Progettazione

Fase di progettazione di un'applicazione

Fase di implementazione

Prima di andare avanti

- Abbiamo appreso i concetti fondamentali di classe/oggetto/metodo
- Sappiamo scrivere semplici applicazioni
- Conosciamo i meccanismi di base di funzionamento del linguaggio Java
- D'ora in poi approfondiremo la programmazione intesa in senso classico, **ma prima di andare avanti**:
  - Focalizziamo l'attenzione sul processo di progettazione

Progettazione di un'applicazione

- Finora abbiamo visto semplici applicazioni in cui erano definite, al più, una classe "vera" e una classe test e che usavano un certo numero di classi definite
- In generale un'applicazione è composta da più classi che cooperano per realizzare ciò per cui l'applicazione è stata scritta
- Ma come si decide quali classi definire?

La programmazione a oggetti

- L'approccio della programmazione a oggetti antepone alla programmazione vera e propria una importante fase di **modellazione**
- All'inizio si osserva il **dominio del discorso** (termini tecnici, tipi di documenti, figure professionali, agenti/persone coinvolte, ...) dell'applicazione che si intende realizzare e l'**ambiente** in cui essa andrà ad operare (azienda, ufficio del magazzino, banca, biblioteca, ...)

Progettazione

- Lo studio del dominio dell'applicazione ha lo scopo di dare al progettista i mezzi e le conoscenze per scrivere un **modello** accurato del dominio in questione
- Nell'approccio attuale si hanno diversi strumenti per modellare diversi aspetti di un dominio di applicazione (statico, funzionale, dinamico, ...)
- Esistono linguaggi e formalismi creati appositamente per esprimere questi modelli

Ingegneria del software

- L'ingegneria del software è la branca dell'informatica che si occupa di definire metodologie, modelli e tecniche atte alla realizzazione di un buon software
- Nel corso di ingegneria del software verranno approfonditi gli aspetti cruciali della modellazione e verranno studiati formalismi adeguati (es: UML Unified Modeling Language, ...)

Per ora

- In questo corso introduttivo alla programmazione basterà accennare ad alcuni aspetti fondamentali
- Il concetto di classe è uno di questi
- Abbiamo introdotto intuitivamente il concetto di classe
- Nella modellazione di un dominio di applicazione le classi sono usate per modellare insiemi di **entità**

Entità

- Una entità nel dominio dell'applicazione rappresenta un oggetto, in senso lato, che può avere diversi **attributi**, essere in **relazione** con altre entità e compiere/essere_oggetto_di alcune **operazioni**
- A seconda del tipo di applicazione un'entità può essere molto concreta, oppure molto astratta: tutto dipende da ciò che si vuole modellare
- Esempi: automobili, conti bancari, persone, studenti, fatture, libri, schedari, finestre, pulsanti, riunioni, appuntamenti, componenti di un elettrodomestico, funzioni matematiche, matrici, fiori, ...

Entità

- Le entità individuate nel dominio di applicazione nella fase di progettazione diventano gli **oggetti** di una certa classe nell'implementazione dell'applicazione in un linguaggio orientato agli oggetti
- La definizione di una classe corrisponde alla formalizzazione delle caratteristiche comuni a certe entità che si sono individuate
- Gli attributi di queste entità diventano le variabili istanza degli oggetti della classe (ogni entità ha un certo insieme di attributi i cui valori la caratterizzano)

Operazioni

- Ogni entità, oltre a un insieme di attributi, può avere la capacità di compiere alcune operazioni o di effettuare dei calcoli e restituire dei risultati
- Queste caratteristiche sono modellate attraverso i metodi delle classi
- In questo tipo di modelli ogni tipo di operazione deve essere associata ad un certo tipo di entità, o a un insieme di entità

Relazioni

- Le entità sono in genere in relazione le une con le altre
- Entità dello stesso tipo hanno in genere relazioni dello stesso tipo con altre entità
- Queste relazione vengono tradotte, nel modello, in relazioni fra classi
- Esempio: relazioni “isa” (“è un”) fra sottoclassi e superclassi (ereditarietà) per avere a disposizione diverse astrazioni delle stesse entità
- Esempio: relazioni “use” (“usa”) fra classi che cooperano per realizzare una certa operazione che coinvolge diverse entità

