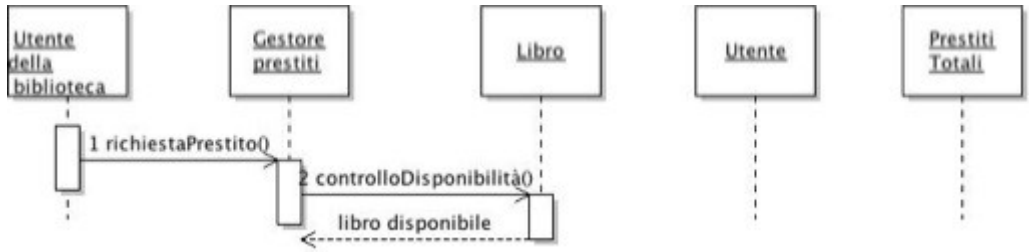
Ingegneria del Software a.a. 2013-14
Prova Scritta del 14 gennaio 2014

BOZZA DI SOLUZIONE

Esercizio 1

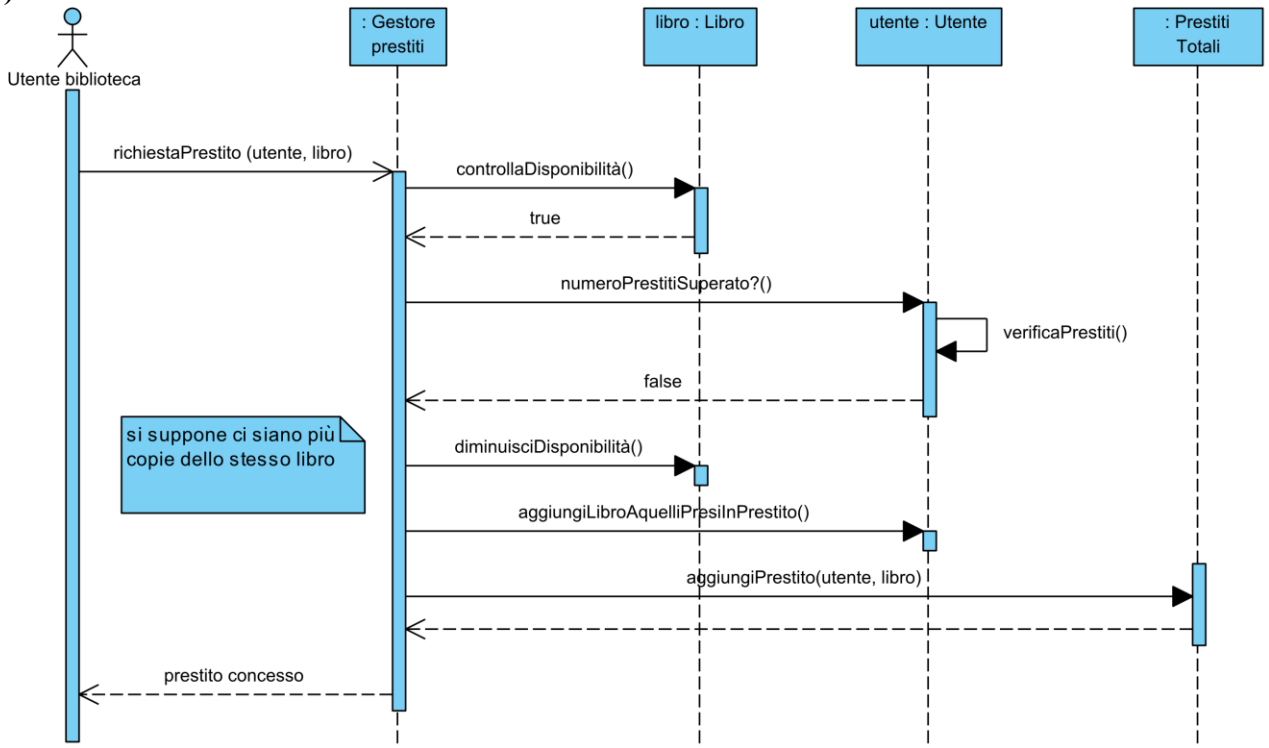
Supponiamo di dover progettare un sistema software per la gestione di prestiti di libri per una biblioteca. Lo scenario di **richiesta prestito** che si vuole modellare è il seguente: se il libro è disponibile il sistema effettua il controllo sul numero dei prestiti già erogati per l'utente. Se il numero massimo di prestiti è stato raggiunto allora non è possibile prendere in prestito il libro. In caso contrario il prestito viene concesso e il sistema aggiorna la lista dei libri in prestito. Il responsabile dell'erogazione del prestito è il *Gestore prestiti* che sostanzialmente rappresenta l'interfaccia tra l'utente e il resto del sistema. Sarà dunque suo compito verificare se il libro è disponibile o meno, concedere il prestito, e fare gli opportuni aggiornamenti. La figura seguente rappresenta un diagramma di sequenza "parziale" per lo scenario sopra descritto.



