

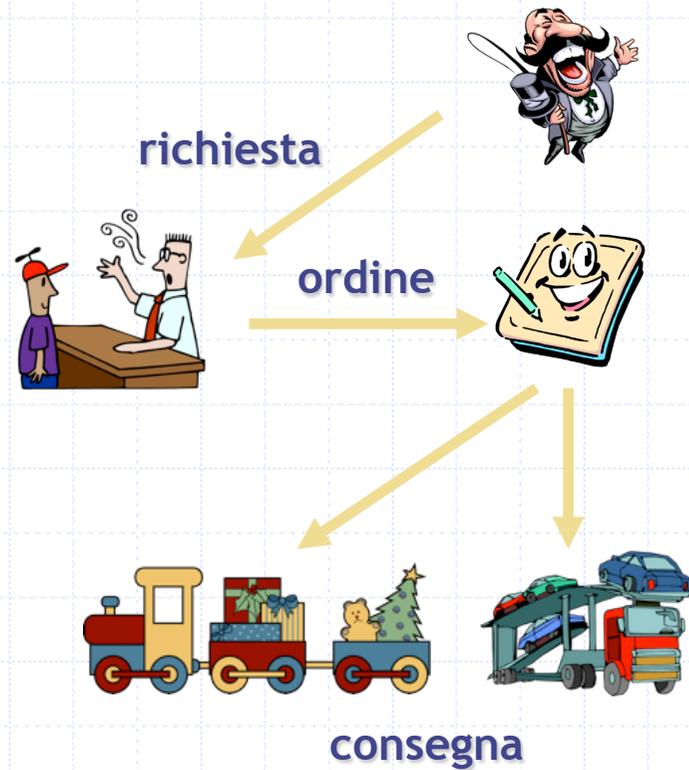


# Unified Modeling Language (UML): una breve introduzione

slides adattate da Gian Pietro Picco



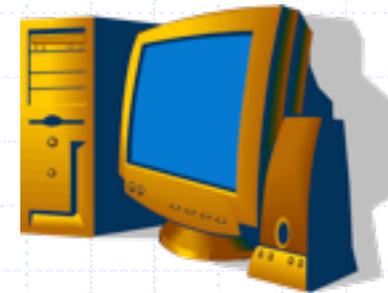
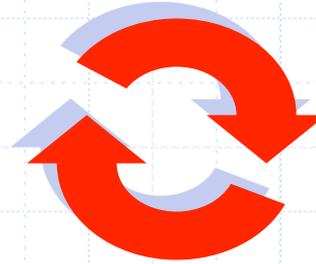
# Modellazione visuale



**Business Process**

“Un modello cattura le parti essenziali di un sistema”

James Rumbaugh



**Computer System**

# Perché UML

È il linguaggio visuale standard (OMG) per definire, progettare, realizzare e documentare i sistemi software ad oggetti

Riunisce molte proposte esistenti (Booch, Rumbaugh e Jacobson)

Copre l'intero processo di produzione

È sponsorizzato dalle maggiori industrie produttrici di software

# Perché UML

Riunisce aspetti dell'ingegneria del software, delle basi di dati e della progettazione di sistemi

È indipendente da qualsiasi linguaggio di programmazione

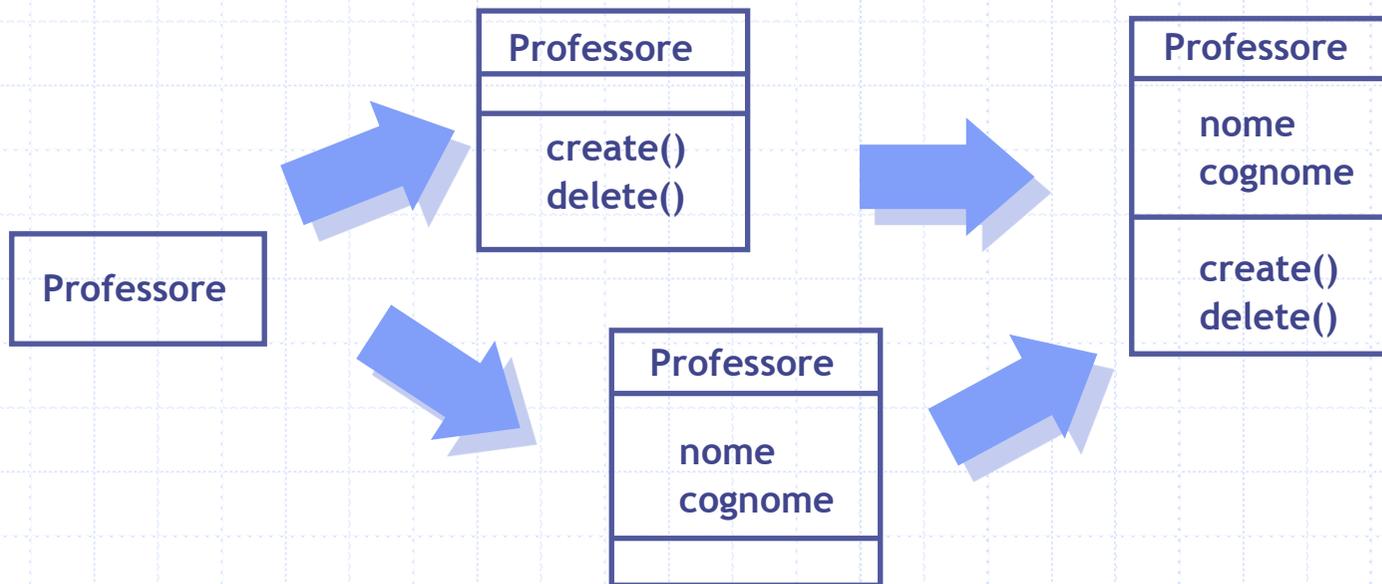
È utilizzabile in domini applicativi diversi e per progetti di diverse dimensioni

