UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione 3
e
Laboratorio di Programmazione 3

Paradigmi della programmazione Object Oriented

Proff. Angelo Ciaramella – Emanuel Di Nardo

Paradigmi

- Paradigmi della programmazione ad oggetti
  - Incapsulamento
  - Ereditarietà
  - Polimorfismo
  - Astrazione
  - Riuso
  - …

Astrazione

- Astrazione
  - arte di sapersi concentrare solo sui dettagli veramente essenziali nella descrizione di un'entità

- Tre livelli di astrazione
  - astrazione funzionale
    - Implementazione di un metodo
  - astrazione dei dati
    - definiamo una classe, raccogliendo in essa solo le caratteristiche e le funzionalità essenziali degli oggetti che essa deve definire
  - astrazione del sistema
    - definiamo un'applicazione nei termini delle classi essenziali che devono soddisfare agli scopi dell'applicazione stessa

Riuso

- Riuso
  - conseguenza dell'astrazione e degli altri paradigmi della programmazione ad oggetti (incapsulamento, ereditarietà e polimorfismo)

Incapsulamento

- Incapsulamento
  - chiave della programmazione orientata agli oggetti
  - una classe riesce ad acquisire caratteristiche di robustezza, indipendenza e riusabilità

- Filosofia
  - *Accesso controllato ai dati* mediante metodi che possono prevenirne l'usura e la non correttezza
    - dichiarare privati gli attributi di una classe e quindi inaccessibili fuori dalla classe stessa
  - accesso ai dati potrà essere fornito da un'interfaccia pubblica

Incapsulamento

- Incapsulamento con codice Java consiste il più delle volte nel
  - dichiarare tutti dati privati
  - fornire alla classe metodi pubblici di tipo *mutator* "set" e accessor "*get*" per accedervi rispettivamente in scrittura e lettura

Incapsulamento

```
public class ContoBancario {
private String contoBancario = "5000000 di Euro";
private int codice = 1234;
private int codiceInserito;

public void setCodiceInserito(int cod){
codiceInserito = cod;
}
public int getCodiceInserito(){
return codiceInserito;
}
public String getContoBancario() {
if (codiceInserito == codice) {
return contoBancario;
}
else {
return "codice errato!!!";
}
}
}
```

setter

getter

Incapsulamento funzionale

- Un metodo diventa privato
  - potrà essere *invocato solo da un metodo definito nella stessa classe*, che potrebbe a sua volta essere dichiarato pubblico

```
public class ContoBancario {
. . .
public String getContoBancario(int codiceDaTestare){
return controllaCodice(codiceDaTestare);
}
private String controllaCodice(int codiceDaTestare) {
if (codiceInserito == codiceDaTestare) {
return contoBancario;
}
else {
return "codice errato!!!";
}}}
```


Incapsulamento funzionale

- L'incapsulamento
  - permette a due oggetti istanziati dalla stessa classe di accedere in modo pubblico ai rispettivi membri privati

Parametro implicito

- All'interno di un metodo possibile definire ed utilizzare nuove variabili
  - Queste variabili hanno scope locale alla funzione e spariscono una volta terminata la chiamata al metodo
  - Il JAVA permette di dichiarare variabili locali con lo stesso nome delle variabili membro della classe
    - le variabili membro della classe sono messe in ombra dalle nuove variabili

- L'accesso alla variabile membro invece che alla variabile locale viene risolto tramite l'uso del parametro implicito
  - this

Incapsulamento funzionale

```
public class Dipendente {
private String nome;
private int anni; //intendiamo età in anni
. . .
public String getNome() {
return nome;
}
public void setNome(String n) {
nome = n;
}
public int getAnni() {
return anni;
}
public void setAnni(int n) {
anni = n;
}
public int getDifferenzaAnni(Dipendente altro) {
return (anni - altro.anni);
}
}
```

```
altro.getAnni()
```


This

```
public class Cliente {
private String nome;
private String indirizzo;
private int numeroDiTelefono;

public Cliente(String nome, String indirizzo, String
numeroDiTelefono) {
this.setNome(nome);
this.setIndirizzo(indirizzo);
this.setNumeroDiTelefono(numeroDiTelefono);
}
public void setNumeroDiTelefono(String numeroDiTelefono) {
this.numeroDiTelefono = numeroDiTelefono;
}
```


This

```
public void setIndirizzo(String indirizzo) {
this.indirizzo = indirizzo;
}
public void setNome(String nome) {
this.nome = nome;
}
// . . .
}
```


Modificatori di accesso

- I modificatori di accesso sono
  - `public`
    - può essere utilizzato sia per un membro (attributo o metodo) di una classe, sia per una classe stessa
    - *accessibile* da una *qualsiasi classe situata in qualsiasi package*
    - per l'incapsulamento le variabili d'istanza non dovrebbero mai essere dichiarate pubbliche
  - `protected`
    - accessibile all'interno dello *stesso package* ed in tutte le sottoclassi* della classe in cui è definito, anche se non appartenenti allo stesso package
    - non è possibile dichiarare una classe protected

* Una sottoclasse è una classe che estende un'altra classe

Modificatori di accesso

- I modificatori di accesso sono
  - `Nessun modificatore`
    - accessibile *solo da classi appartenenti al package dove è definito*
  - `private`
    - restringe la *visibilità di un membro* di una classe alla classe stessa

Modificatori di accesso

MODIFICATORE	STESSA CLASSE	STESSO PACKAGE	SOTTOCLASSE	OVUNQUE
`public`	SI	SI	SI	SI
`protected`	SI	SI	SI	NO
nessun modificatore	SI	SI	NO	NO
`private`	SI	NO	NO	NO

Riepilogo dei modificatori di accesso

static

```
public class Counter {
private static int counter = 0;
private int number;

public Counter() {
counter++;
setNumber(counter);
}

public void setNumber(int number) {
this.number = number;
}

public int getNumber() {
return number;
}
}
```

Una variabile static e public può considerarsi come "variabile globale"

Incapsulamento static

```
public class IncapsulamentoStatico {
private static int attributoStatico = 0;

public static void setAttributoStatico(int
attributoStatico) {
IncapsulamentoStatico.attributoStatico = attributoStatico;
}

public static int getAttributoStatico() {
return attributoStatico;
}
}
```

I metodi statici non sono associati ad una istanza ma solo ad una classe (non potranno interagire con le variabili di istanza, ma solamente con quelle statiche)

Vedremo il Design Pattern Singleton che usa metodi static