a) Il diagramma mostrato sopra contiene un imprecisione a livello di notazione UML. Quale?
b) Completare il sequence diagram considerando che tutte le condizioni (es. il libro è disponibile) siano soddisfatte
c) Disegnare il diagramma delle classi relativo allo scenario sopra esposto facendo attenzione che quest'ultimo risulti allineato (ovvero sia coerente) al diagramma di sequenza proposto al punto precedente
d) Utilizzando il costrutto "Frame" disegnare un unico diagramma di sequenza che tenga conto anche degli scenari in cui: 1) l'utente ha superato il numero massimo di prestiti e 2) che il libro non è disponibile

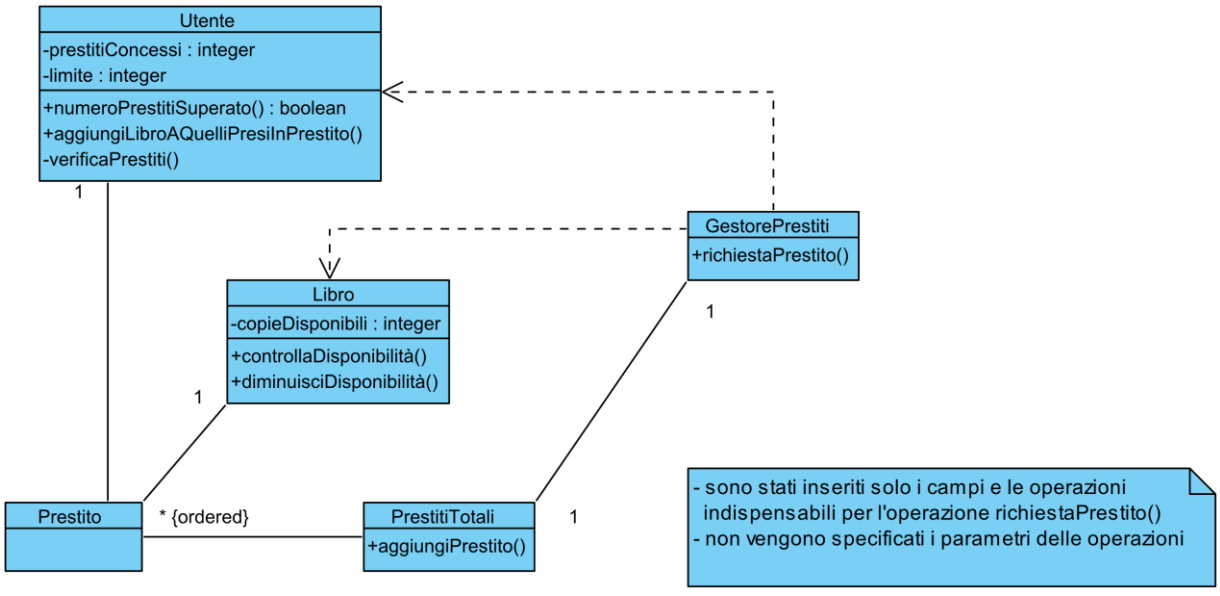
a) In effetti esistono diverse "imprecisioni" e non una sola. La più evidente però è il fatto che la freccia del messaggio controlloDisponibile() e quella della risposta "libro disponibile" è quella di una chiamata asincrona quando invece dovrebbe essere sincrona. Altre imprecisioni (o migliorie da apportare) per essere più aderenti alle specifiche UML 2.0 sono il fatto che i nomi dei participant non devono essere sottolineati e dovrebbero essere preceduti da ":". Inoltre sarebbe meglio usare l'omino stilizzato per identificare l'utente della biblioteca.

