Java

Capitolo 3 – OOP Base

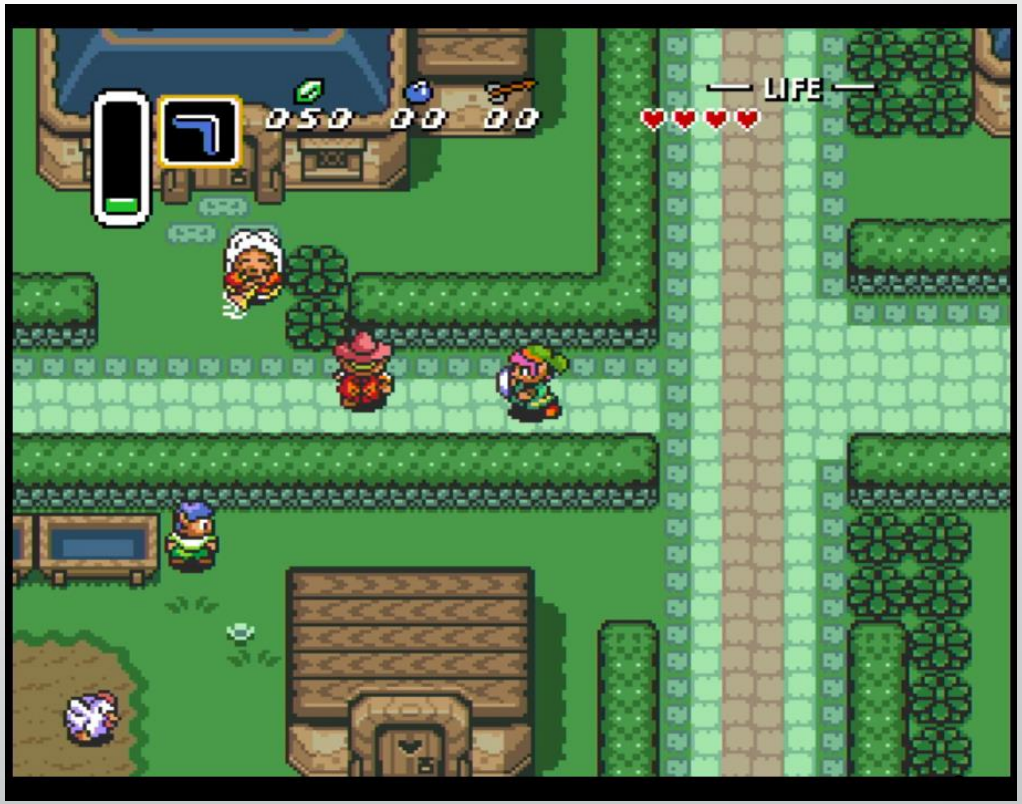
Prof. Ivan Gentile

Programmazione Orientata agli Oggetti

- **OOP = Oject Oriented Programming**
- Si parte dagli «oggetti» che dobbiamo rappresentare
- E poi si fanno relazioni tra loro
- Strategia *bottom-up*

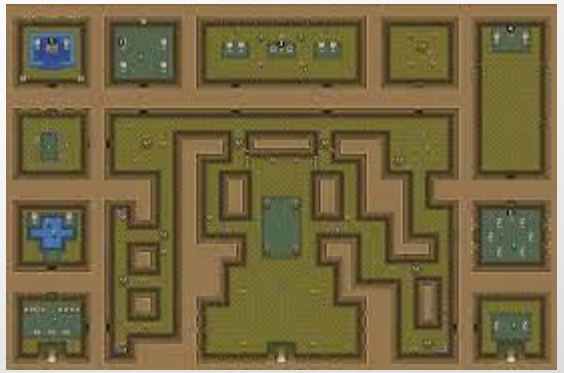
Esempio

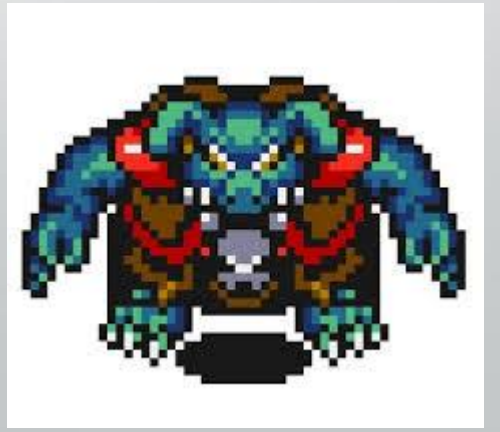
- Devo realizzare un Videogame
  - Un personaggio che si può creare con certe caratteristiche e che si muove in una mappa per raggiungere un certo obiettivo
- Immaginate tutte gli if, i cicli etc. da fare
  - Complicatissimo



Esempio

- Ragioniamo quali oggetti ci sono nel nostro videogame
- Personaggio
- Mappa
- Nemici
- Arma
- …
- Poi dipende da quanto accurato vogliamo farlo





