

**Corso di Progettazione del Software**

Esame del 14 Giugno 2018

Tempo a disposizione: 3 ore

**Requisiti.** L'applicazione da progettare riguarda una parte di un videogioco di fantascienza. Un gioco è costituito da diversi giocatori. I giocatori, che hanno un nome (una stringa) sono divisi in 3 categorie: i piloti, i droidi e i passeggeri. Dei piloti interessano le ore di volo (un intero), dei droidi il modello (una stringa) e dei passeggeri il peso trasportato (un reale). Ogni pilota è abilitato a pilotare un insieme non vuoto di astronavi. Delle astronavi interessa la descrizione (una stringa). Ai piloti piace volare con alcuni droidi (almeno uno) e tra questi uno in particolare che funge da compagno di volo. Ogni droide è il compagno di volo di esattamente un pilota. Altri dettagli non interessano.

Siamo interessati al comportamento dei piloti. Un pilota è inizialmente alla base. Se riceve il comando di volare con payload una astronave e un passeggero, se il pilota è abilitato a pilotare l'astronave allora chiede al suo droide compagno di salire a bordo dell'astronave, mettendosi in attesa del droide. Quando questo gli comunica droide-salito allora chiede al passeggero di salire a bordo dell'astronave, mettendosi in attesa del passeggero. Quando il passeggero gli comunica passeggero-salito si mette in volo. Cosa fa quando in volo non interessa, eccetto che quando riceve il comando di rientro torna alla base. Il comportamento degli altri tipi di giocatore non interessa.

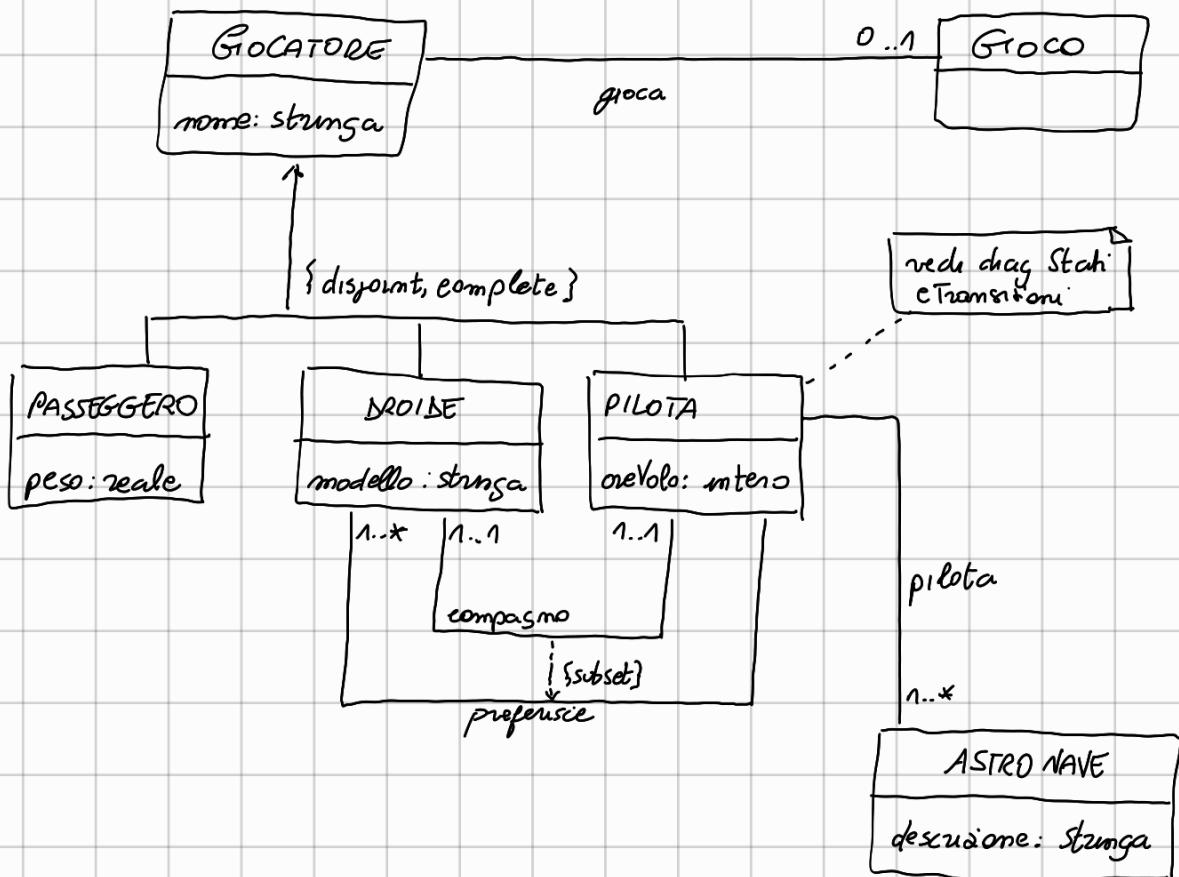
Siamo interessati alla seguente attività principale. L'attività prende in input un (gioco G) verifica che il numero di piloti, droidi e passeggeri di G sia congruo e che tutti i droidi compagni dei piloti di G siano anche loro giocatori di G (i dettagli non interessano). Se la verifica non va a buon fine, l'attività termina segnalando in output un errore. Altrimenti concorrentemente esegue le seguenti due sottoattività: (i) gioca, e (ii) analisi. La sottoattività di gioco (i) avvia il gioco attivando tutti i giocatori di G mandando opportuni eventi (i dettagli non interessano). Poi si mette in attesa del comando di fine-gioco da parte dell'utente che interrompe il gioco. La sottoattività di analisi (ii) calcola quanti piloti, droidi e passeggeri sono presenti nel gioco G mandando un report con questi dati in output. Una volta che tali sottoattività sono state completate, l'attività principale manda un segnale di output di saluto e termina.

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo del diagramma delle classi (inclusi vincoli non esprimibili in UML), diagramma stati e transizioni per la classe Pilota, diagramma delle attività, specifica del diagramma stati e transizioni, e specifica dell'attività principale e delle sottoattività NON atomiche (indicando in modo esplicito quali attività atomiche sono di I/O e quali sono Task), motivando, qualora ce ne fosse bisogno, le scelte di progetto.

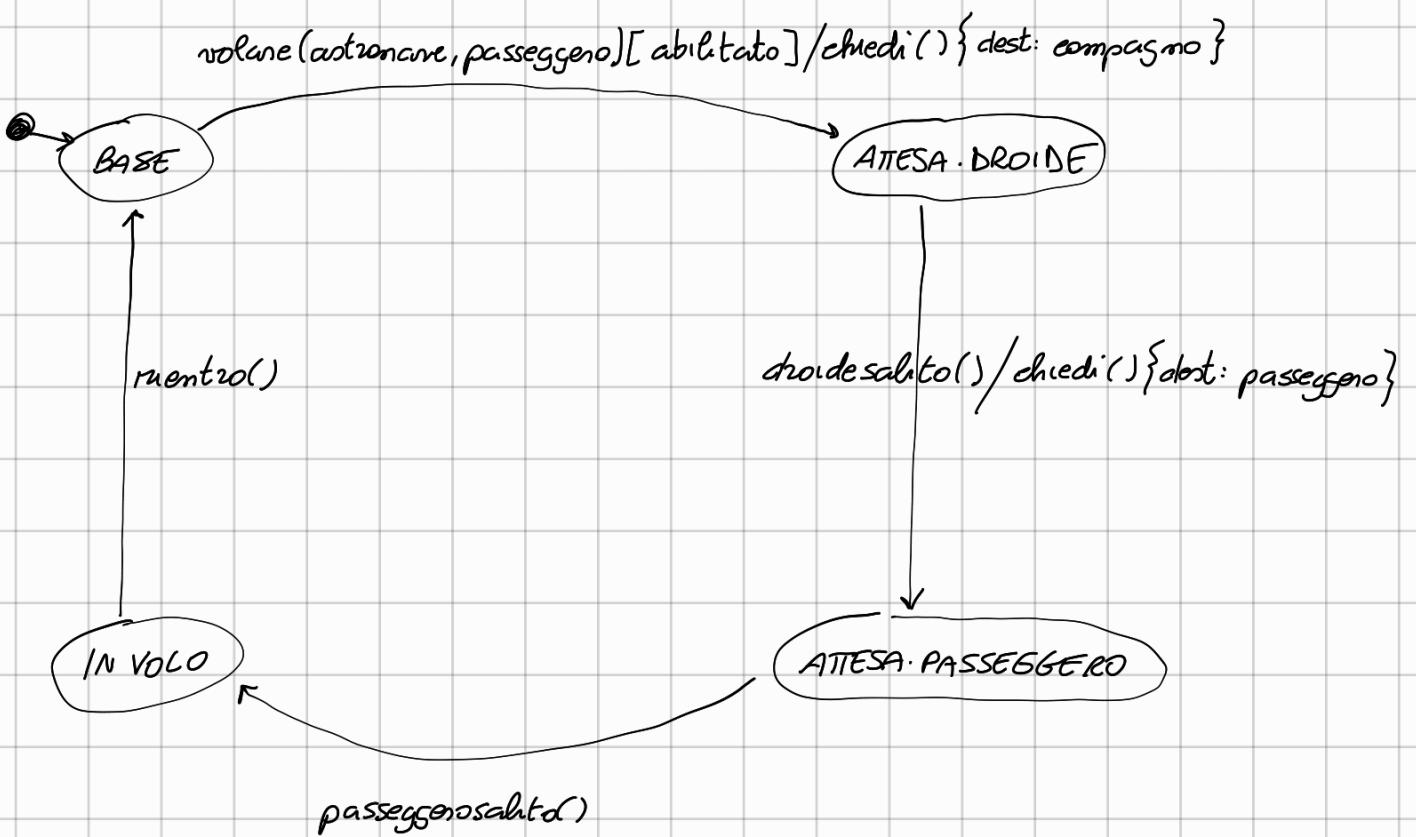
**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. È obbligatorio definire solo le responsabilità sulle associazioni del diagramma delle classi.