b)

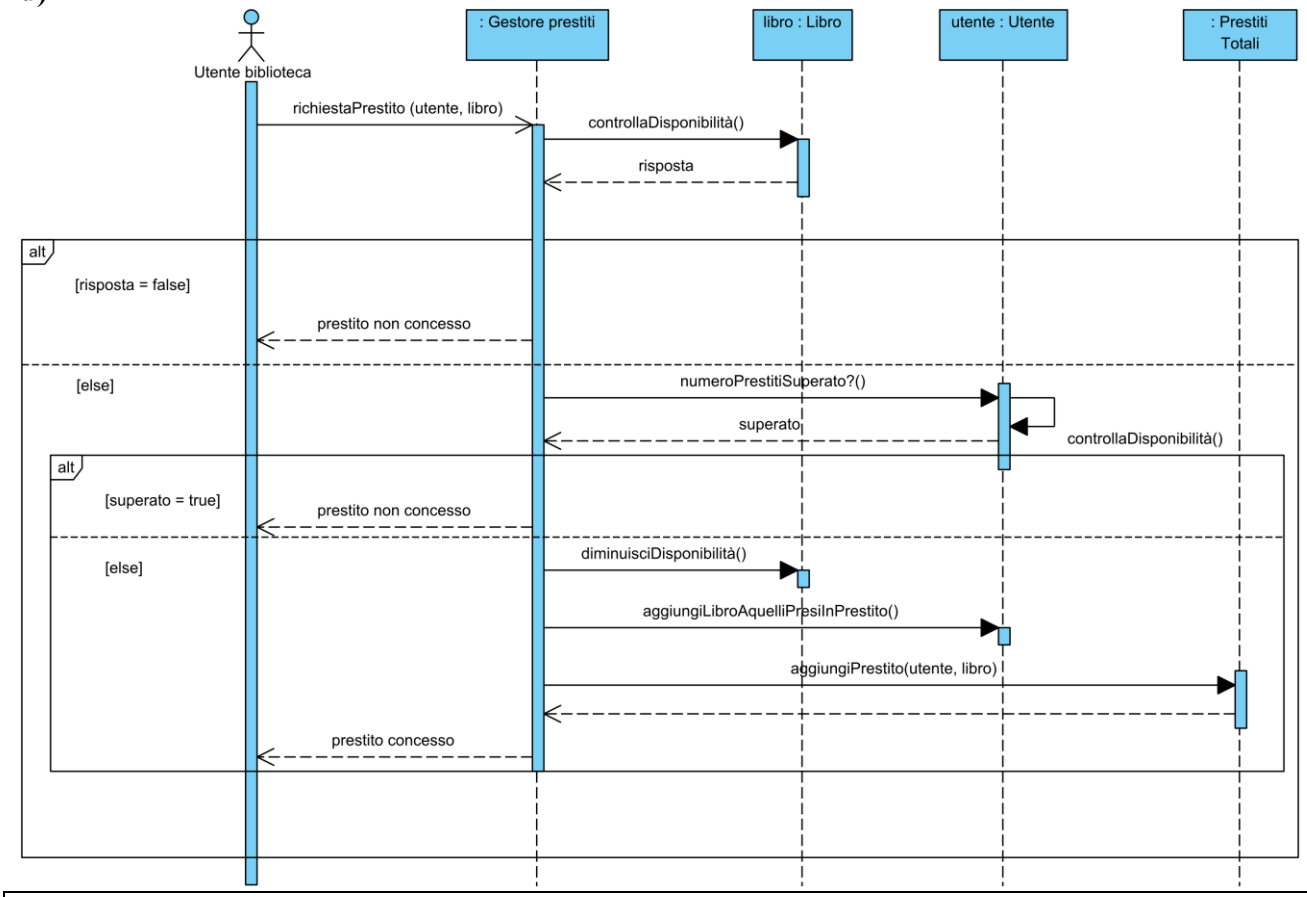


Il *gestore clienti*, non appena ricevuto il valore true dell'operazione *controllaDisponibilità()*, effettua una richiesta all'oggetto utente per verificare se ha superato il numero massimo di prestiti concessi contemporaneamente. Come vediamo questa richiesta, non svolge direttamente nessuna operazione, ma delega il calcolo all'operazione *verificaPrestiti()*. Questo tipo di operazione è diversa rispetto alle altre precedentemente mostrate in quanto non c'è interazione con altri oggetti, ma tutto avviene all'interno dell'oggetto. Ciò vuol dire che operazioni di questo tipo possono essere calcoli matematici, esecuzione di algoritmi ed in generale quella serie di operazioni necessarie per raggiungere un risultato. Successivamente le due operazioni *diminuisciDisponibilità()* e *aggiungiLibroAQuelliPresiInPrestito()* aggiornano i due contatori contenuti nei corrispettivi oggetti. Infine si ha l'operazione richiesta di aggiornamento dei prestiti della biblioteca.

