UNIVERSITÀ
DEGLI STUDI
DI PALERMO

Object Design

Ingegneria del Software

Object Design: una panoramica

- Concettualmente, lo sviluppo di sistema di software colma il gap tra un dato problema ed un computer.
- Le attività di sviluppo di sistema chiudono incrementalmente questo gap identificando e definendo oggetti che realizzano parte del sistema
- L'analisi riduce il gap tra il problema ed il computer identificando gli oggetti che rappresentano concetti del problema specifico
- Durante l'analisi il sistema è descritto in termini di comportamento esterno
  - Le sue funzionalità - i casi d'uso
  - Concetti del dominio che vengono manipolati - il modello degli oggetti
  - Il suo comportamento in termini di interazioni
  - I requisiti non funzionali

Object Design: una panoramica

- Durante l'object design noi rifiniamo/raffiniamo I modelli di analisi e del system design
  - Aggiungiamo tipo e la visibilità delle informazioni
- Le attività dell'Object Design includono (tra le altre):
  - Identificare attributi ed operazioni mancanti
  - specificare tipi e visibilità
  - specificare vincoli
  - specificare eccezioni
  - identificare ed modificare librerie di classi

Identificare attributi e operazioni

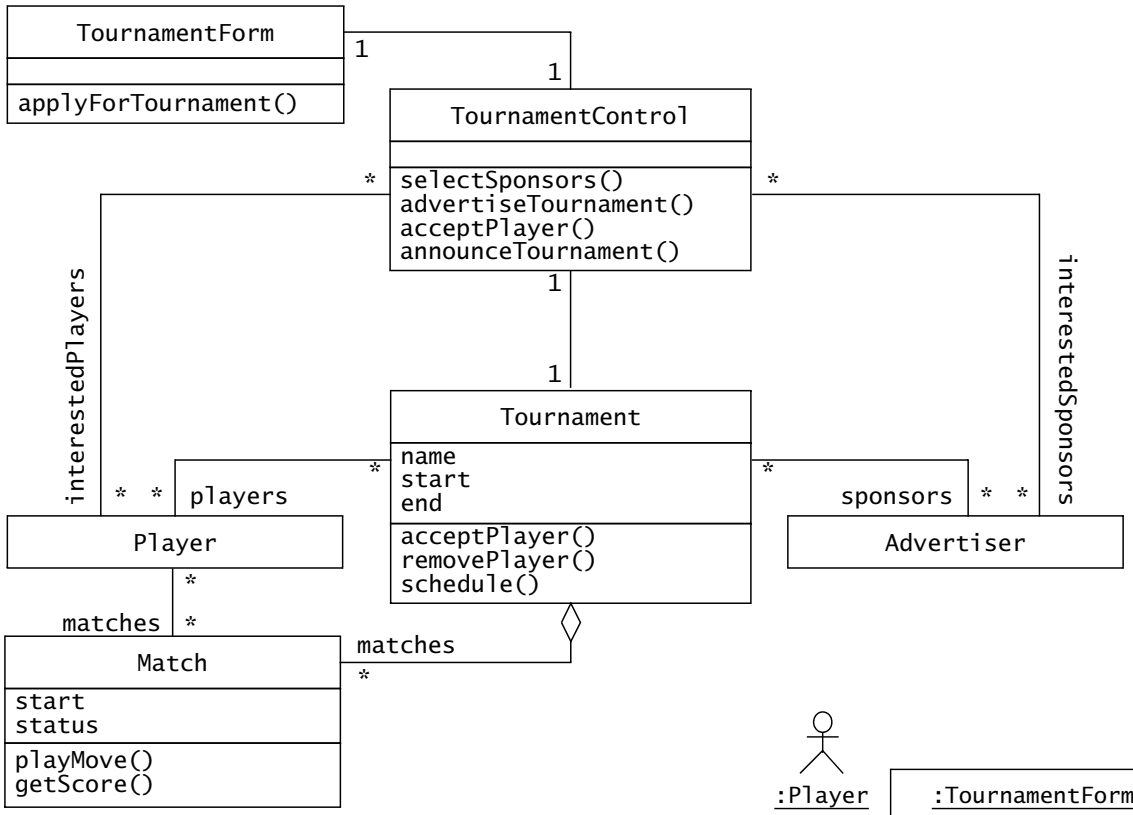
- L'elemento principale che si tiene in considerazione è il modello dinamico - sequence diagram

Specificare type, signature and visibility

- Per ogni attributo si specificano il tipo e la visibilità
- Per ogni metodo la signature (equivalente di prototipo per C++)
- Si tengono in considerazione eventuali altri attributi presenti nelle classi delle librerie che si vogliono usare