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. È obbligatorio realizzare in JAVA solo i seguenti aspetti dello schema concettuale:

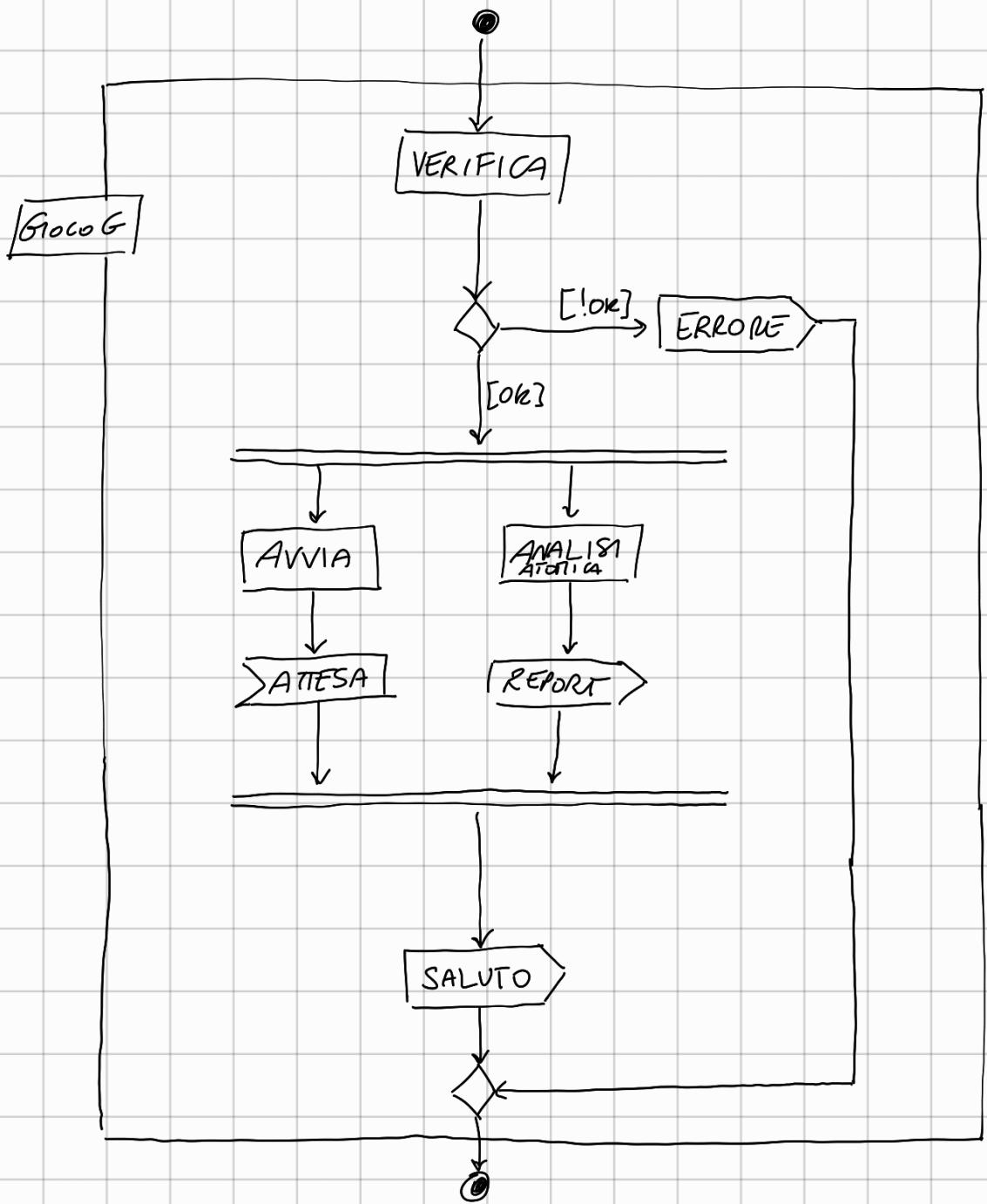
- La classe Pilota con classe PilotaFired, le eventuali superclassi, e le classi JAVA per rappresentare le associazioni di cui la classe Pilota ha responsabilità.
- L'attività principale e le sue eventuali sottoattività NON atomiche.



ASSOCIAZIONE	CLASSE	ha RESP?
giocatore	Giocatore	Si <sup>1</sup>
	Alieno	Si <sup>0</sup>
pilota	Pilota	Si <sup>1</sup>
	Astronave	No
preferenze	Pilota	Si <sup>1</sup>
	Droide	No
compagno	Droide	Si <sup>1</sup>
	Pilota	Si <sup>1,0</sup>



Var Aux      passeggero: Passeggero



**Corso di Progettazione del Software**Esame del **27 Luglio 2018**

Tempo a disposizione: 3 ore

**Requisiti.** L'applicazione da progettare riguarda un sistema di diffusori audio e video per domotica. Una istallazione domotica ha un indirizzo (una stringa) e un insieme di reti. Le reti hanno un nome e appartengono a una sola istallazione. Inoltre le reti sono costituite da uno o più diffusori; tra questi esattamente uno svolge il ruolo di master. Ciascun diffusore appartiene esclusivamente ad una rete. I diffusori hanno un nome e sono partizionati in diffusori audio e diffusori video. Dei diffusori audio interessano i watt nominali (un intero) e dei diffusori video interessano i pollici (un reale).

Siamo interessati al comportamento dei diffusori. Un diffusore è inizialmente in standby. Se, nello stato di standby, riceve il comando di diffondere con payload un (contenuto) di tipo audio o video, verifica se egli stesso lo può diffondere, cioè se esso stesso è del tipo (audio o video secondo quanto richiesto nel payload) giusto, e se lo è lo diffonde mettendosi in diffusione. Altrimenti sceglie uno dei diffusori della sua rete del tipo giusto (si assume di avere una ausiliaria funzione di scelta già definita) e gli passa il comando di diffondere con il rispettivo (payload) mettendosi nello stato di attesa. Se arriva il comando di stop, se è nello stato di diffusione smette di diffondere il contenuto e torna in standby; se invece è nello stato di attesa, passa il comando stop al diffusore scelto e torna in standby.

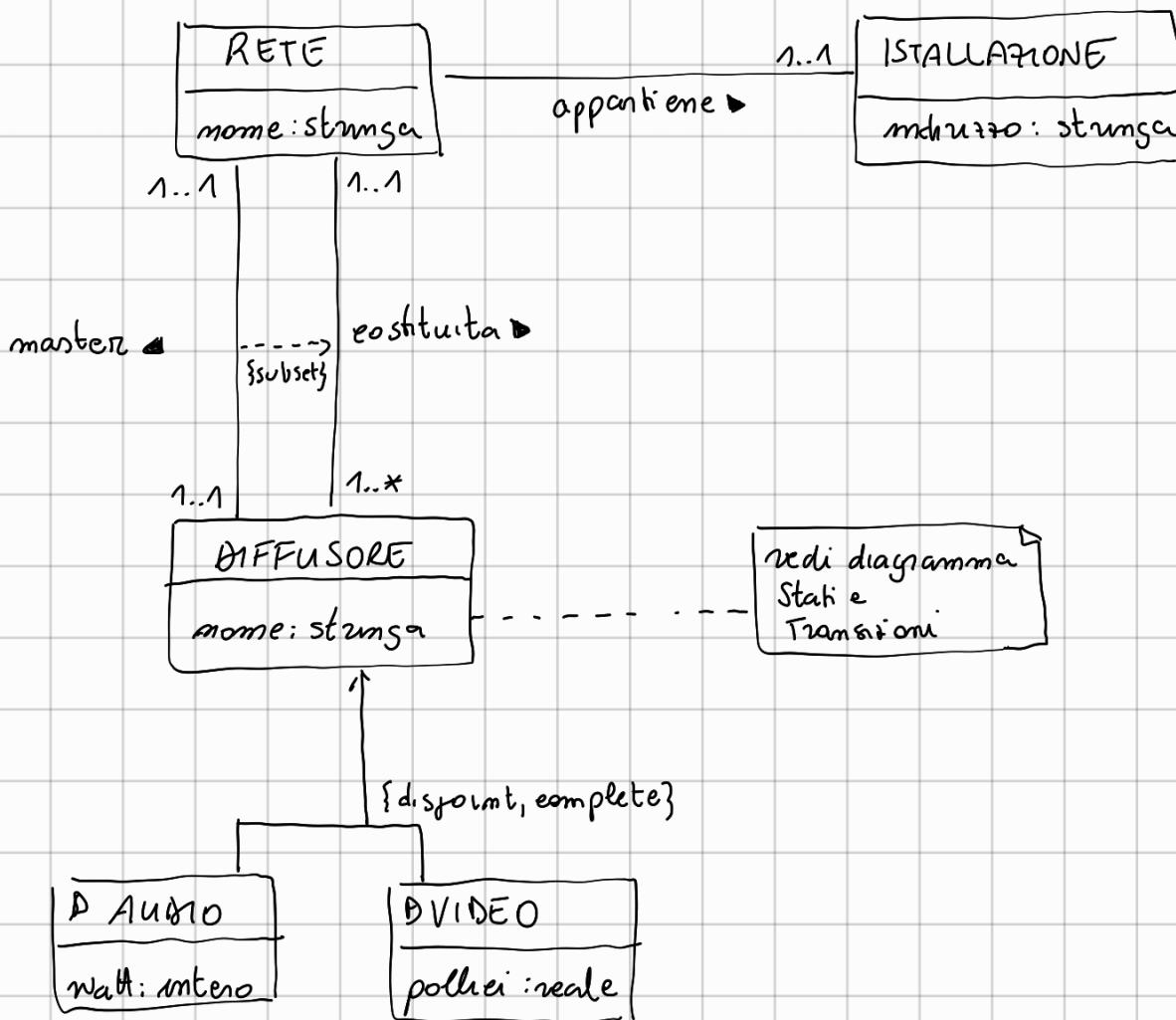
Siamo interessati alla seguente attività principale. L'attività prende in input una (istallazione  $I$ ) e verifica che ogni rete abbia un congruo numero di diffusori audio e video (i dettagli non interessano). Se la verifica non va a buon fine, l'attività termina segnalando in output un errore. Altrimenti concorrentemente esegue le seguenti due sottoattività: (i) funzionamento e (ii) analisi. La sottoattività di funzionamento (i) avvia l'istallazione portando tutti i diffusori di tutte le reti dell'istallazione  $I$  nello stato di standby. Poi si mette in attesa del segnale di input di fine funzionamento che interrompe il funzionamento stesso. La sottoattività di analisi (ii) calcola il numero di diffusori audio e video di ciascuna rete dell'istallazione  $I$  mandando un report con questi dati in output. Una volta che tali sottoattività sono state completate, l'attività principale manda un segnale di output con il numero medio di diffusori audio e video per rete dell'istallazione e termina.