È estendibile per modellare meglio le diverse realtà applicative

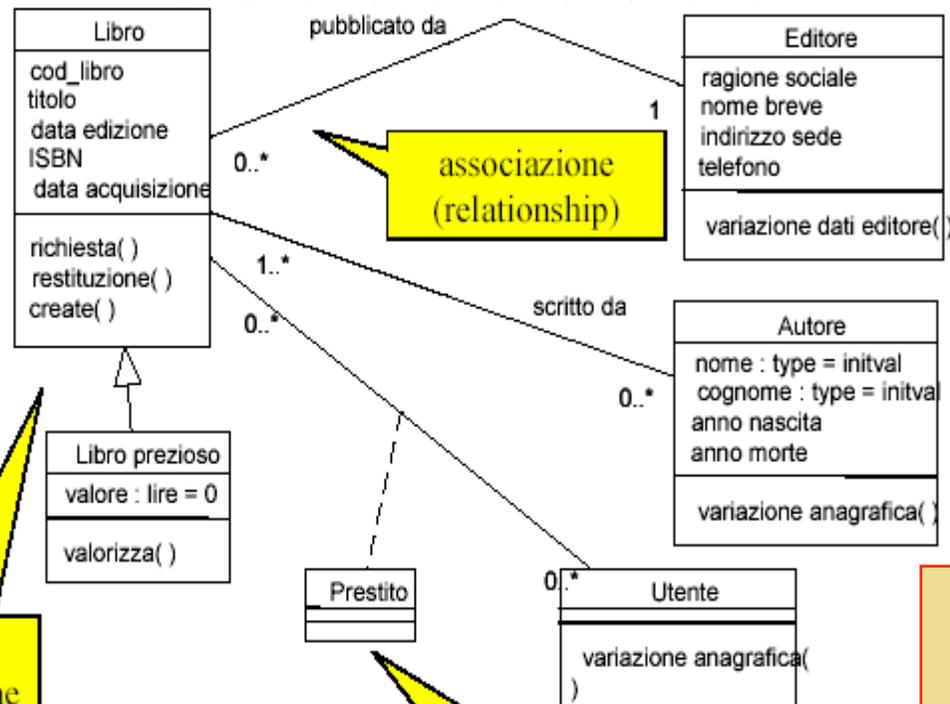
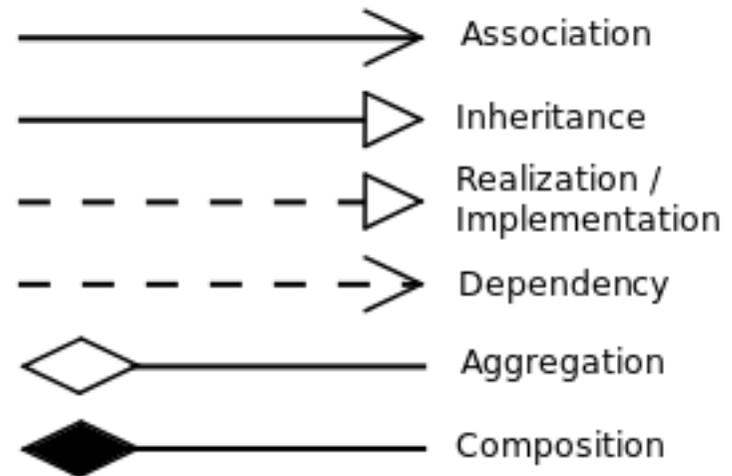
# Classe

In UML è composta da tre parti

- nome
- attributi (lo stato)
- metodi (il comportamento)



# Relazioni



gerarchia di specializzazione (superclasse-sottoclasse)  
comai@acm.org

classe associativa

Disegno ripreso da:  
**Adriano Comai**  
[http://www.analisi-disegno.com/a\\_comai/corsi/sk\\_uml.htm](http://www.analisi-disegno.com/a_comai/corsi/sk_uml.htm)

# Sequence diagram

Evidenziano la sequenza temporale delle azioni

Non si vedono le associazioni tra oggetti

Usabili in due forme diverse

- generica: tutte le sequenze (esecuzione) possibili
- istanza: una sequenza particolare, consistente con quella generica

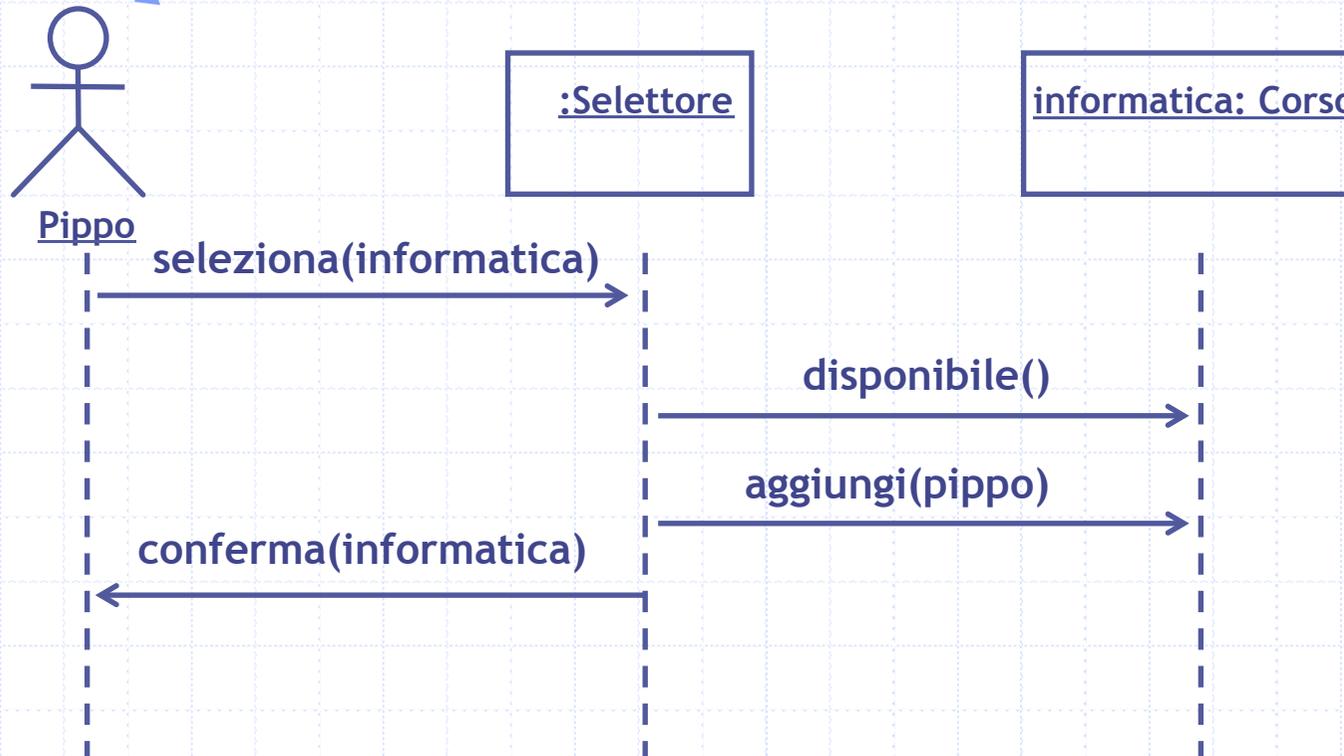
# Esempio (istanza)