c)



d)



Esercizio 2

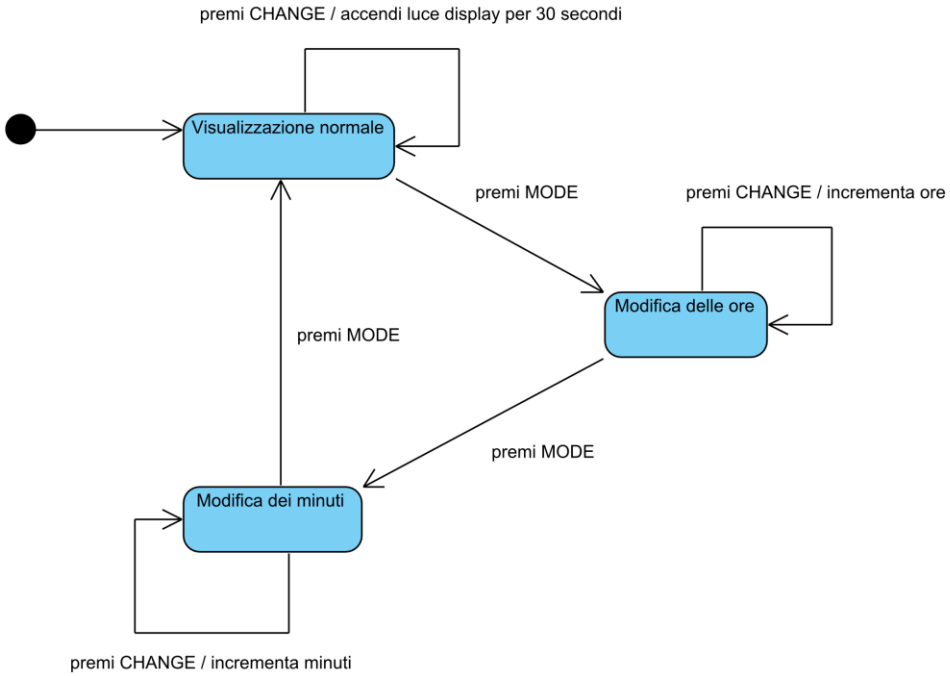
Si pensi ad un orologio che possiede due pulsanti: MODE e CHANGE. Il primo pulsante serve per settare ciclicamente il modo di operazione: da "visualizzazione normale" a "modifica delle ore" a "modifica dei minuti" e poi ancora "visualizzazione normale" ed avanti così. Il secondo pulsante, invece, serve per accendere la luce del display per 30 secondi se è in modalità di visualizzazione normale, oppure per incrementare di una unità le ore o i minuti, se è in modalità di modifica di ore o di minuti.

a) Rappresentare il comportamento dell'orologio con un diagramma UML che ritenete più opportuno

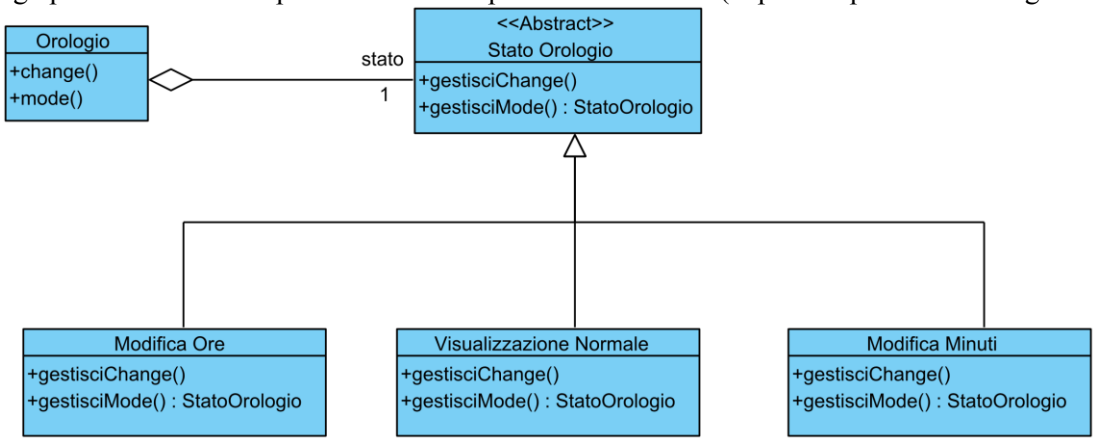
b) Instanziando un design pattern visto a lezione disegnare un class diagram che sia un utile punto di partenza per l'implementazione di un applicazione che simuli il funzionamento dell'orologio descritto sopra