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo del diagramma delle classi (inclusi vincoli non esprimibili in UML), diagramma stati e transizioni per la classe *Diffusore*, diagramma delle attività, specifica del diagramma stati e transizioni, e specifica dell'attività principale e delle sottoattività NON atomiche (cioè NON va riportata la specifica del Task e dei segnali di I/O), motivando, qualora ce ne fosse bisogno, le scelte di progetto.

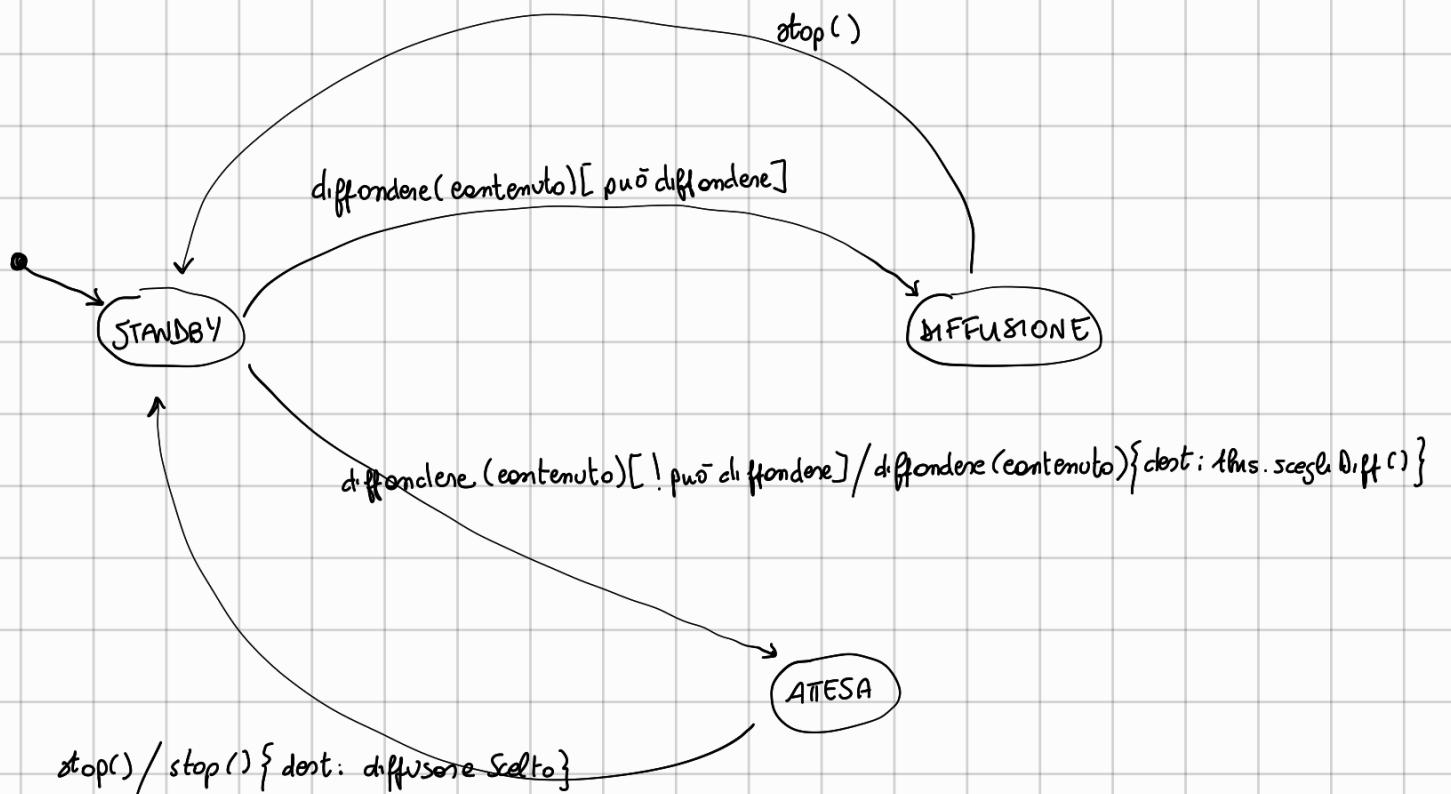
**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. È obbligatorio definire solo le responsabilità sulle associazioni del diagramma delle classi.

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. È obbligatorio realizzare in JAVA solo i seguenti aspetti dello schema concettuale:

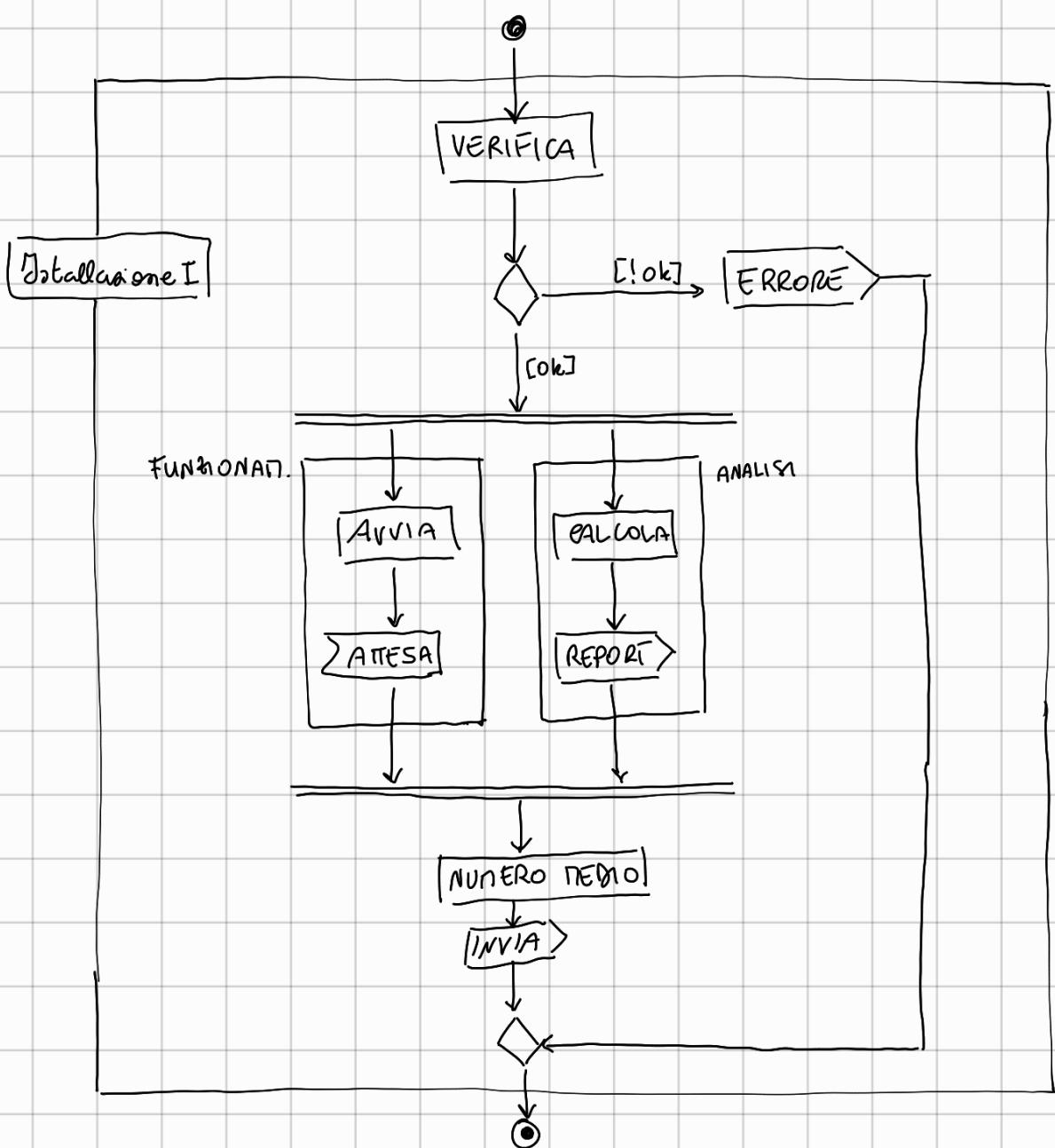
- La classe *Diffusore* con classe *DiffusoreFired*, le eventuali sottoclassi, e le classi JAVA per rappresentare le *associazioni* di cui la classe *Diffusore* ha responsabilità. Qualora sia necessario realizzare più classi *TipoLink* e *Manager* è obbligatorio riportarne solo una di entrambe.
- L'*attività principale* e le sue eventuali sottoattività NON atomiche.