# Esempio

(raffinamento diagramma precedente)

rappresenta un attore esterno



# Un esempio riassuntivo, parte 2



**Tombola!**

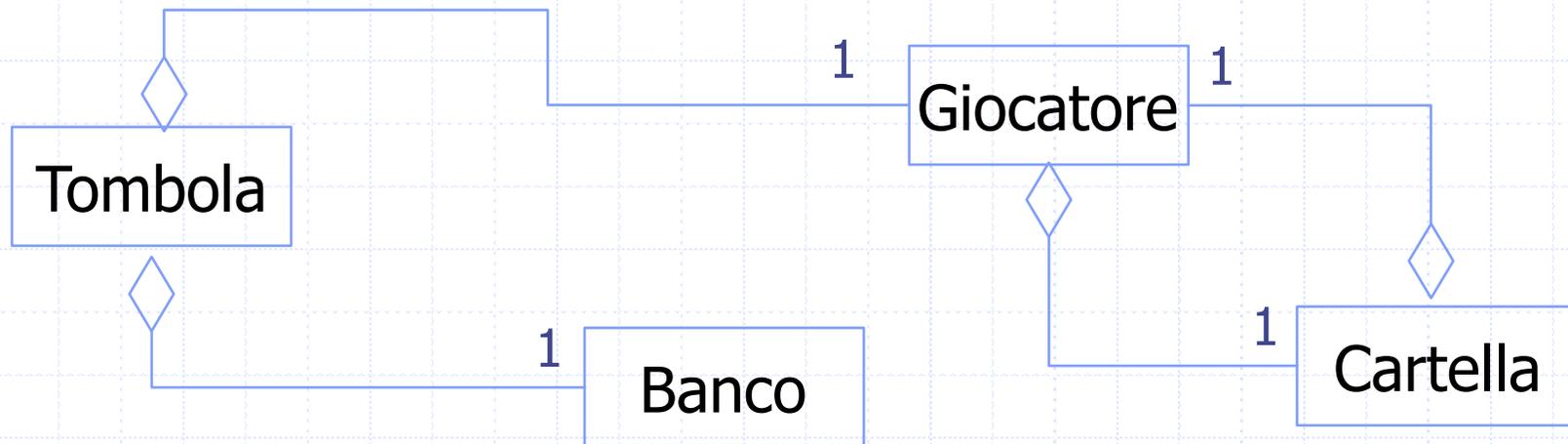
	17	22		45		66		83
1	19		30	48				70
		29	37		53		74	86
		58	31		23		14	89

# Tombola - esercizio

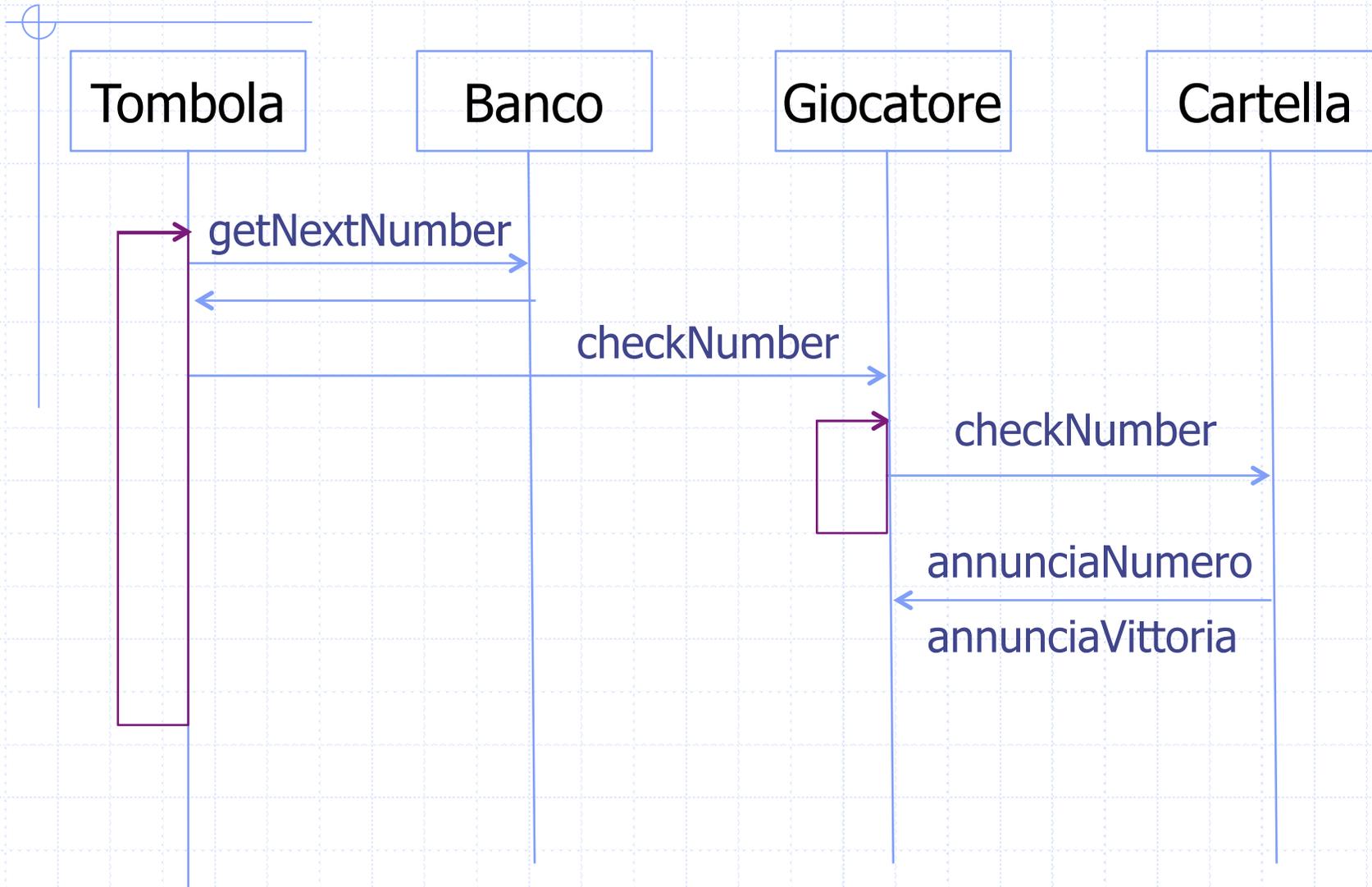
Esercizio:

