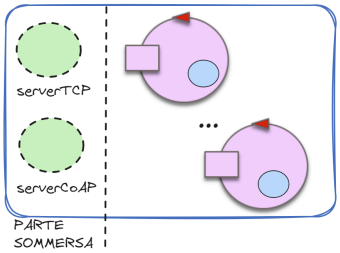
Naive Actors 2024

Progetto ProdCons

Le classi inserite nel package *unibo.naiveactors24* sono un esempio di una possibile implementazione del concetto di *Actor* limitato ad un ambiente locale.



Un attore che specilizza la classe *unibo.naiveactors24.ActorBasic24* e che appartiene a un *contesto* realizzato dalla classe *unibo.naiveactors24.ActorContext24*, può:

- inviare messaggi di tipo *IApplMessage* ad altri Attori locali al suo Contesto;
- elaborare i messaggi ricevuti, prelevandoli da una **coda di ingresso**;
- i messaggi possono provenire da attori locali e/o da componenti remoti, detti Alien.
- un *Alien* conosce solo il nome del compoente–Actor cui vuole inviare infromazione (strutturata come un *IApplMessage*) e l'indirizzo della sua 'porta' di ingresso.

unibo.naiveactors24.ActorBasic24	Classe astratta che realizza il concetto di actor in Java, come componentoi attivo dotato di una coda di messaggi di tipo *IApplMessage* che gestisce in modo FIFO invocando il metodo abstract **elabMsg(IApplMessage msg)**. La classe fornisce anche metodi per l'invio di messaggi (**forward, request, reply**) ad altri attori dello stesso Contesto.
unibo.naiveactors24.ActorContext24	Classe che realizza il concetto di Contesto in Java, attivando un server TCP che gestisce i messaggi usando un *unibo.naiveactors24.ContextMsgHandler*
Come inviare la risposta a un Alien?	Il Contesto permette a un componente renoto (Alien) di inviare messaggi agli actor del Contesto. Nel caso di una

	request, occorre che la risposta sia inviata sulla connessione TCP stabilita dal componente remoto.
unibo.naiveactors24.ContextMsgHandler	Classe che estende *ApplMsgHandler*. Gestisce i messaggi compatibili con *IApplMessage* in arrivo sulla connesisone TCP, inserendoli nella coda dei messaggi dell'attore destinatario locale (se esiste). Nel caso di **request** da componente Alien remoto, inserisce nel messaggio un rifermento alla connessione, per rendere possibile il corretto invio della risposta.

In sintesi:

- abbiamo definito un layer che realizza, in modo parziale, il concetto di *Actor*
- un Actor può inviare messaggi ad un altro Actor dello stesso contesto, ma non ad Actor di un altro Contesto (remoto)
- un Actor può ricevere ed elaborare messaggi da programmi 'Alieni'

ProdsConsTowardsActors24

- I componenti-attori sono specializzazioni della classe *unibo.naiveactors24.ActorBasic24*. La loro logica applicativa è nel metodo

  ```
  protected void elabMsg(IApplMessage msg)
  ```

  che viene invocato dalla classe-base quando c'è un (nuovo) messaggio in coda.

- Il **Contesto**, istanza di *unibo.naiveactors24.ActorContext24*, in cui vivono i componenti-attori è creato dal Main program *main.towardsactors24.MainOneNodeWithActors24*.

main.towardsactors24.ConsumerAsActors24	Un attore che opera nel Contesto di nome *ctxprodcons*: riceve ed elabora messaggi *dispatch* e *request*
main.towardsactors24.ProducerAsActors24	Un attore che opera nel Contesto di nome *ctxprodcons*: invia

	messaggi *dispatch* e *request* al Consumer
main.towardsactors24.MainOneNodeWithActors24	Main program che crea il Contesto *ctxprodcons* e gli attori *Producer* e *Consumer*.

Un Producer Alieno

Un *Producer* Alieno è rappresentato da un programma che non conosce cosa siano gli Actor, ma che è capace di inviare messaggi su una connessione con il server TCP inglobato nel *Contesto* del *Consumer*.

main.towardsactors24.ProduceAsActor24External	Usa *unibo.naiveactors24.ActorNaiveCaller* per stabilire la connessione con il *Consumer*.
main.towardsactors24.ProducerExternalPython	Non essendo disponibile in Python un supporto simile a *ActorNaiveCaller*, invia messaggi usando una socket.

Da fare: observable e observer

Ciò che non posso creare, non capisco: *RICHARD PHILLIPS FEYNMAN*.

Observer in ActorBasic24 : estendere la casse *unibo.naiveactors24.ActorBasic24* in modo da realizzare il metodo

```
protected void updateResource(String s) {...}
```

Invocato da un attore di nome **a**, *updateResource* invia a tutti gli attori **obs** che sono stati **registrati come osservatori** presso **a**, un dispatch della forma:

```
msg(update,dispatch,a,obs,msg,<N>)
```

Confronto con oop : descrivere la differenza del funzionamento tra quanto realizzato e il funzionamento del *Observer pattern* nella convenzionale programmazioni ad oggetti.

Observer logger : realizzare un actor di nome **obslogger** che, registrato presso un *Producer* e il *Consumer*, riceve i messaggi **update** emessi dal *Producer* e dal *Consumer* e li gestisce aggiornando un log-file di nome **obsloggerLog.txt** che memorizza le interazioni avvenute nel sistema.

Logger check : realizzare un programma che effettua il testing del sistema ProdCons analizzando il file **obsloggerLog.txt**, al termina della esecuzione del sistema .