c) Implementare utilizzando lo pseudocodice (oppure se preferite direttamente il linguaggio Java) le classi più importanti dell'applicazione

a) Il diagramma più opportuno è il diagramma di stato (state machine diagram)



b) Il design pattern è uno State pattern. Ecco una possibile soluzione (la più semplice non la migliore!)



c) Una possibile implementazione Java che rispetta il class diagram dato sopra. Non viene gestito il conteggio di ore e minuti. Si poteva fare inserendo due campi interi (ora e minuti) nella classe Orologio e gestendo l'incremento nei metodi gestisciChange() nelle classi ModificaOre e ModificaMinuti. L'operazione stampaStato() è stata aggiunta solo per effettuare il testing dell'applicazione.

```
public class Orologio {
        StatoOrologio stato;
        public Orologio()  {
                stato = new VisualizzazioneNormale();
        }
        public String stampaStato() {
                System.out.println(stato.print());
                return stato.print();
        }
        public void gestisciChange() {
                stato.gestisciChange();
        }
        public void gestisciMode() {
                stato = stato.gestisciMode();
        }
}
```

```
public abstract class StatoOrologio {
        public abstract void gestisciChange();
        public abstract StatoOrologio gestisciMode();
        public abstract String print();
}

public class VisualizzazioneNormale extends StatoOrologio {
        public void gestisciChange() {
                System.out.println("accendi luce");
        }
        public StatoOrologio gestisciMode() {
                return new ModificaOre();
        }
        public String print() {
                return "VisualizzazioneNormale";
        }
}

public class ModificaOre extends StatoOrologio {
        public void gestisciChange() {
                System.out.println("aumenta ora");
        }
        public StatoOrologio gestisciMode() {
                return new ModificaMinuti();
        }
        public String print() {
                return "ModificaOre";
        }
}

public class ModificaMinuti extends StatoOrologio {
        public void gestisciChange() {
                System.out.println("aumenta minuti");
        }
        public StatoOrologio gestisciMode() {
                return new VisualizzazioneNormale();
        }
        public String print() {
                return "ModificaMinuti";
        }
}
```