Modificare il codice aggiungendo un numero arbitrario di giocatori, ciascuno con un numero arbitrario di cartelle

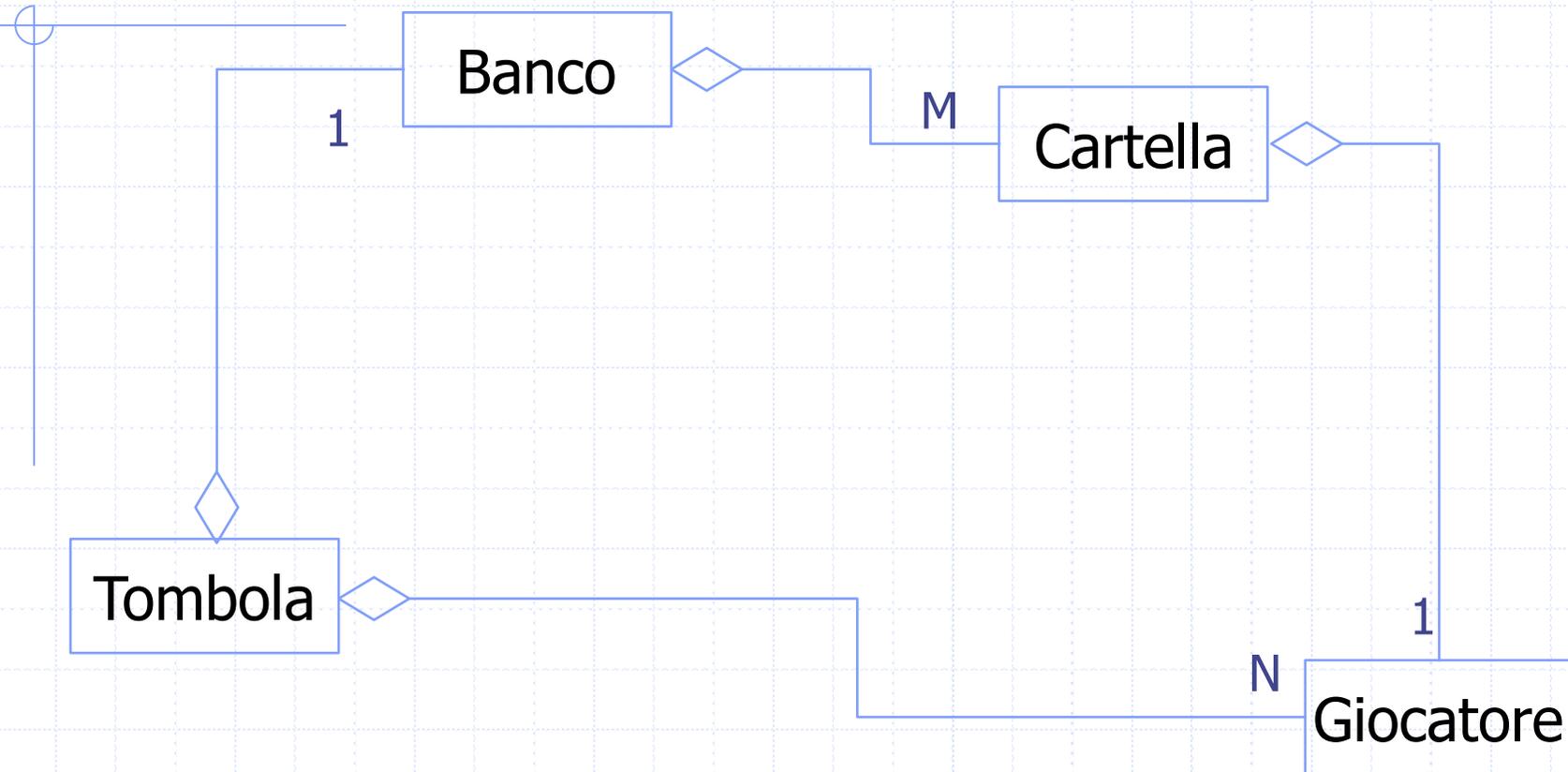
# Class diagram



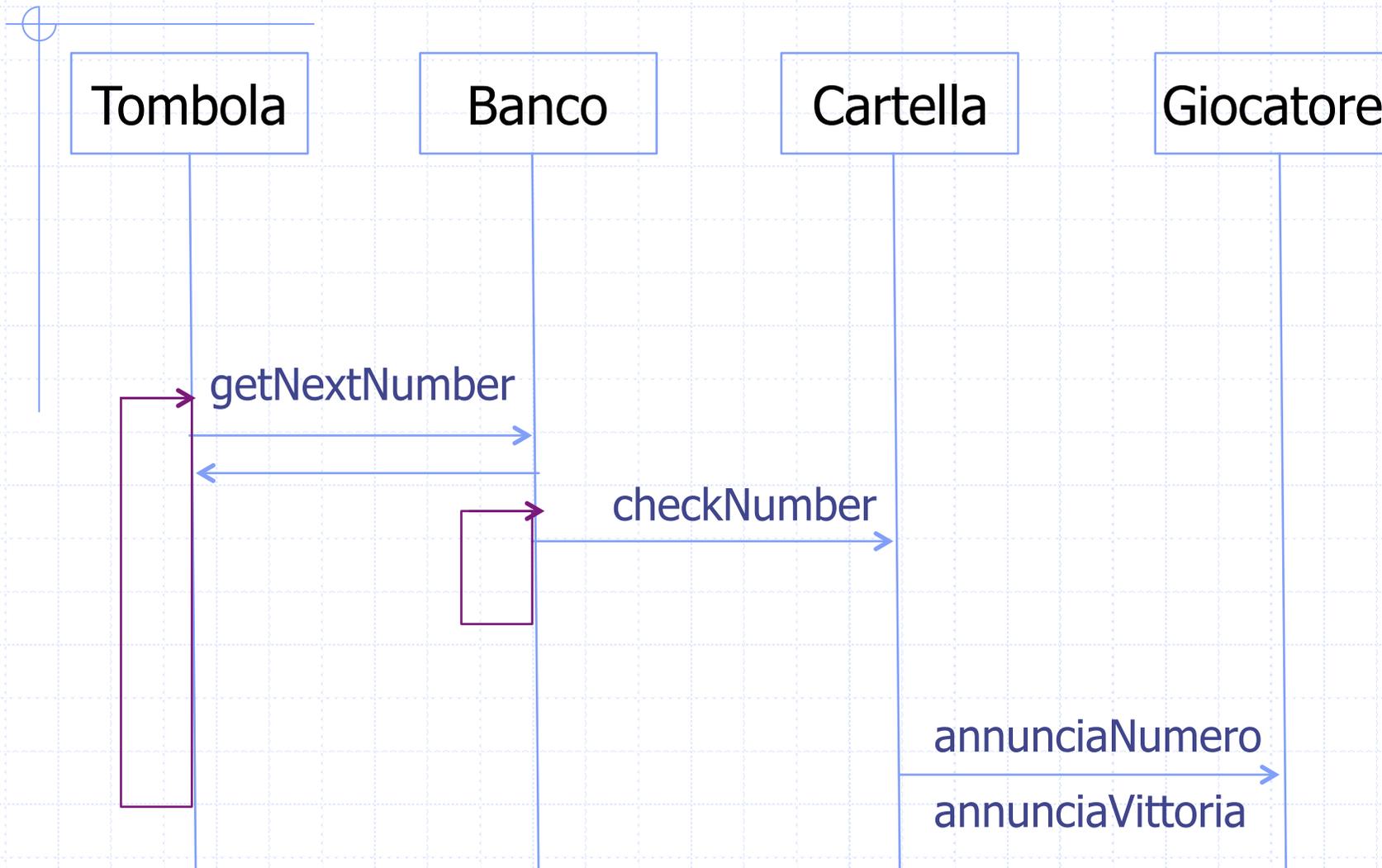
# Sequence diagram (swim lanes)