Il progetto

- Usiamo i concetti di classe, oggetto, metodo, variabile istanza che abbiamo visto per definire **un modello del dominio dell'applicazione** che vogliamo scrivere
- La prima fase, quella di progettazione, si deve occupare di:
  - Individuare le classi di entità che servono
  - Definire l'insieme di attributi da tenere in considerazione per le entità di ogni classe
  - Definire le operazioni associate alle entità o alle classi

Il modello

- Quello che deve scaturire dalla progettazione è l'interfaccia pubblica delle classi che comporranno l'applicazione
- Le classi di identità sono le **classi**
- Gli attributi sono le **variabili istanza**
- Le operazioni sono i **metodi** da chiamare sugli oggetti o sulle classi (metodi statici che eseguono operazioni associate a tutta una classe più che a un singolo oggetto della classe)

Implementazione

- Una volta individuata questa struttura dell'applicazione non resta che iniziare la fase di implementazione:
  - Scrittura dei metodi
  - Definizione di classi private e/o uso di classi delle API necessarie all'implementazione
  - Gestione degli errori
  - ...
- Ma non si risolve sempre tutto con una sola passata!!!

Dall'implementazione alla progettazione

- Spesso, soprattutto per applicazioni di dimensione medio-grande, in fase di implementazione si possono scoprire errori di progettazione: entità non considerate, operazioni mal distribuite, attributi troppo scarni o troppo abbondanti, ...
- In questo caso si deve ritornare indietro a modificare il modello e poi si ritorna ad implementare
- Questo può accadere più volte se la modellazione non è stata svolta in maniera accurata
- Accade naturalmente quando si vuole ampliare/modificare l'applicazione

Esempio

- Supponiamo che il nostro dominio del discorso sia quello delle automobili
- Possiamo immaginare diverse applicazioni in questo dominio:
  - Gestione del parco auto di un concessionario
  - Archivio della motorizzazione
  - Programma di supporto ai meccanici per la riparazione
  - Archivio di un'associazione di appassionati di auto d'epoca
  - Documentazione e previsione dei consumi di carburante delle macchine di una comunità

Quali entità?

- È facile rendersi conto che ognuna di queste applicazioni convolge diverse entità, alcune in comune con le altre, altre caratteristiche di ognuna
- L'entità "automobile", comunque, dovrà essere individuata sicuramente in ognuna di esse
- Tuttavia ogni applicazione necessita di considerare **attributi** di diversa natura, a diversi livelli di dettaglio

Attributi

- L'archivio della motorizzazione focalizza l'attenzione solo su alcune delle caratteristiche possibili di un'automobile
- Il programma di assistenza per i meccanici avrà bisogno di una dettagliatissima descrizione tecnica di ogni automobile
- Il concessionario gestisce anche la "storia" di auto usate, i prezzi, le ordinazioni
- ....

Altre entità

- Oltre alle automobili le diverse applicazioni coinvolgono altre entità caratteristiche del loro contesto
- L'associazione gestirà "ritrovi", "soci", "rassegne stampa", ...
- Il programma di supporto per meccanici probabilmente avrà bisogno di "apparecchiature", "pezzi di ricambio", "fasi di smontaggio", ...
- .....

Esercizio

- Immaginare altre possibili applicazioni
- Quali entità si potrebbero individuare? Con quali attributi?
- Che tipo di operazioni sono caratteristiche di ogni applicazione?
- Come possono essere associate alle varie entità?
- Servono delle entità “tramite”?
- In che relazione sono le varie entità?

Esempio: gestione dei consumi

- Supponiamo di voler scrivere un'applicazione che gestisce i consumi di alcune automobili
- Dividiamo il lavoro in diverse **fasi** dalla progettazione all'implementazione
- Per prima cosa dobbiamo individuare le entità che ci servono e definire quindi le classi

Fase 1: definizione delle classi

- Sicuramente avremo una classe Car che rappresenta le automobili
- Per ognuna, visto l'intento dell'applicazione, ci serviranno pochi dati: giusto quelli per identificarla univocamente e sapere a chi appartiene
- Vista la semplicità dell'applicazione sembrerebbe che non ci sia bisogno di considerare altre classi
- In ogni caso possiamo sempre aggiungerne

Fase 1: definizione delle classi

- Abbiamo visto gli attributi, vediamo le operazioni
- Per ogni auto vorremmo essere in grado di eseguire le seguenti operazioni:
  - Aggiungere del carburante
  - Percorrere una certa distanza
  - Conoscere quanto carburante è rimasto nel serbatoio