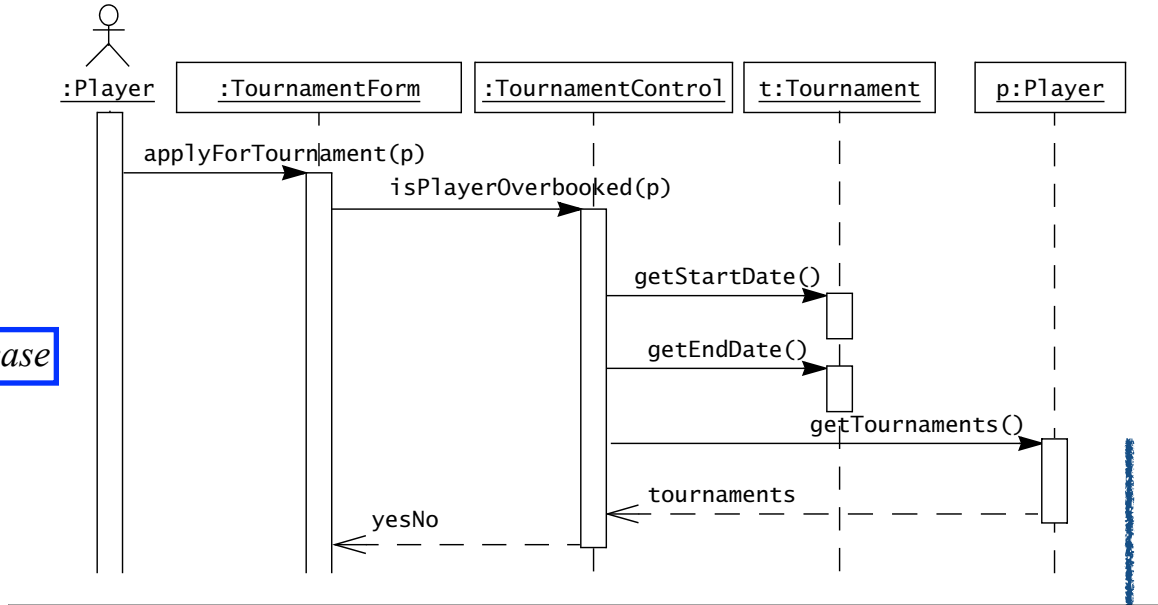
Tournament
-maxNumPlayers:int
+getMaxNumPlayers():int +getPlayers():List +acceptPlayer(p:Player) +removePlayer(p:Player) +isPlayerAccepted(p:Player):boolean