# Class diagram



# Sequence diagram (swim lanes)

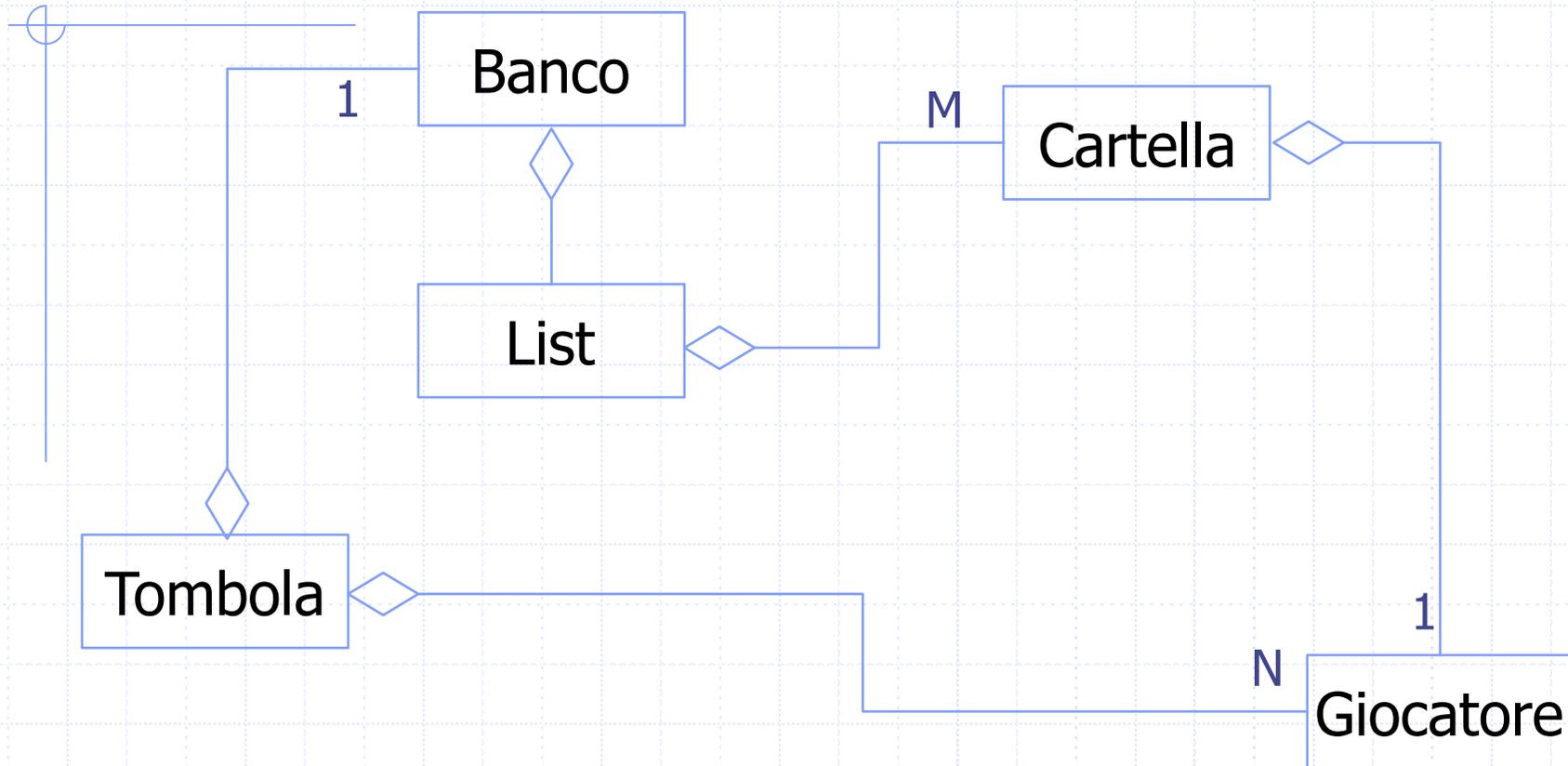


# Listener

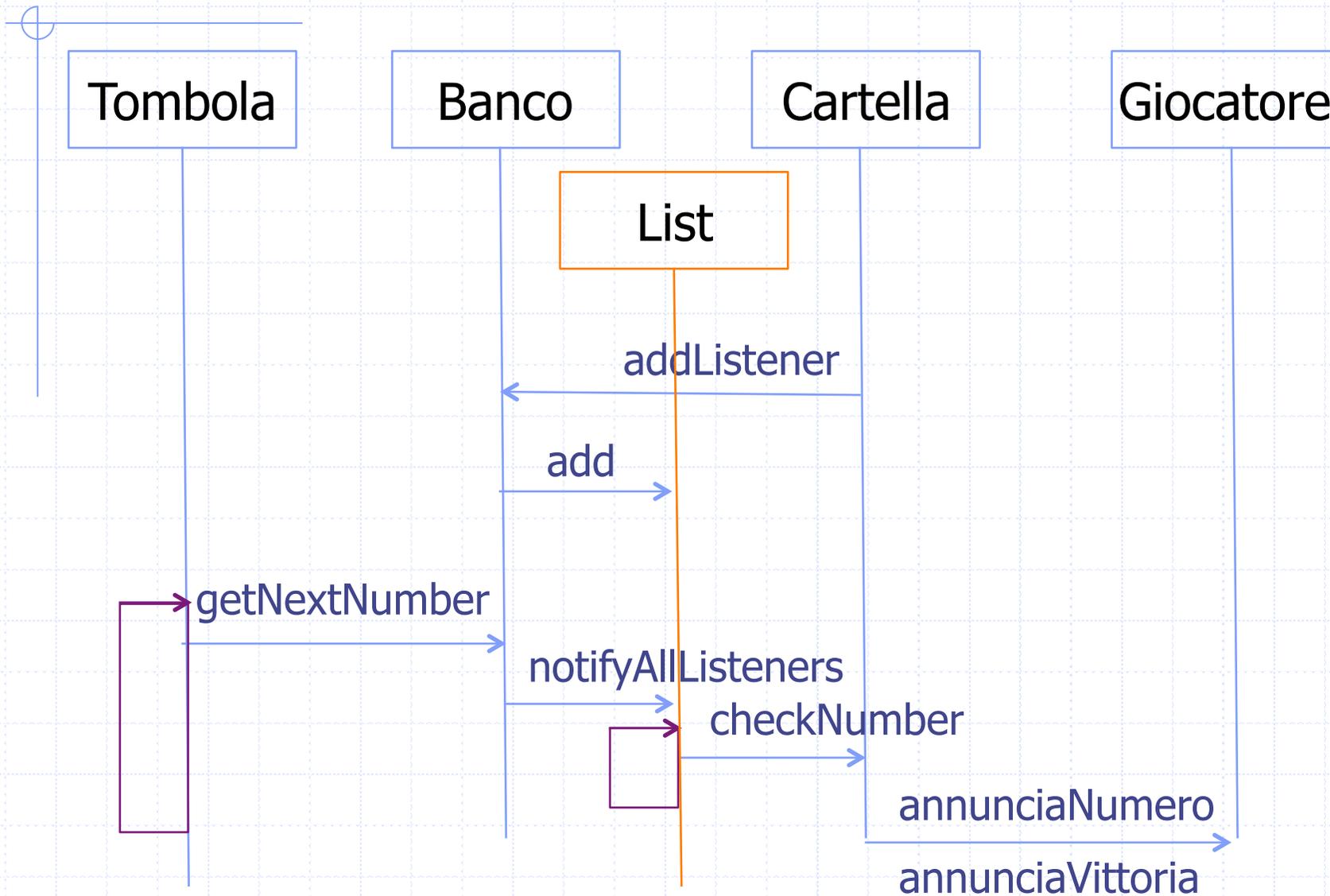
Introduciamo la nozione di "ascoltatore"

secondo il paradigma "publish & subscribe"

# Class diagram



# Activity diagram (swim lanes)



# Common



```
package tombola;  
import java.util.Random;  
public class Common {  
    static final int NCELLS=3;  
    static final int MAXNUM=10;  
    static final Random generatore =  
        new Random(System.currentTimeMillis());  
}
```

# Giocatore



*Eliminiamo qualche pezzo*

```
package tombola;  
public class Giocatore {  
    public String name;  
    // private Cartella cartella; non serve  
    Giocatore(String name){  
        this.name=name;  
        // cartella=new Cartella(this); non serve  
    }  
    void checkNumber(int x){  
    cartella.checkNumber(x);  
    }  
}
```

# Giocatore

```
void annunciaNumero(int num, int cartellaId){  
    System.out.println(name+" ha il numero  
"+num+" in cartella "+cartellaId);  
}
```

```
void annunciaVittoria(int cartellaId) {  
    System.out.println(name+" ha vinto con  
cartella "+cartellaId);
```

```
    cartella.printOriginale();
```

```
    System.exit(1);
```

```
}
```

```
}
```



*Quasi invariato*

# Cartella

```
package tombola;
```

```
import java.util.HashSet;  
import java.util.Iterator;
```

```
public class Cartella {
```

```