ASSOCIAZIONE	CLASSI	ha RESP?	TIPO UNL	REALISAZIONE JAVA
appartenente	Rete	Sì $\sqcap$	stringa	String
costituita	Rete	Sì $\sqcap, \sqcup$	intero	int
master	Diffusore	Sì $\sqcap$	reale	double
	Diffusore	Sì $\sqcap$	insieme	HashSet
	Rete	Sì $\sqcap$	booleano	boolean



Van Aux    diffusioneScelto: Diffusione



**Corso di Progettazione del Software**

Esame del 11 Settembre 2018

Tempo a disposizione: 3 ore

**Requisiti.** L'applicazione da progettare riguarda la gestione di carovane per attraversare il deserto. Una carovana ha un nome (una stringa) ed è costituita da uno o più mezzi (di trasporto persone). Ogni mezzo e di un determinato tipo (una stringa) e trasporta uno o più persone. Ogni persona ha un nome (una stringa) ed è trasportato da esattamente un mezzo. I mezzi sono suddivisi in mezzi a guida autonoma e mezzi a guida non autonoma. Dei primi interessa l'anno dell'ultimo collaudo (in intero). Dei secondi interessa chi tra le proprie persone trasportate è il pilota, con il numero di volte che egli ha guidato il mezzo stesso. R

Siamo interessati al comportamento dei mezzi. Un mezzo è inizialmente alla a riposo. Se a riposo riceve l'evento partenza si mette in marcia (con le proprie persone incluso il pilota). Se in marcia riceve il evento di carica, dalla carovana o da un'altro mezzo, con payload un insieme di utenti li aggiunge alle proprie persone trasportate, rilanciando l'evento caricate al mittente dell'evento carica, rimanendo in marcia. Se in marcia riceve l'evento guasto manda in broadcasting la richiesta di accogliere le proprie persone e si mette in attesa. Se quando in attesa riceve l'ok da un'altro mezzo, manda ad esso l'evento carica con payload l'intero insieme delle proprie persone escluso il pilota mettendosi in trasbordo. Quando in in trasbordo riceve l'evento caricate del mezzo scelto si mette a riposo. Infine se in marcia riceve l'evento stop si mette a riposo.

Siamo interessati alla seguente attività principale. L'attività prende in input una carovana C ed un numero positivo n e verifica che il numero di utenti in ogni mezzo della carovana C sia inferiore o uguale a n. Se la verifica non va a buon fine, l'attività termina segnalando in output un errore. Altrimenti concorrentemente esegue le seguenti due sottoattività: (i) esecuzione, e (ii) analisi. La sottoattività di esecuzione (i) avvia l'esecuzione di tutti i mezzi mandando opportuni eventi (i dettagli non interessano) e si mette in attesa del segnale di input di fine-esecuzione che interrompe l'esecuzione stessa. La sottoattività di analisi (ii) calcola la percentuale di mezzi autonomi sul numero totale di mezzi della carovana C. Una volta che tali sottoattività sono state completate, l'attività principale manda un segnale di output con il risultato dell'attività di analisi e termina.

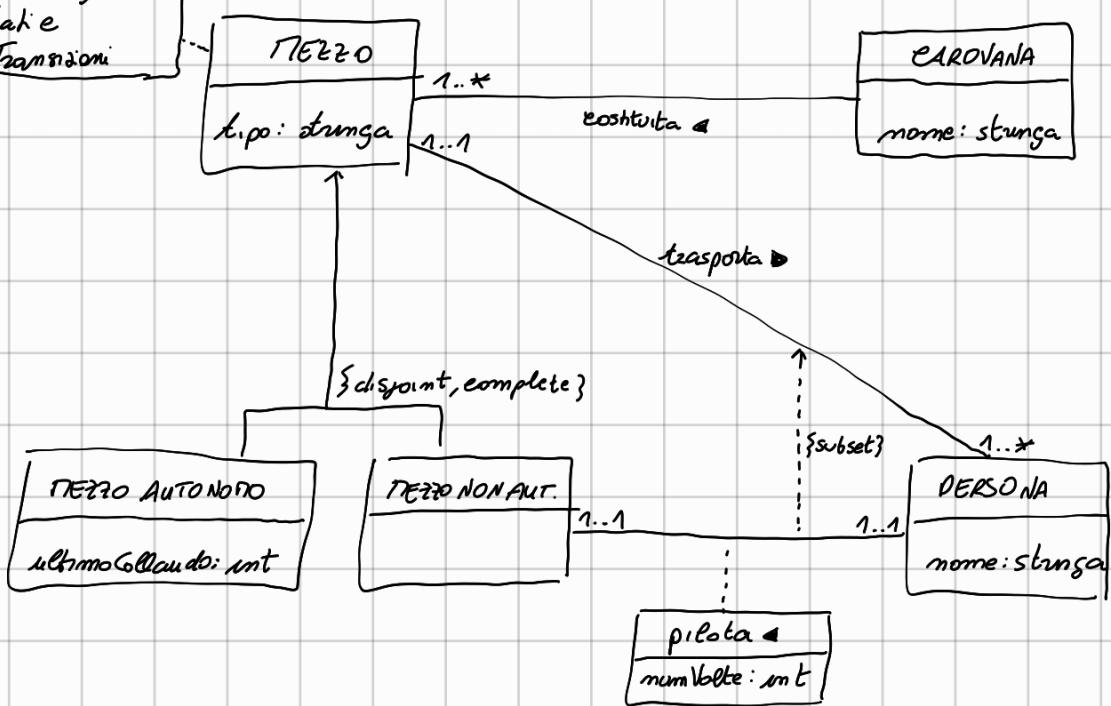
**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo del diagramma delle classi (inclusi vincoli non esprimibili in UML), diagramma stati e transizioni per la classe Mezzo, diagramma delle attività, specifica del diagramma stati e transizioni, e specifica dell'attività principale e delle sottoattività NON atomiche (indicando in modo esplicito quali attività atomiche sono di I/O e quali sono Task), motivando, qualora ce ne fosse bisogno, le scelte di progetto.

**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. È obbligatorio definire solo le responsabilità sulle associazioni del diagramma delle classi.

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. È obbligatorio realizzare in JAVA solo i seguenti aspetti dello schema concettuale:

- La classe Mezzo con classe MezzoFired, le eventuali sottoclassi, e le classi JAVA per rappresentare le associazioni di cui la classe Mezzo e le sue sottoclassi hanno responsabilità.
- L'attività principale e le sue eventuali sottoattività NON atomiche.