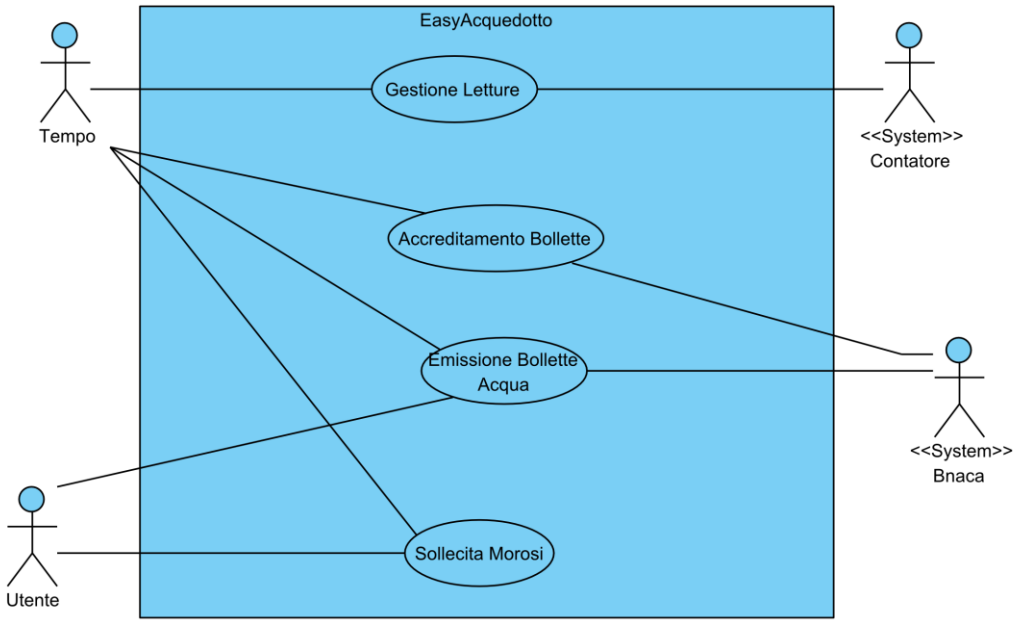
Esercizio 3

Si consideri il sistema *EasyAcquedotto* di gestione delle bollette dell'acquedotto avente quattro funzionalità ben distinte:

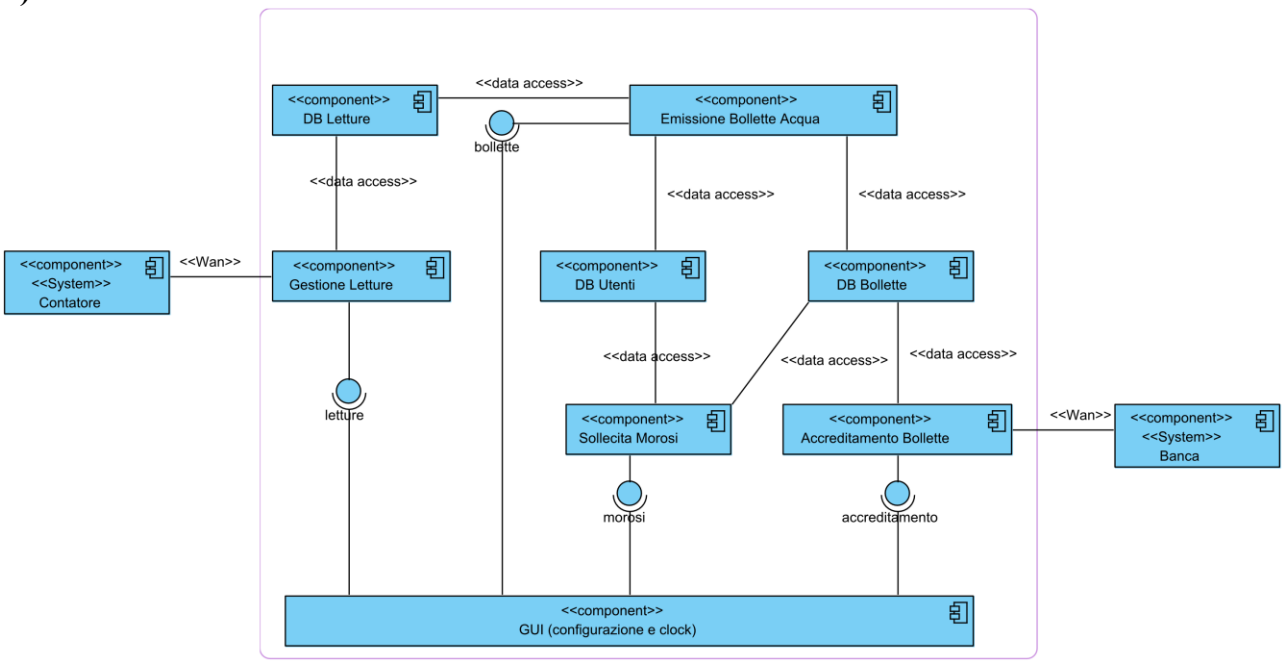
- *GestioneLetture*: periodicamente, il sistema preleva dai contatori elettronici le letture del consumo e le registra in una base di dati, DBLetture.
- *EmissioneBolletteAcqua*: periodicamente, il sistema stampa, per l'invio all'Utente, usando dati prelevati dal DBUtente, le bollette relative all'ultima lettura, e le registra nel DBBollette; nel caso di bollette domiciliate le invia pure elettronicamente alla Banca, per il pagamento.
- *AccreditamentoBollette*: periodicamente, il sistema riceve elettronicamente dalla Banca i pagamenti delle bollette, e li registra nel DBBollette.
- *SollecitiMorosi*: periodicamente, estraendo le informazioni dal DBBollette, stampa i solleciti per i clienti morosi, usando i dati del DBUtenti.