    private HashSet<Integer> numeri = new HashSet();  
    private HashSet<Integer> mancanti = new HashSet();  
    private Giocatore proprietario=null;  
    private int id=0;  
    static int nCartelle=0;
```

```
    Cartella(Giocatore g) {
```

```
        id=++nCartelle;
```

```
        proprietario=g;
```

```
        for (int i = 1; i <= Common.NCELLS; i++) {
```

```
            boolean creatoNuovoNumero = false;
```

```
            do {
```

```
                int x = Common.generatore.nextInt(Common.MAXNUM)+1;
```

```
                creatoNuovoNumero = numeri.add(x);
```

```
                if (creatoNuovoNumero) System.out.println("aggiunto "+ x);
```

```
            } while (!creatoNuovoNumero);
```

```
        }
```

```
        mancanti.addAll(numeri);
```

```
    }
```



*invariato*

# Cartella

```
public boolean checkNumber(int x) {  
    boolean result = mancanti.remove(x);  
    if(proprietario!=null) {  
        if (result) proprietario.annunciaNumero(x, id);  
        if (mancanti.isEmpty()) proprietario.annunciaVittoria(id);  
    }  
    return result;  
}
```

```
private void print(HashSet<Integer> list) {  
    Iterator<Integer> iter = list.iterator();  
    while (iter.hasNext()) {  
        System.out.print(iter.next()+" ");  
    }  
    System.out.println();  
}  
public void printOriginale() {print(numeri);}  
public void printCurrent() {print(mancanti);}  
}
```



