UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Metodologie di Programmazione

Lezione 7: Incapsulamento e inizializzazione di default

Lezione 7: Sommario

UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Incapsulamento
- Modalità di accesso a campi e metodi
- Inizializzazione implicita dei campi

Incapsulament

UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Ma perché utilizziamo le parole chiave public e private?
- Per nascondere le informazioni ("information hiding") all'utente
- Strategia non solo informatica, ma più in generale ingegneristica
  - Ad esempio, considerate la centralina elettronica nelle moderne automobili vs. carburatore del passato (notoriamente complesso)
- Il processo che nasconde i dettagli realizzativi, rendendo pubblica un'interfaccia, prende il nome di incapsulamento
  - Dettagli realizzativi: campi e implementazione
  - Interfaccia pubblica: metodi pubblici

Perché incapsulare?

UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

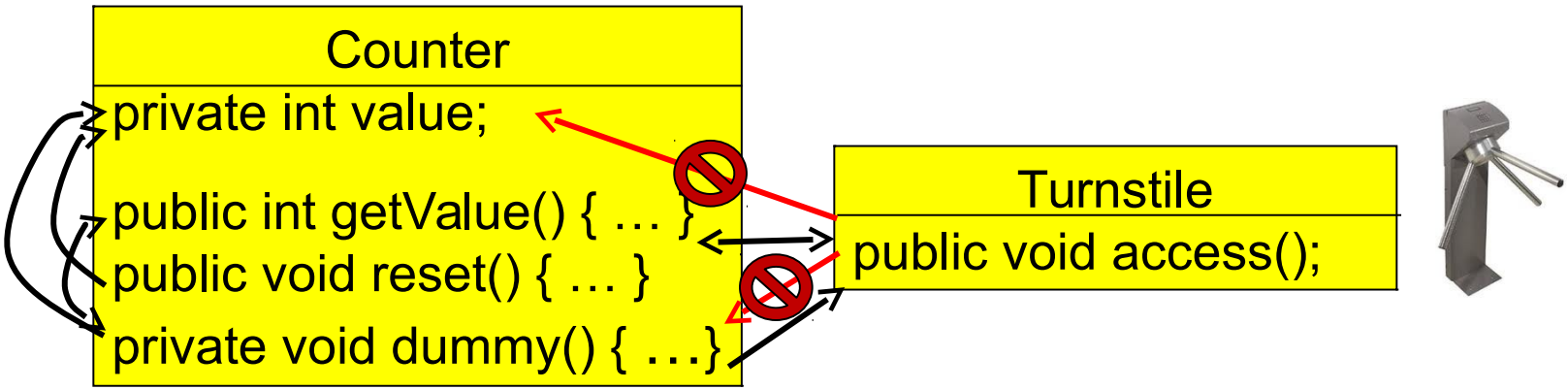
- Si semplifica e modularizza il lavoro di sviluppo assumendo un certo funzionamento a "scatola nera"
- Non è necessario sapere tutto, soprattutto molti inutili dettagli
- L'incapsulamento facilita il lavoro di gruppo e l'aggiornamento del codice (maintenance)
- Aiuta a rilevare errori: in presenza di moltissime classi, un certo errore si verifica solo in una determinata classe per cui ci si può concentrare su di essa

Come interagiscono le classi tra loro?

- Una classe interagisce con le altre **principalmente (=quasi solo)** attraverso i costruttori e i metodi pubblici
- Le altre classi non devono conoscere i dettagli implementativi di una classe per usarla in modo efficace

Accesso a campi e metodi (inclusi i costruttori)

- I campi e i metodi possono essere pubblici, privati (o protetti, come vedremo più in là)
- I metodi di una classe possono chiamare i metodi pubblici e privati della stessa classe
- I metodi di una classe possono chiamare **SOLO** i metodi pubblici di altre classi



Un esempio: realizzare un semplice menù

- Dovete realizzare la classe Menu, in grado di visualizzare un menù come questo:

1) Inizia il gioco
2) Carica gioco
3) Aiuto
4) Esci

Fase 1: identifica i metodi richiesti

- Aggiungere una nuova opzione
- Visualizzare il menù

Fase 2: specifica l'interfaccia pubblica

- Aggiungere una nuova opzione:

```
public void addOption(String option) { }
```

- Visualizzare il menù:

```
public void display() { }
```

- Costruire l'oggetto:
  - Costruttore con una prima opzione in input?
  - Costruttore vuoto?
- Meglio evitare casi speciali:

```
public Menu() { }
```

Fase 3: scrivere la documentazione per l'interfaccia pubblica


```
/**
 * Un menu' che viene visualizzato in una finestra di console
 * @author navigli
 *
 */
public class Menu
{
	/**
	 * Costruisce un menu' vuoto (senza opzioni)
	 */
	public Menu()
	{
	}

	/**
	 * Aggiunge un'opzione alla fine del menu'
	 * @param option l'opzione da aggiungere
	 */
	public void addOption(String option)
	{
	}

	/**
	 * Visualizza il menu' su console
	 */
	public void display()
	{
	}
}
```

Fase 4: identifica i campi

- In ogni momento, qual è lo stato di un oggetto di tipo Menu?

```
public class Menu
{
        private String menuText;
        private int optionCount;

        ...
}
```

Fase 5: implementa i metodi


```
public class Menu
{
	...

	/**
	 * Costruisce un menu' vuoto (senza opzioni)
	 */
	public Menu()
	{
		menuText = "";
		optionCount = 0;
	}

	/**
	 * Visualizza il menu' su console
	 */
	public void display()
	{
		System.out.println(menuText);
	}

	/**
	 * Aggiunge un'opzione alla fine del menu'
	 * @param option l'opzione da aggiungere
	 */
	public void addOption(String option)
	{
		optionCount++;
		menuText += optionCount + ") " + option + "\n";
	}
}
```

Fase 6: collauda la classe

- Scriviamo un'altra classe per "testare" (collaudare) la nostra:

```
public class MenuTest
{
	static public void main(String[] args)
	{
		Menu menu = new Menu();

		menu.addOption("Open new account");
		menu.addOption("Log into existing account");
		menu.addOption("Help");
		menu.addOption("Quit");

		menu.display();
	}
}
```

Inizializzazione implicita per i campi di classe

- Al momento della creazione dell'oggetto i campi di una classe sono inizializzati automaticamente

Tipo del campo	**Inizializzato implicitamente a**
int	0
float, double	0.0
char	'\0'
boolean	false
classe X	null

- ATTENZIONE: le inizializzazioni sono automatiche per i campi di una classe, ma **NON** per le variabili locali dei metodi

Esempio di inizializzazione implicita


```
public class Room
{
    /**
     * Contiene il numero di persone attualmente nella stanza
     */
    private int numberOfPeople;

    /**
     * Conta il numero di accessi alla stanza
     */
    private Counter accessCounter;

    public static void main(String[] args)
    {
        int n;
        Counter myCounter;

        System.out.println(n+","+myCounter);

        Room room = new Room();
        System.out.println(room.numberOfPeople);
        System.out.println(room.accessCounter);
    }
}
```

non inizializzato esplicitamente

idem…

Errore a tempo di compilazione: **variable *n* may not have been initialized**

Crea un nuovo oggetto in memoria contenente i campi *numberOfPeople* e *accessCounter* **inizializzati**

Stampa **0**

Stampa **null**

Esercizio: registratore di cassa

- Progettare una classe che costituisca un modello di registratore di cassa
- La classe deve consentire a un cassiere di inserire i prezzi di articoli e la quantità di denaro pagata dal cliente, calcolando il resto dovuto
- La classe fornisce i seguenti metodi:
  - Registra il prezzo di vendita per un articolo
  - Registra la somma di denaro pagata finora
  - Conclude la transazione, calcola il resto dovuto al cliente nel momento e restituendolo in uscita