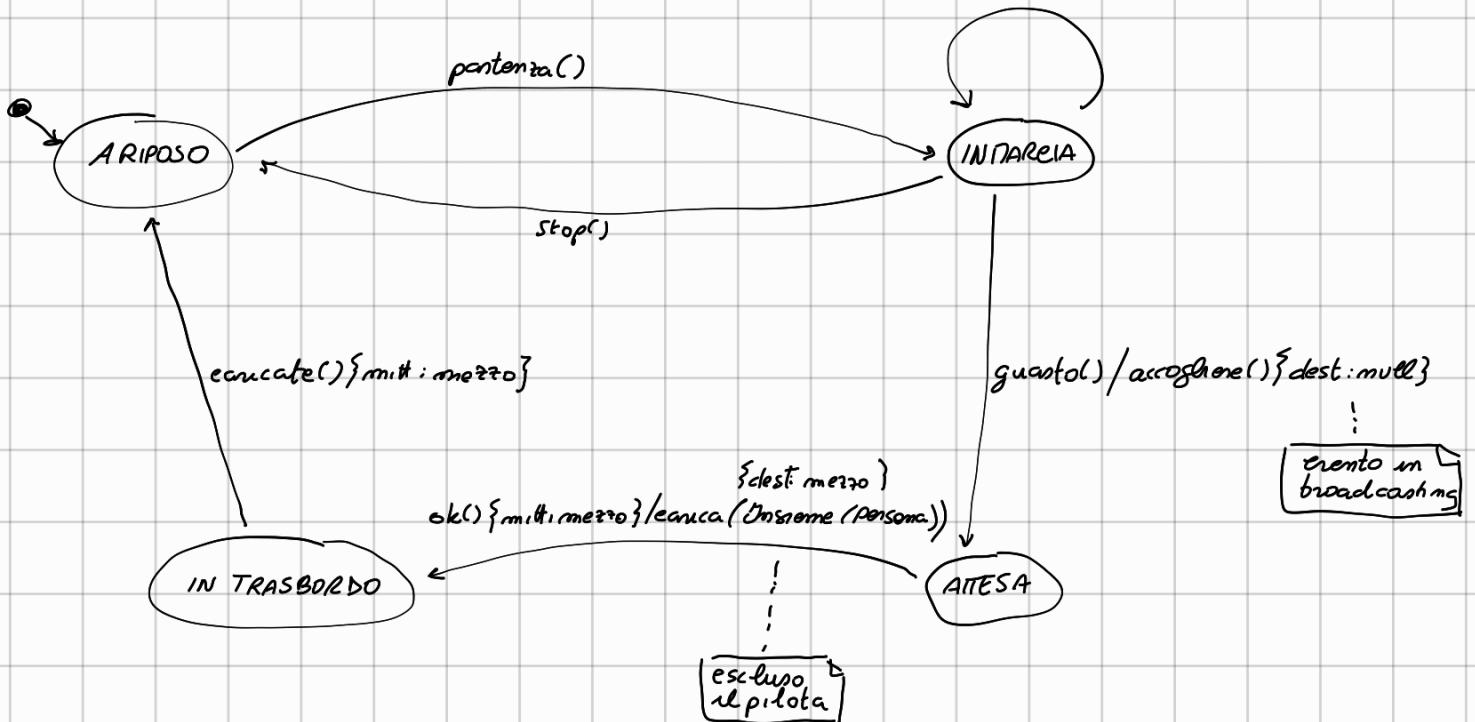
vedi diagramma  
Stati e  
Transizioni

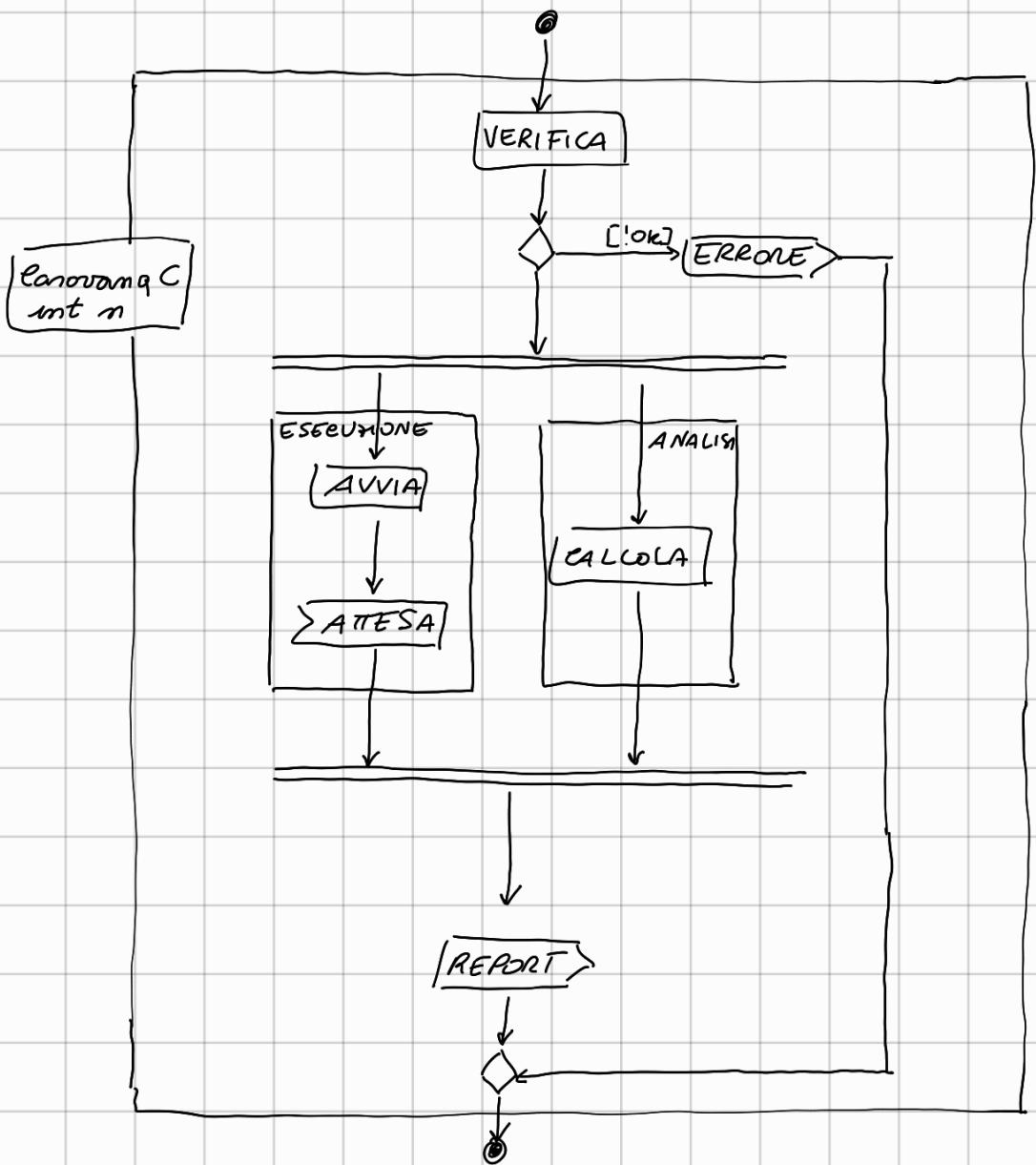


ASSOCIAZIONE	CLASSE	HA RESP?
costituta	Carovana	Sì $n,0$
	Nezzo	No
trasporta	Nezzo	Sì $n,0$
	Persona	Sì $n$
pilota	Persona	Sì $n$
	NezzoNonAut.	Sì $n,r$

TIPO UDL	REALIZZ. in JAVA
stringa	String
intesa	int
Insieme	HashSet
booleano	boolean

ecuca(insieme(Persone))/ecuca( ) {dest: ecuca.getIt, it: ()}





SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
Esame del **21 gennaio 2019**  
*Tempo a disposizione: 3 ore*

**Requisiti.** L'applicazione da progettare riguarda una parte di un videogioco di fantascienza. Un gioco è costituito da diversi giocatori. Ciascun giocatore appartiene ad un solo gioco e ha un nome (una stringa). I giocatori sono divisi in 3 categorie: i droidi, i robot, gli umani. Dei droidi interessa il modello (una stringa), dei robot l'anno di produzione (un intero), e degli umani una descrizione testuale (una stringa). Un droide ed un robot formano un equipaggio di cui interessa un codice identificativo. Ogni droide fa parte di diversi equipaggi. Invece ogni robot fa parte esattamente di un equipaggio. Un equipaggio pilota esattamente una astronave ed una astronave ha un solo equipaggio. Delle astronavi interessa una descrizione testuale (una stringa). Gli umani possono essere capitani di una astronave. Ogni astronave può avere un numero arbitrario di capitani.

Siamo interessati al comportamento dei droidi. Un droide è inizialmente alla base. Se riceve il comando di volare con payload un umano ed una astronave del cui equipaggio lui è il droide, chiede all'umano di salire sull'astronave come capitano, ponendosi in attesa del capitano. Quando l'umano gli comunica capitano-salito si mette in volo. Cosa fa quando in volo non interessa, eccetto che quando riceve il comando di rientro torna alla base. Il comportamento degli altri tipi di giocatore non interessa.