*invariato*

# Banco

```
package tombola;
```

```
import java.util.LinkedList;  
import java.util.List;
```

```
public class Banco {  
    List<Integer> sacchetto;  
    List<Cartella> cartelle;  
  
    public Banco() {  
        cartelle= new LinkedList();  
        sacchetto= new LinkedList();  
        for (int i=1; i<=Common.MAXNUM;i++) {  
            sacchetto.add(i);  
        }  
    }  
}
```

# Banco – Lista di ascoltatori

```
void addListener(Cartella c){  
    cartelle.add(c);  
}
```

```
void removeListener(Cartella c){  
    cartelle.remove(c);  
}
```

```
private void notifyAllListeners(int x){  
    Iterator<Cartella> iter=cartelle.iterator();  
    while (iter.hasNext()) {  
        iter.next().checkNumber(x);  
    }  
}
```

# Banco – Lista di ascoltatori

```
void addListener(Cartella c){  
    cartelle.add(c);  
}  
void removeListener(Cartella c){  
    cartelle.remove(c);  
}  
private void notifyAllListeners(int x){  
    for (Cartella c:cartelle) {  
        c.checkNumber(x);  
    }  
}
```



Implementazione  
alternativa

# Banco

```
public int getNextNumber() {  
    if (sacchetto.size()==0) {  
        System.out.println("NUMERI FINITI!");  
        System.exit(1);  
    }  
    int index=Common.generatore.nextInt(sacchetto.size());  
    int num= sacchetto.get(index);  
    sacchetto.remove(index);  
    System.out.println("====> ESTRATTO: "+num );  
    notifyAllListeners(num);  
    return num;  
}  
}
```

# Tombola

```
package tombola;
```

```
public class Tombola {
```

```
    public Tombola() {
```

```
        Banco banco = new Banco();
```

```
        Giocatore g1 = new Giocatore("Pippo");
```

```
        banco.addListener(new Cartella(g1));
```

```
        banco.addListener(new Cartella(g1));
```

```
        banco.addListener(new Cartella(g1));
```

```
        Giocatore g2 = new Giocatore("Pluto");
```

```
        banco.addListener(new Cartella(g2));
```

```
    while (true) {
```

```
        int x = banco.getNextNumber();
```

```
        System.out.println("Il numero estratto é + x);
```

```
    }
```

```
}
```

```
public static void main(String[] args) {
```

```
    Tombola x=new Tombola();
```

```
}
```

```
}
```



*Eliminati i riferimenti  
Diretti alle cartelle*

run!