Fase 2: Assegnamo i nomi ai metodi

- Definiamo i nomi ai metodi e facciamo degli esempi di applicazione ad un certo oggetto:
- `Car myFiat = new Car(...); /* non sappiamo ancora che tipo di costruttori abbiamo */`
- `myFiat.addGas(20);`
- `myFiat.drive(100);`
- `myFiat.getGas();`
- `myFiat.getDescription();`

Fase 3: interfaccia pubblica

- Scriviamo la documentazione per l'interfaccia pubblica:

```
/** Un'automobile può percorrere una
  certa distanza e consumare carburante
*/
public class Car {
  /** Aggiunge carburante al serbatoio.
      @param amount la quantità in litri
  */
  public void addGas(double amount) {
  }
```

Fase 3: interfaccia pubblica

```
/** Percorre una distanza
   consumando carburante.
   @param distance distanza in km
*/
public void drive(double
                  distance) {}
/** Ispeziona la quantità di
  carburante rimasta nel
  serbatoio
```

Fase 3: interfaccia pubblica

```
    @returns la quantità rimasta in
    litri
  */
  public double getGas() {
  }
  /** Dà una descrizione dell'auto
      @returns una stringa di
               descrizione
  */
  public String getDescription() {}
}
```

Fase 4: variabili istanza

- Chiediamoci quali variabili istanza possono servire affinché ogni oggetto della classe Car possa autonomamente (cioè con il suo stato e con i parametri dei metodi) eseguire tutti i metodi che possono essere chiamati su di lui
- Innanzitutto, per la descrizione, inseriamo i dati minimi necessari all'identificazione della macchina e del proprietario
- Inoltre dobbiamo sicuramente tener traccia del carburante presente nel serbatoio

Fase 4: variabili istanza

```
private String owner;
private String registrationNum;
private double gas;
```

- Ma non basta! Infatti con queste informazioni non riusciamo a prevedere, dopo una certa distanza percorsa, quanto carburante sarà stato consumato
- Abbiamo bisogno di inserire nello stato l'informazione decisiva: i km percorsi per litro

Fase 4: variabili istanza

```
private double efficiency;
```

- Questo numero ci permette di decrementare della giusta quantità il carburante presente nel serbatoio dopo una certa distanza percorsa
- Carburante consumato = distanza percorsa / efficiency

Fase 5: identificare i costruttori

- A volte basta quello di default che mette tutte le variabili istanza ai valori di default
- Ma in molti casi è bene definire costruttori specifici
- In questo caso è necessario inizializzare ogni oggetto con i valori per il nome del proprietario, la targa e i km per litro
- Per il carburante possiamo considerare zero come plausibile valore iniziale visto che il carburante può essere inserito con il metodo apposito
- Tuttavia possiamo definire un altro costruttore che specifichi anche una quantità di carburante iniziale

Fase 5: identificare i costruttori

```
/** Costruisce un'automobile con dati
  descrittivi e efficienza assegnati.
  Carburante iniziale = 0
  @param anOwner nome del proprietario
  @param aRegistrationNum numero di targa
  @param anEfficiency km percorsi con un litro
                          di carburante
*/
public Car(String anOwner, String
  aRegistrationNum, double anEfficiency) {
}
```

Fase 5: identificare i costruttori

```
/** Costruisce un'automobile con dati
  descrittivi, efficienza e carburante iniziale
  assegnati.
  @param anOwner nome del proprietario
  @param aRegistrationNum numero di targa
  @param anEfficiency km percorsi con un litro
                       di carburante
  @param initialGas litri di carburante
                       iniziale
*/
public Car(String anOwner, String
          aRegistrationNum, double anEfficiency,
          double initialGas) {
}
```

Fase 6: realizzare i metodi

- Farlo per esercizio.
- Cominciare dai più semplici.
- Se ci si accorge che ci sono problemi si può tornare in una delle fasi precedenti (soprattutto quelle iniziali) e cercare di risolverli
- Compilare tutto e correggere tutti gli errori di compilazione

Fase 7: collaudare la classe

- Scrivere una classe test *oppure*
- Caricare la classe su Bluej ed eseguire i test
- Es:

```
Car paperCar = new Car("Paperino",
  "313", 20);
paperCar.addGas(20);
paperCar.drive(100);
Double gasLeft = paperCar.getGas();
System.out.println("Rimasti " + gasLeft
  + " litri");
```