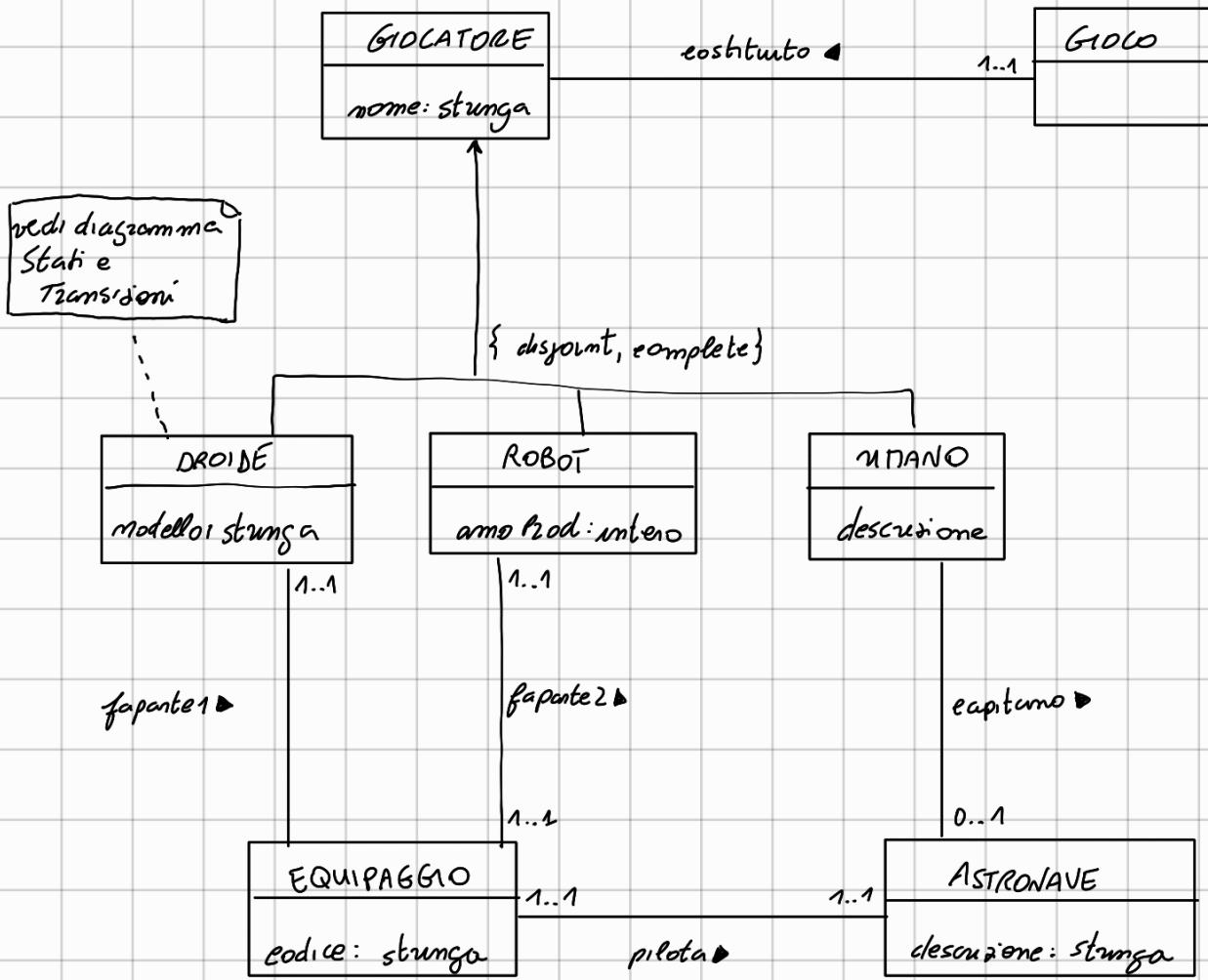
Siamo interessati alla seguente attività principale. L'attività prende in input un gioco  $G$  e verifica che il numero di droidi, robot, e umani di  $G$  sia congruo (i dettagli non interessano). Se la verifica non va a buon fine, l'attività termina segnalando in output un errore. Altrimenti concorrentemente esegue le seguenti due sottoattività: (i) gioco e (ii) analisi. La sottoattività di gioco (i) avvia il gioco attivando tutti i giocatori di  $G$  mandando opportuni eventi (i dettagli non interessano). Poi si mette in attesa del segnale di fine-gioco che interrompe il gioco stesso. La sottoattività di analisi (ii) calcola un report sul gioco  $G$  (i dettagli non interessano). Una volta che tali sottoattività sono state completate, l'attività principale manda un segnale di output con il report calcolato nella sottoattività di analisi e termina.

**XDomanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi vincoli non esprimibili in UML); diagramma stati e transizioni per la classe Droide; diagramma delle attività; specifica del diagramma stati e transizioni; segnatura dell'attività principale, sottoattività non atomiche, atomiche e segnali di input/output. Si noti che NON è richiesta la specifica delle attività. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

**XDomanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi.

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

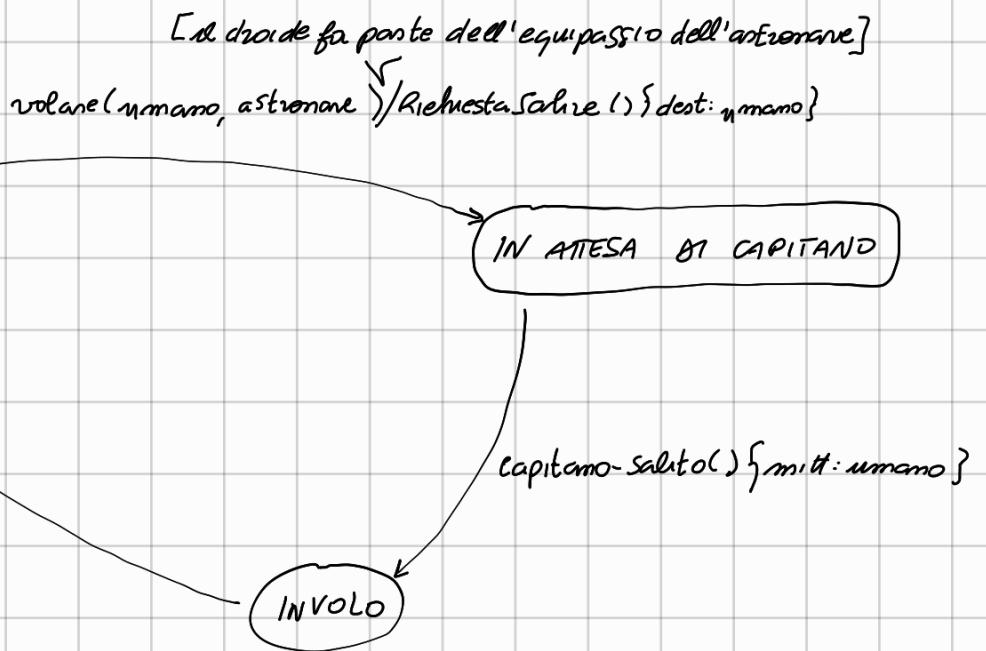
- La classe Droide con classe DroideFired, le eventuali superclassi, e le classi JAVA per rappresentare le associazioni di cui la classe Droide e le sue superclassi hanno responsabilità.
- L'attività principale e le sue eventuali sottoattività NON atomiche.



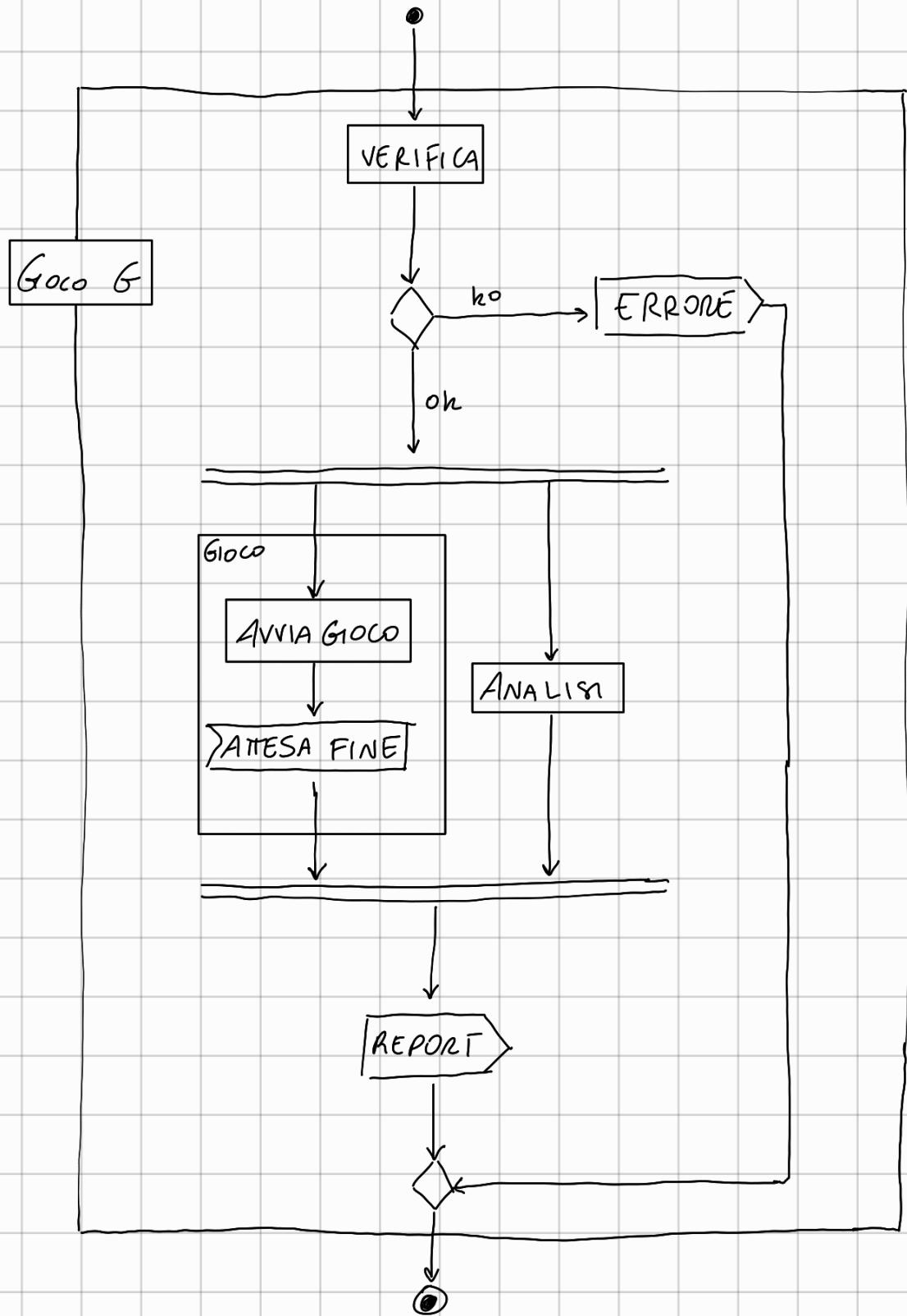
ASSOCIAZIONE	CLASSE	ha RESP.
costituto	Gioco	0
	Giocatore	1
fapante1	Droidé	0
	Equipaggio	1
fapante2	Robot	1
	Equipaggio	1
pilota	Equipaggio	1 0
	Astronave	1
capitano	Umano	1
	Astronave	/