**Cartella 1 per giocatore Pippo : 5 3 6**

**Cartella 2 per giocatore Pippo : 8 7 10**

**Cartella 3 per giocatore Pippo : 1 2 4**

**Cartella 4 per giocatore Pluto : 1 5 4**

**====> ESTRATTO: 7**

**Pippo ha il numero 7 in cartella 2**

**Il numero estratto é 7**

**====> ESTRATTO: 5**

**Pippo ha il numero 5 in cartella 1**

**Pluto ha il numero 5 in cartella 3**

**Il numero estratto é 5**

**====> ESTRATTO: 9**

**Il numero estratto é 9**

**====> ESTRATTO: 6**

**Pippo ha il numero 6 in cartella 1**

**Il numero estratto é 6**

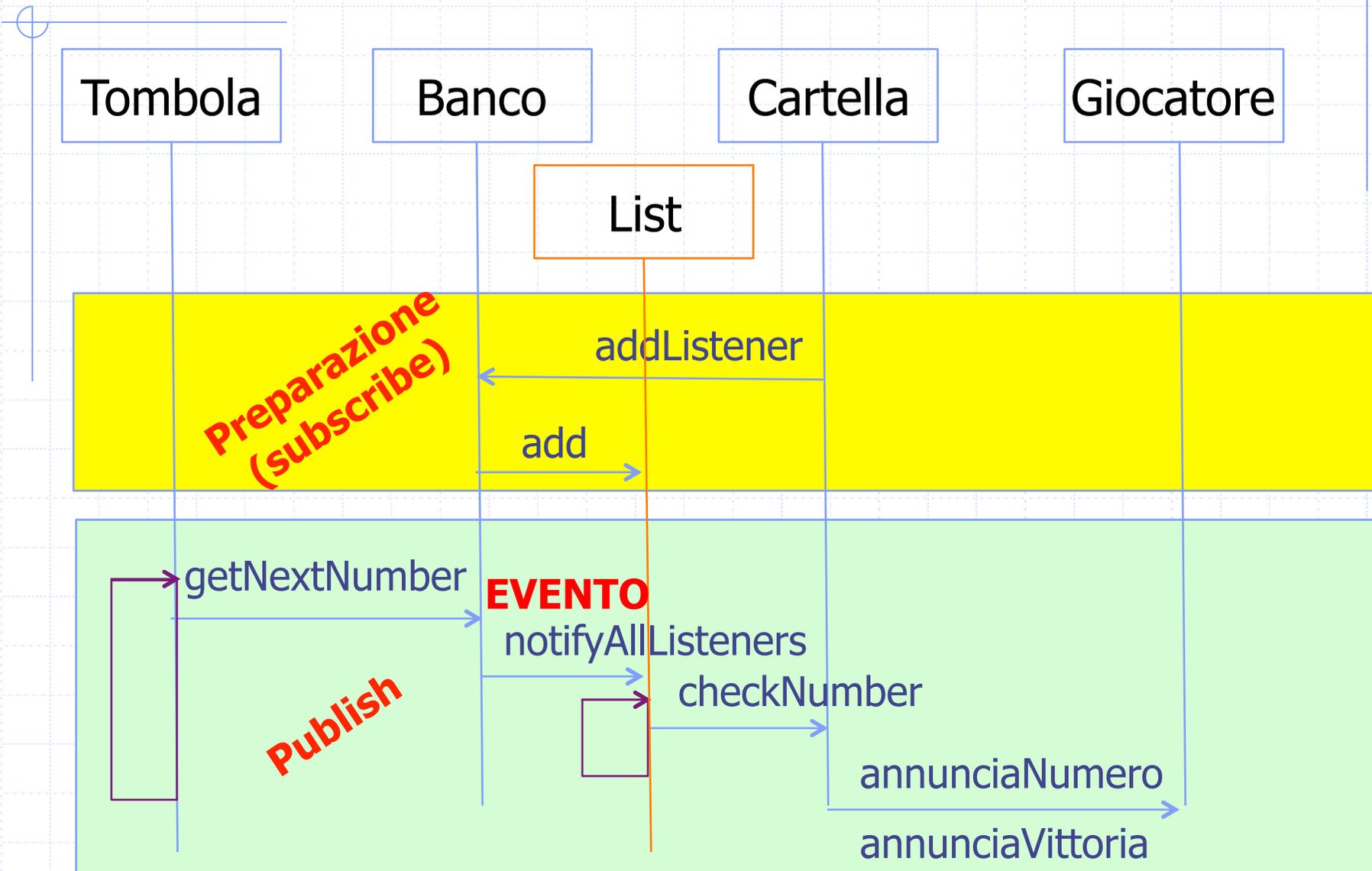
**====> ESTRATTO: 3**

**Pippo ha il numero 3 in cartella 1**

**Pippo ha vinto con cartella 1**

**3 5 6**

# Activity diagram (swim lanes)



# Events, Frameworks e e callbacks

**Eventi:** accadimenti "esterni":

- Un bottone è stato premuto
- Il mouse si è mosso
- Un pin di un microcontroller ha cambiato livello di voltaggio
- E' arrivata una richiesta http
- Il caricamento di un'immagine è terminato
- ...

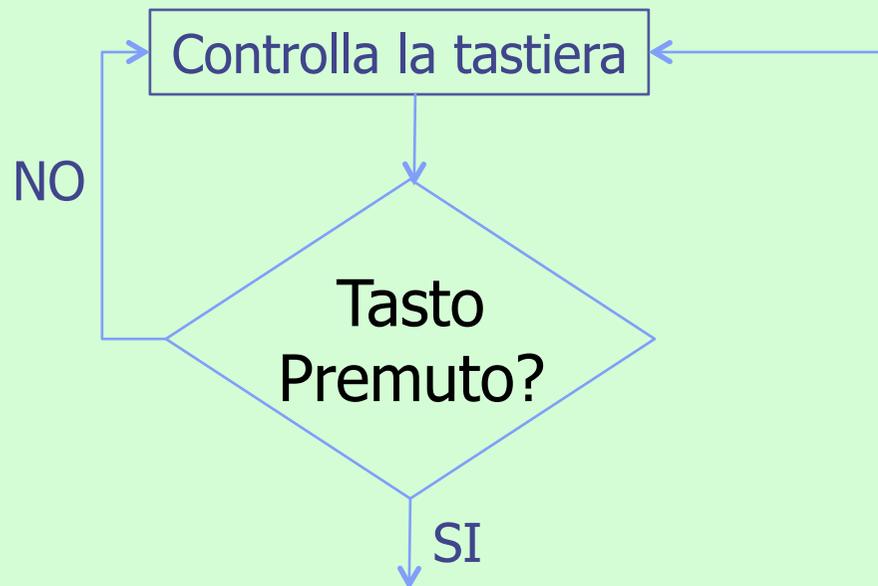
**Framework:** codice che gestisce in modo standardizzato un certo tipo di eventi, e che può essere customizzato scrivendo la (sola) parte non standard del codice

**Callback (event handler):** codice che customizza il comportamento di risposta a un tipo di evento.

# Architettura di un framework

## FRAMEWORK

abbonaUtente (KbEventHandler u):  
Aggiungi u alla lista degli abbonati.



Crea un KeyboardEvent,  
chiama "gestisci (KeyboardEvent)"  
su tutti gli abbonati

```
interface KbEventHandler {
    gestisci (KeyboardEvent k)
}
```

```
class MyProgram extends Framework
implements KbEventHandler {
    MyProgram () {
        startFramework();
        abbonaUtente(this);
    }
    void gestisci (KeyboardEvent k) {
        faiQualcosaCon(k);
    }
}
```

Nota: in JavaFx:

```
class Myprog extends Application {
    ...main(...) {
        launch(args);
    }
}
```

# JavaFX – gestione bottoni

Subscribe

Callback

```
btn.setOnAction(new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent event) {  
        System.out.println("Hello World!");  
    }  
});
```

## Definizioni:

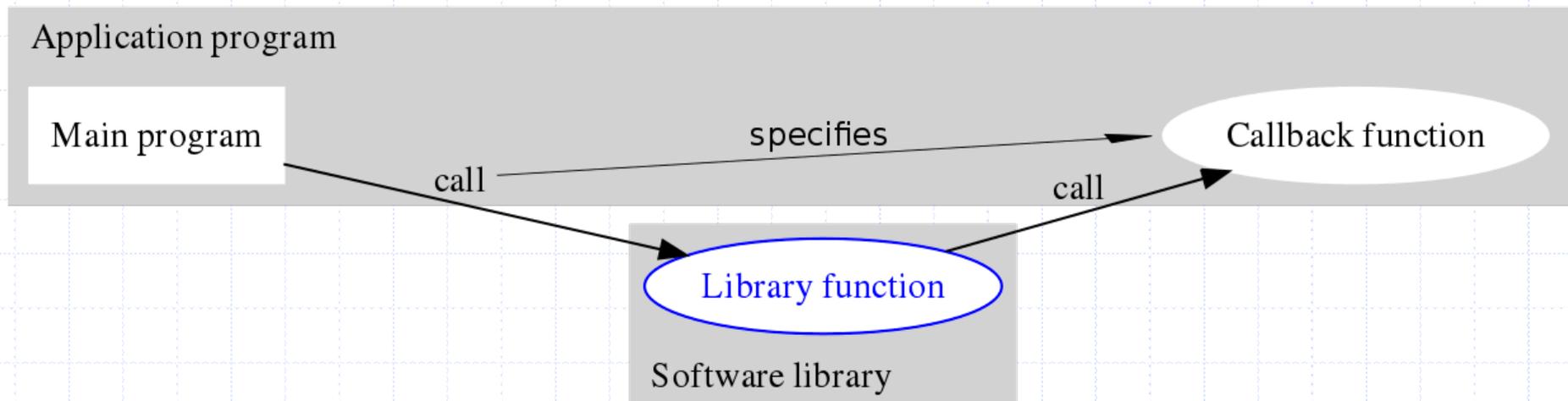
**Software framework** is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software.

**Event** : an action or occurrence recognized by software, often originating asynchronously from the external environment, that may be handled by the software.

## Definizioni:

**Callback:** any executable code that is passed as an argument to other code that is expected to call back (execute) the argument at a given time.

**Event handler:** a callback subroutine that handles inputs received in a program



## Definizioni:

**software framework** is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software.