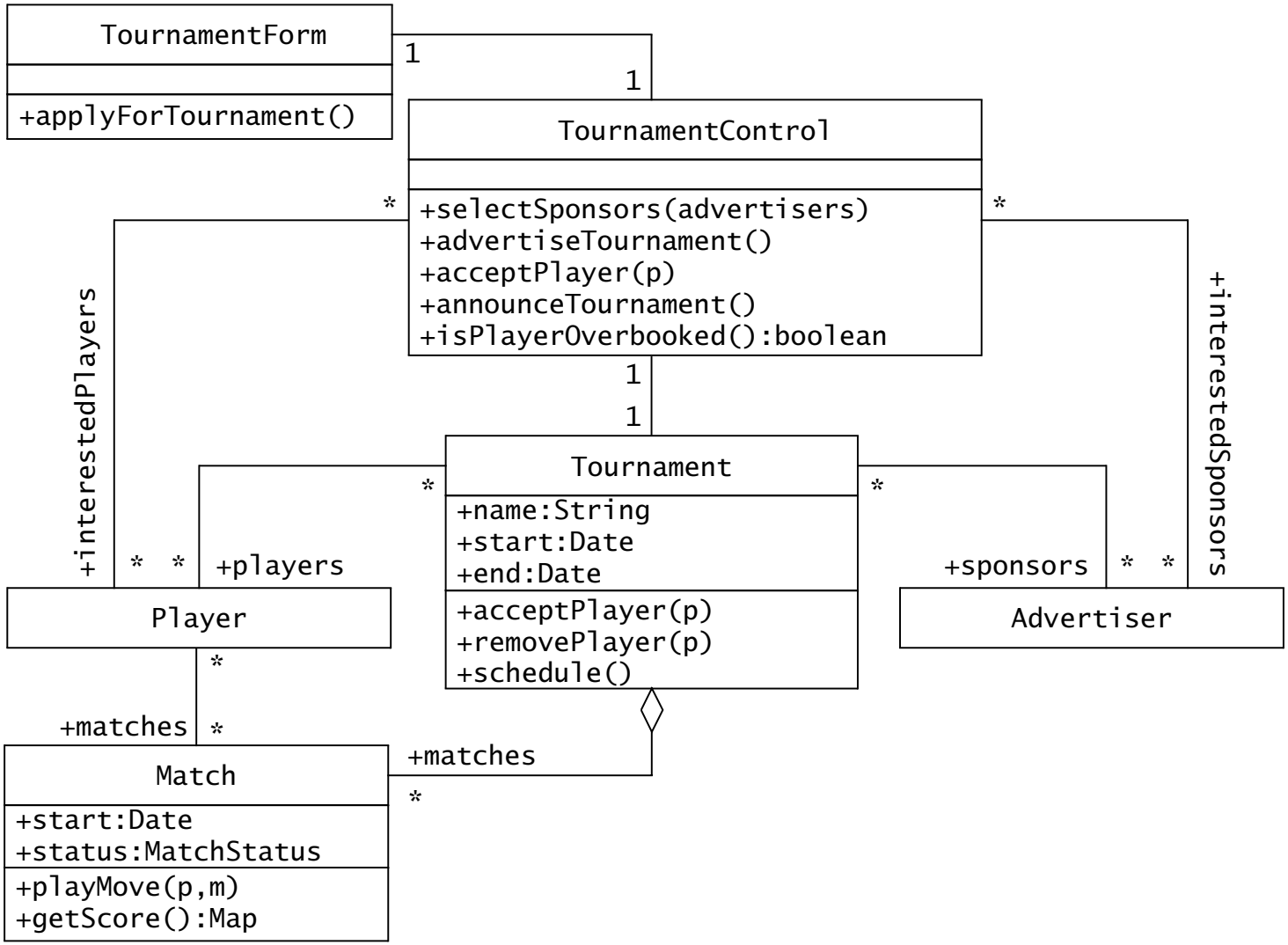
```
public class Tournament {
    private int maxNumPlayers;
    /* Other fields omitted */

    public Tournament(League l, int maxNumPlayers)
    public int getMaxNumPlayers() {…};
    public List getPlayers() {…};
    public void acceptPlayer(Player p) {…};
    public void removePlayer(Player p) {…};
    public boolean isPlayerAccepted(Player p) {…};

    /* Other methods omitted */
}
```



AnnounceTournment use case

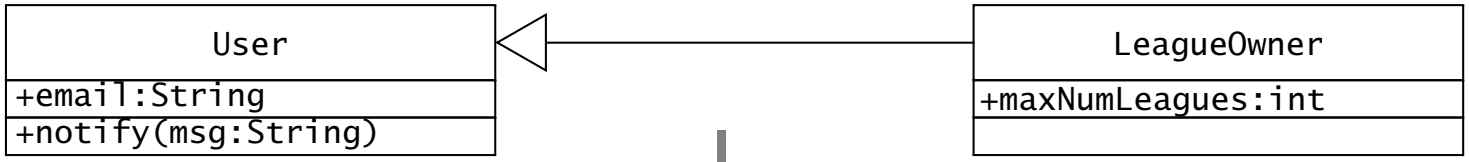
ApplyForTournment use case




TournamentForm
+applyForTournament()
1
1
TournamentControl
+selectSponsors(advertisers)
+advertiseTournament()
+acceptPlayer(p)
+announceTournament()
+isPlayerOverbooked():boolean
*
*
+interestedPlayers
+interestedSponsors
1
1
Tournament
+name:String
+start:Date
+end:Date
+acceptPlayer(p)
+removePlayer(p)
+schedule()
*
*
* *
+players
+sponsors
* *
Player
Advertiser
*
+matches
*
+matches
*
Match
+start:Date
+status:MatchStatus
+playMove(p,m)
+getScore():Map