( ➤ verso di lettura)

TIPO MNL	REALIZZ. IN JAVA
stringa	String
int	int
insieme	HashSet



Var aux : astronave: Astronave



SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
Esame del **15 febbraio 2019**  
*Tempo a disposizione: 3 ore*

**Requisiti.** L'applicazione da progettare riguarda una compagnia di noleggio limousine con autista. La compagnia opera attraverso vari depositi. Di ogni deposito interessa la superficie in metri quadri gli autisti che vi operano (almeno uno) e le macchine che gestisce (almeno una). Di ogni autista è di interesse il nome e il tipo di patente (una lettera da B a F) e il deposito a cui afferisce. Delle macchine interessa la targa e se sono disponibili o in riparazione (un booleano). Alcune macchine sono lunghe e richiedono la patente D, altre hanno la struttura di pulmini e richiedono la patente C. Delle macchine lunghe viene memorizzata la lunghezza, di quelle di tipo pulmino il numero massimo di passeggeri. Un cliente, di cui interessa il numero di carta di credito, prenota corse. Le corse hanno un identificativo, un orario e sono relative a una macchina che deve essere disponibile.

Occorre seguire il comportamento degli autisti. Un autista è inizialmente a riposo. Se riceve una richiesta di operare una (corsa) nel caso in cui abbia la patente idonea, si mette alla guida. Quando arriva a destinazione (segnalato da un opportuno evento) torna a riposo. Se invece non ha la patente idonea chiede ad un collega del suo deposito con la patente idonea di sostituirlo (si assuma di avere una funzione predefinita di scelta e che ci sia sempre almeno un collega con la patente idonea disponibile) e si mette in attesa della risposta da parte del sostituto. Quando questa arriva torna a riposo. Se mentre a riposo riceve una richiesta di sostituzione per una corsa da parte di un collega allora si mette alla guida notificando il collega, e come detto prima, quando arriva a destinazione torna a riposo.

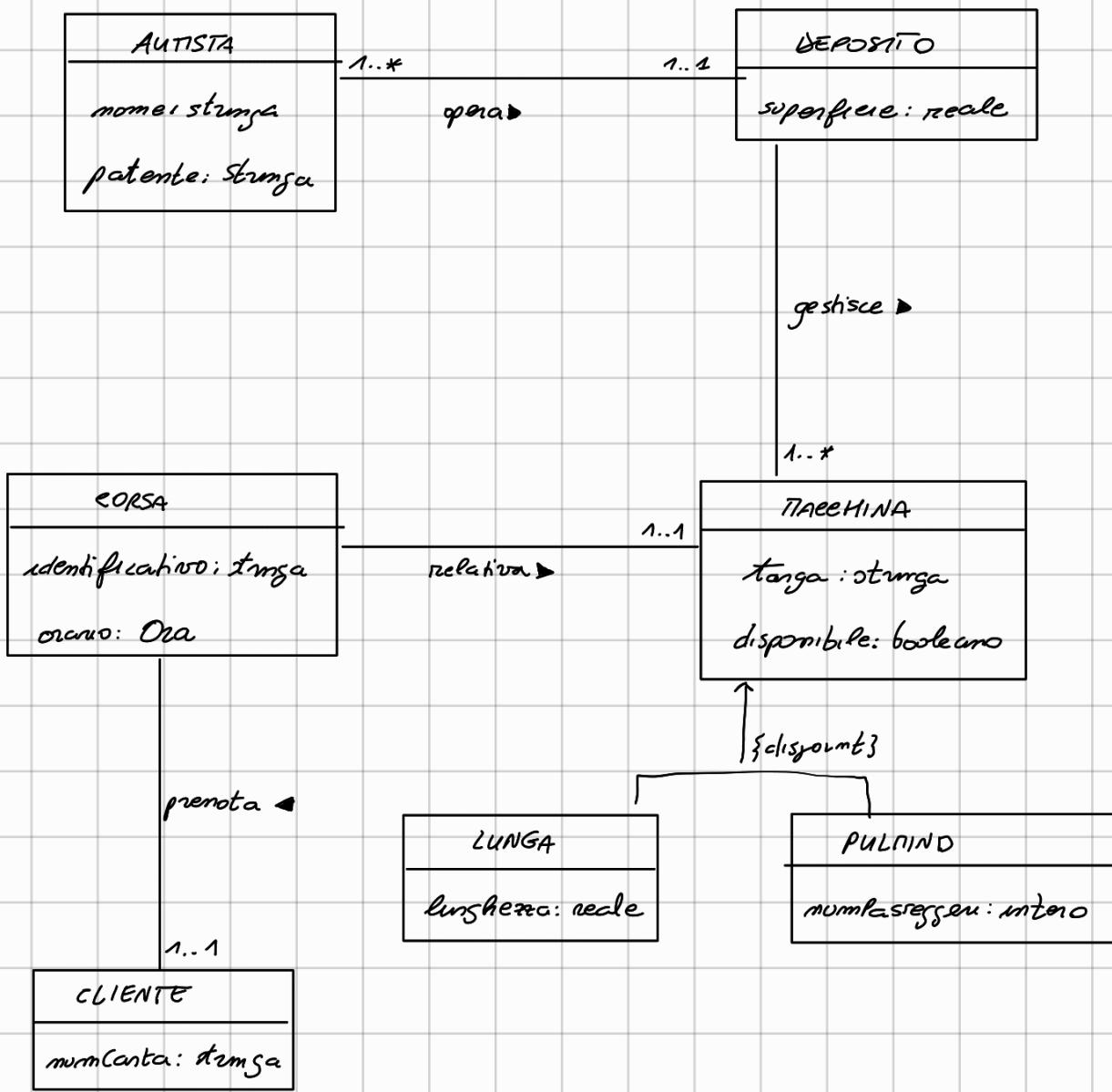
Siamo interessati alla seguente attività principale che viene eseguita giornalmente. Dato un (deposito) all'inizio della giornata, una attività atomica determina l'insieme delle macchine disponibili e di quelle in manutenzione. Questi due insiemi vengono usati poi nelle due sottoattività concorrenti, una di gestione delle corse e l'altra di manutenzione. La prima sottoattività fa partire tutti gli autisti e le macchine disponibili (i dettagli non interessano) e termina quando arriva il segnaletico di I/O di fine giornata. La seconda si occupa di riparare le macchine (i dettagli non interessano) producendo la lista delle macchine che possono tornare in servizio. Quando queste due sottoattività terminano viene stampata la lista delle macchine ancora in riparazione.

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi vincoli non esprimibili in UML); diagramma stati e transizioni per la classe Autista; diagramma delle attività; specifica del diagramma stati e transizioni; segnatura dell'attività principale, sottoattività non atomiche, atomiche e segnali di input/output. Si noti che NON è richiesta la specifica delle attività. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi.

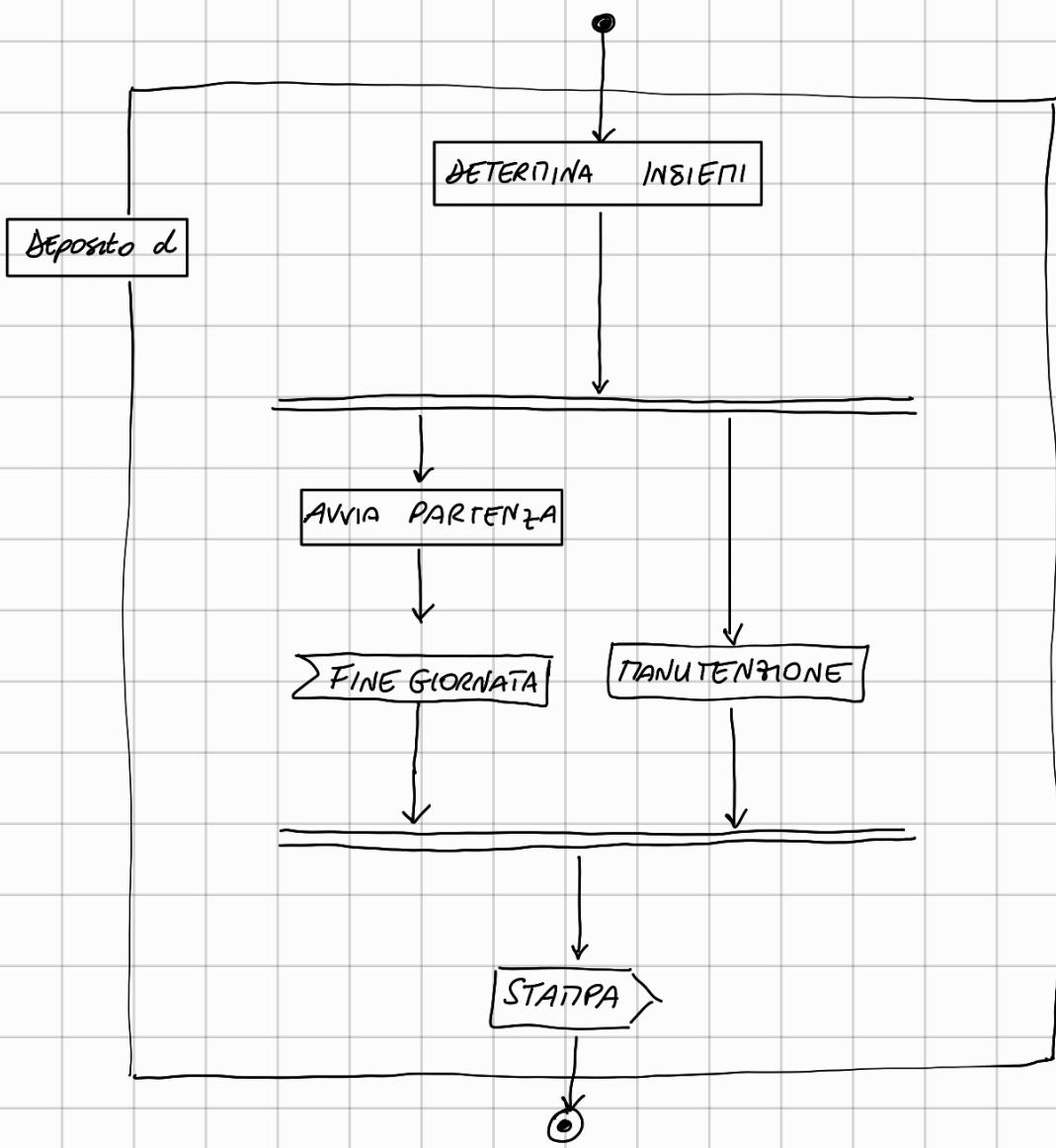
**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