a) Fornire un diagramma dei casi d'uso per EasyAcquedotto. Per rappresentare il fatto che le funzionalità sono attivate "periodicamente" utilizzare un attore fittizio chiamato "Tempo". Notare che i DB sono parte del sistema
b) Rappresentare con un diagramma delle componenti l'architettura del sistema EasyAcquedotto
c) Descrivere la logica della componente *SollecitiMorosi* mediante un activity diagram UML

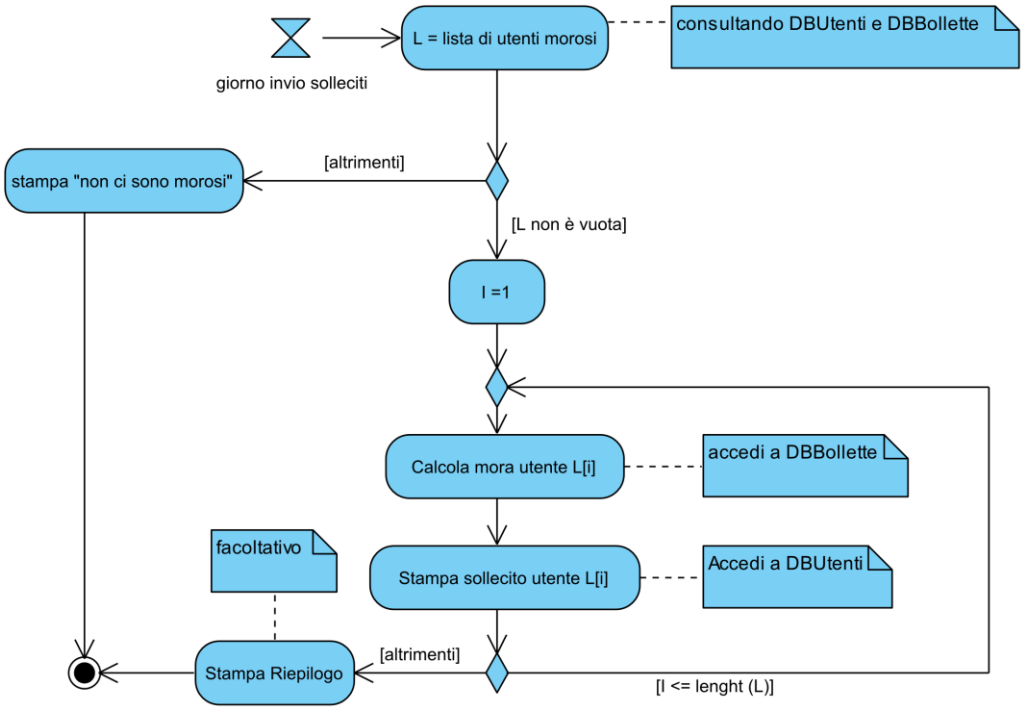
a)


EasyAcquedotto
Gestione Letture
Accreditamento Bollette
Emissione Bollette Acqua
Sollecita Morosi
Tempo
<<System>>
Contatore
<<System>>
Bnaca
Utente

b)


<<component>>
DB Letture
<<data access>>
<<component>>
Emissione Bollette Acqua
bollette
<<data access>>
<<data access>>
<<data access>>
<<component>>
<<System>>
Contatore
<<Wan>>
<<component>>
Gestione Letture
<<component>>
DB Utenti
<<component>>
DB Bollette
<<data access>>
<<data access>>
<<data access>>
letture
<<component>>
Sollecita Morosi
<<component>>
Accreditamento Bollette
<<Wan>>
<<component>>
<<System>>
Banca
morosi
accreditamento
<<component>>
GUI (configurazione e clock)

c)


giorno invio solleciti
L = lista di utenti morosi
consultando DBUtenti e DBBollette
[altrimenti]
stampa "non ci sono morosi"
[L non è vuota]
I =1
Calcola mora utente L[i]
accedi a DBBollette
facoltativo
Stampa sollecito utente L[i]
Accedi a DBUtenti
[altrimenti]
Stampa Riepilogo
[I <= lenght (L)]