre quello della classe piu’ derivata possibile purche’
  - il metodo invocato deve essere presente anche nella classe base
  - la signature del metodo deve essere la stessa in classe base e classe derivata INCLUSO IL TIPO RITORNATO
- non e’ possibile restringere l’accesso a un metodo derivato

Interfacce

- Guerra di religione Distributed OOP:
- Implementation Inheritance vs. Delegation and Containement
- Derivazione da classi base vs. Definizione di Interfacce e loro implementazione

Uso delle Interfacce

contratto tra utilizzatore e fornitore di servizi

permette di nascondere completamente l'implementazione e cristallizzare la modalita' di interazione

metodologia usata pesantemente in quasi tutti gli schemi di distributed OOP (e.g. CORBA, RMI e DCOM)

ATTENZIONE: contratto sintattico, non si dice nulla in proposito alla semantica!!!

Definire un’Interfaccia

```
public interface Button
{
    public String getName();
    public void press();
    public boolean isPressed();
}
```

Implementare un'Interfaccia

```
public class MyButton implements Button
{
    String name = "Innominato";
    boolean state = false;
    public MyButton(String theName)
    {
        name = theName;
    }
```

```
public String getName()
{
    return new String(“Il mio
      bottone si chiama “ + name);
}
```

```
    public void press()
    {
        state ^= true;
    }
    public boolean isPressed()
    {
        return state;
    }
}
```

Concetto e uso dei Package

- package == insieme di classi legate da un comune attributo
  - implementano un servizio
  - sono logicamente simili e.g. tutte le classi che implementano un formato di immagine
  - ...
  - stesso programmatore?
- sono l'unita' di visibilita' di default

Concetto e uso dei Package (Cont)

- I nomi dei package cominciano con la minuscola
- I nomi dei package dovrebbero seguire la convenzione contraria a quella degli URL

```
package it.unige.dist.laser.pippo;
package com.sun.pluto;

import java.util.*;
```