Tradurre le relazioni

Object design model before transformation



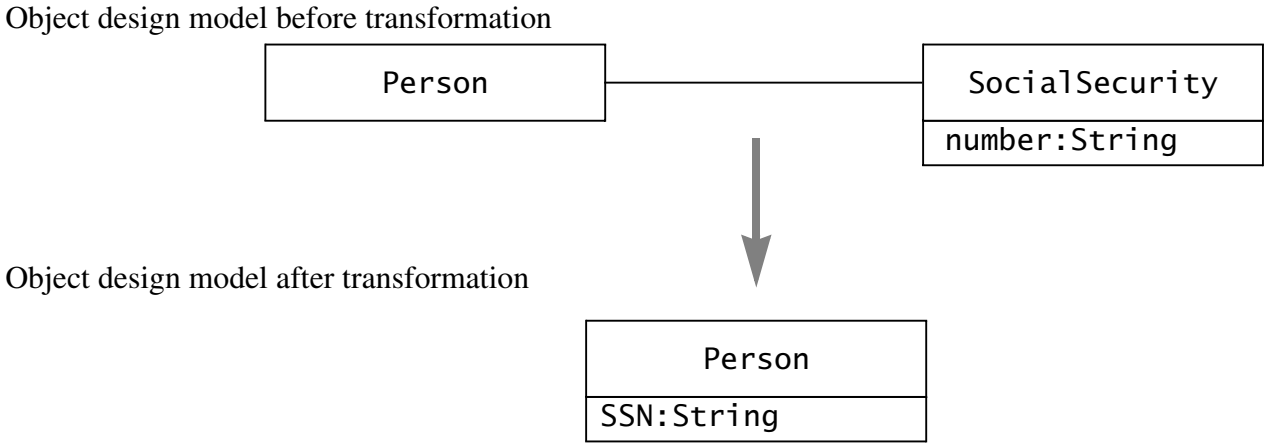
Source code after transformation

```
public class User {
    private String email;
    public String getEmail() {
        return email;
    }
    public void setEmail(String
value){
        email = value;
    }
    public void notify(String msg) {
        // ....
    }
    /* Other methods omitted */
}
```

```
public class LeagueOwner extends User
{
    private int maxNumLeagues;
    public int getMaxNumLeagues() {
        return maxNumLeagues;
    }
    public void setMaxNumLeagues
                        (int value) {
        maxNumLeagues = value;
    }
    /* Other methods omitted */
}
```

Tradurre le relazioni

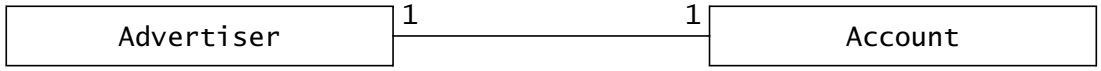
Collapsing Objects



Collapsing an object without interesting behavior into an attribute (UML class diagram).

Tradurre le relazioni

Object design model before transformation



Source code after transformation

```
public class Advertiser {
    private Account account;
    public Advertiser() {
        account = new Account();
    }
    public Account getAccount() {
        return account;
    }
}
```

Object design model before transformation

Source code after transformation

```
public class Advertiser {
    /* The account field is initialized
     * in the constructor and never
     * modified. */
    private Account account;

    public Advertiser() {
        account = new Account(this);
    }
    public Account getAccount() {
        return account;
    }
}
```

```
public class Account {
    /* The owner field is initialized
     * during the constructor and
     * never modified. */
    private Advertiser owner;

    public Account(Advertiser owner) {
        this.owner = owner;
    }
    public Advertiser getOwner() {
        return owner;
    }
}
```

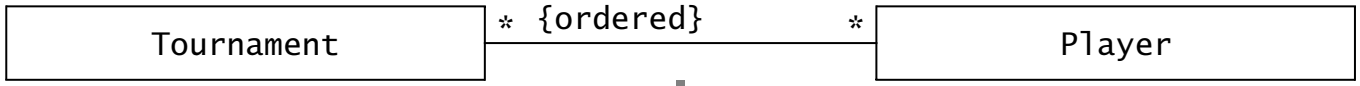
Object design model before transformation

Source code after transformation

```
public class Advertiser {
    private Set accounts;
    public Advertiser() {
        accounts = new HashSet();
    }
    public void addAccount(Account a) {
        accounts.add(a);
        a.setOwner(this);
    }
    public void removeAccount
            (Account a) {
        accounts.remove(a);
        a.setOwner(null);
    }
}
```

```
public class Account {
    private Advertiser owner;
    public void setOwner
            (Advertiser newOwner) {
        if (owner != newOwner) {
            Advertiser old = owner;
            owner = newOwner;
            if (newOwner != null)
              newOwner.addAccount(this);
            if (old != null)
              old.removeAccount(this);
        }
    }
}
```

Object design model before transformation



Source code after transformation

```
public class Tournament {
    private List players;
    public Tournament() {
        players = new ArrayList();
    }
    public void addPlayer(Player p) {
        if (!players.contains(p)) {
            players.add(p);
            p.addTournament(this);
        }
    }
}
```

```
public class Player {
    private List tournaments;
    public Player() {
        tournaments = new ArrayList();
    }
    public void addTournament
            (Tournament t) {
        if (!tournaments.contains(t)) {
            tournaments.add(t);
            t.addPlayer(this);
        }
    }
}
```