Modelliamo ogni oggetto

- Cosa lo caratterizza?
  - Aspetto
  - Forza
  - Punti Vita
  - …
- Cosa può fare e cosa gli si può fare?
  - Camminare
  - Volare
  - Può infliggere danni
  - Può subire danni
  - ---

Videogioco

- Si tratterà di specificare come questi oggetti interagiscono

Elementi della OOP

- Classe
  - Analoga al tipo
  - Aggiungiamo quindi alla struttura le operazioni tra esse
  - Es: Studenti
- Oggetto: istanza di una classe
  - Es: lo studente Rossi Mario
- Ereditarietà
- Polimorfismo
- Binding

Classe

- Conto Corrente
- Un generico conto corrente
- Cosa caratterizza un conto corrente?
  - Attributi o Proprietà
  - Quantità, Proprietario, …
- Cosa posso fare su un conto corrente?
  - Creare un conto corrente (o chiudere)
  - Prelevare denaro
  - Aggiungere denaro
  - Cambiare proprietari
  - Conoscere l'importo
  - …

Classe Conto

Information Hiding
Attributi privato
Metodi pubblici
(salvo casi particolari)

Costruttore

Metodi getter

Metodi setter

```
2  public class Conto {
3      // Attributi
4      private double amount;
5      private String owner;
6
7      /* METODI */
8      // COTRUTTORE
9      public Conto(String proprietario, double quantitaIniziale) {
13
14     public double getAmount() {return amount;}
15     public String getOwner() {return owner;}
16
17     public void setAmount(double q) {
25
26     public void setOwner(String o) {
29
30
31     public void versamento(double importo) {
34
35     /* FINE METODI */
36
37 }
```

Proviamo la classe

```
2  public class ProvaConto {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6
7          Conto c1;
8          c1 = new Conto("Mario", 100.0);
9
10         Conto c2 = new Conto ("Luigi", 50.00);
11
```

```
15         c1.setOwner("Ivan");
16         System.out.println(c1.getAmount());
17         System.out.println(c1.getOwner());
```

Metodi Getter e Setter perché?

- **Incapsulamento e astrazione**
  - Nascondo i dettagli dell'implementazione (information hiding)
- **Immutabilità**
  - Posso cambiare nomi e tipo degli attributi lasciando la stessa interfaccia
- Validazione, error handling, e logica in generale

```
public void setAmount(double q) {
    if(q >= 0) {
        amount = q;
    }
    else {
        System.out.println("Errore quantità inserita < 0");
    }
}
```

Costruttori

- Viene chiamato nel momento in cui si istanzia un oggetto
  - `c = new Conto(…)`
- Possono essere **più di uno (overloading)**
  - purché a firma diversa
- Stesso nome della classe
- Nessun tipo restituito (nemmeno void)
- Si può anche non definire alcun costruttore
  - In tal caso ne crea JAVA uno di default che istanzia solo gli attributi

Stampa di un oggetto

- `System.out.println(ogetto);`
- Stampa il reference dell'oggetto
- Allora per convenzione si crea un metodo `toString`
  - restituisce una stringa
- Si può chiamare in uno dei seguenti modi equivalente
  - `System.out.println(ogetto.toString());`
  - `System.out.println(ogetto);`

this

- Oltre ai parametri espliciti esiste anche un parametro implicito
  - Il puntatore **this**
  - è il puntatore all'oggetto stesso (a cui il metodo appartiene)
  - i membri si accedono con il punto
  - La parola chiave this è usabile solo all'interno del corpo di un metodo
- usato
  - per esempio quando vogliamo assegnare a una variaible membroo un parametro che ha il suo stesso nome
  - per passare l'oggetto stesso (visto che in Java tutto è classe) a metodo per esempio scrivendo `System.out.println(this);`

Costruttore di Copia

- Copia un oggetto
- Ne crea un clone
  - O meglio dovrebbe
  - Sta a noi realizzarlo bene
  - Con i tipi primitivi e Stringhe non abbiamo problemi
  - Con i tipi reference bisogna stare attenti

```
// Costruttore di copia
public Conto(Conto c) {
    this.owner = c.owner;
    this.amount = c.amount;
}
```

```
Conto c1;
c1 = new Conto("Mario", 100.0);
Conto c3 = new Conto(c1);
```

Altro

- Vedere i video di Skill Factory
  - Java Object Oriented: introduzione al paradigma Object Oriented - Lezione 20 (fino al minuto 5): https://youtu.be/uWhjYQSro5Y
    - Dal minuto 5 in poi (Model View Controller), facoltativo
  - Java Object Oriented: classi ed oggetti - Lezione 16: https://youtu.be/YprrPpZlhFs
  - Java Object Oriented: come si dichiarano i metodi - Lezione 11: https://youtu.be/JIkM-FVTn_k