- La classe Autista con classe AutistaFired e le classi JAVA per rappresentare le associazioni di cui la classe Autista ha responsabilità.
- L'attività principale e le sue eventuali sottoattività NON atomiche.



ASSOCIAZIONE	CLASSI	ha RESP?
opera	Autista	Sì "1,0,R"
	deposito	Sì "R,1,0"
gestisce	deposito	Sì "1,0"
	Macchina	No
relativa	Corsa	Sì "1,0"
	Macchina	No
prenota	Cliente	No
	Corsa	Sì "1"

TIPO UNL	REALIZZAZIONE JAVA
stringa	String
intero	int
reale	double
booleano	boolean
insieme	HashSet
Ora	int, int



SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
Esame del **17 giugno 2019**  
*Tempo a disposizione: 3 ore*

**Requisiti.** L'applicazione da progettare riguarda un videogioco basato sul gioco dell'oca. Un **tabellone** è caratterizzato da un link a un file immagine di background (una stringa) una sequenza non vuota di caselle e un insieme di giocatori (almeno 2). Ciascun **giocatore** appartiene ad un solo tabellone e ha un nome (una stringa), ed è posizionato su una casella. Le **caselle** appartengono ad un unico tabellone e hanno un link all'immagine della stessa (una stringa). Le caselle sono suddivise in 3 categorie: caselle **normali**, caselle con **bonus** e caselle di **salto**. Le caselle normali hanno un colore (una stringa). Le caselle con bonus hanno un intero che rappresenta i punti che un giocatore acquisisce fermandosi nella casella. Le caselle con salto sono associate ad un'altra casella dello stesso tabellone cosicché quando un giocatore si ferma in una casella di salto cambia posizione saltando alla casella indicata.

Siamo interessati al comportamento dei giocatori. Un giocatore è inizialmente a **riposo**. Se riceve il comando di **si gioca** si posiziona nella prima casella del tabellone e passa allo stato **in gioco**. Quando **in gioco** riceve l'evento **lancio del dado** con payload il numero **(n)** di caselle si sposta di **n** caselle (se contando **n** si arriva all'ultima casella si ricomincia dalla prima) e se la casella di arrivo è una casella normale non fa altro, se è una casella bonus incrementa il suo punteggio secondo quanto indicato nella casella, e se è una casella di salto, salta alla casella indicata. Infine se quando **in gioco** il giocatore riceve il comando **fine** torna a **riposo**.

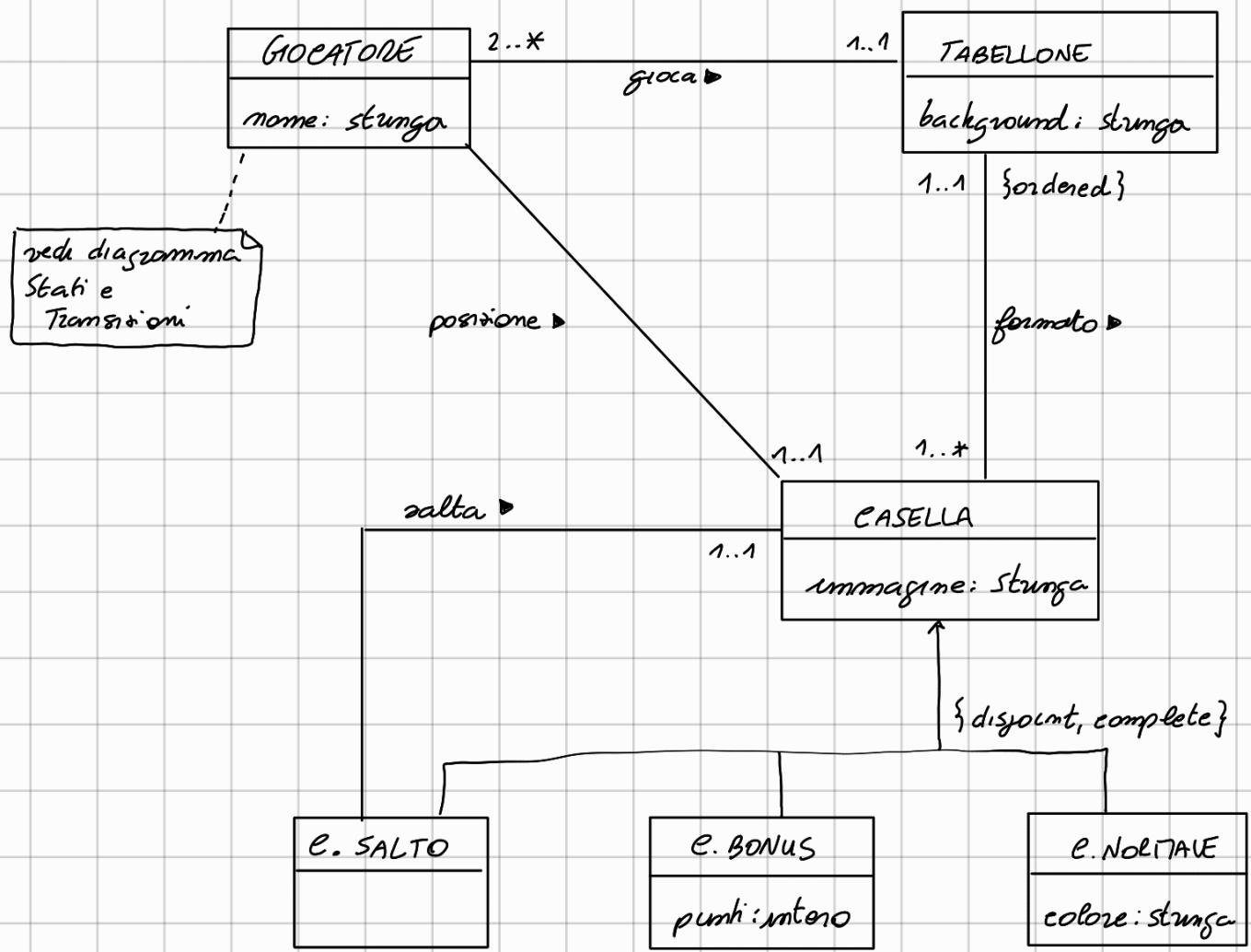
Siamo interessati alla seguente attività principale. L'attività prende in input un **tabellone T** e verifica che il numero di caselle sia almeno 50 e che le posizioni caselle delle varie categorie siano appropriate (i dettagli non interessano). Se la verifica non va a buon fine, l'attività termina segnalando in output un **errore**. Altrimenti concorrentemente esegue le seguenti due sottoattività: *(i) gioco* e *(ii) analisi*. La sottoattività di gioco *(i)* avvia il gioco attivando tutti i giocatori di **T** mandando opportuni eventi (i dettagli non interessano). Poi si mette in **attesa** del segnale di fine-gioco che interrompe il gioco stesso. La sottoattività di analisi *(ii)* calcola un **report** sul tabellone **T** (i dettagli non interessano). Una volta che tali sottoattività sono state completate, l'attività principale **calcola i punti** acquisiti da ogni giocatore durante la sottoattività di gioco, poi manda un segnale di **output** con i **punti** e con il **report** calcolato dalla sottoattività di analisi e infine termina.

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi eventuali vincoli non esprimibili in UML); diagramma stati e transizioni per la classe *Giocatore*; diagramma delle attività; specifica del diagramma stati e transizioni; segnatura dell'attività principale, sottoattività non atomiche, atomiche e segnali di input/output. Si noti che NON è richiesta la specifica delle attività. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi.

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

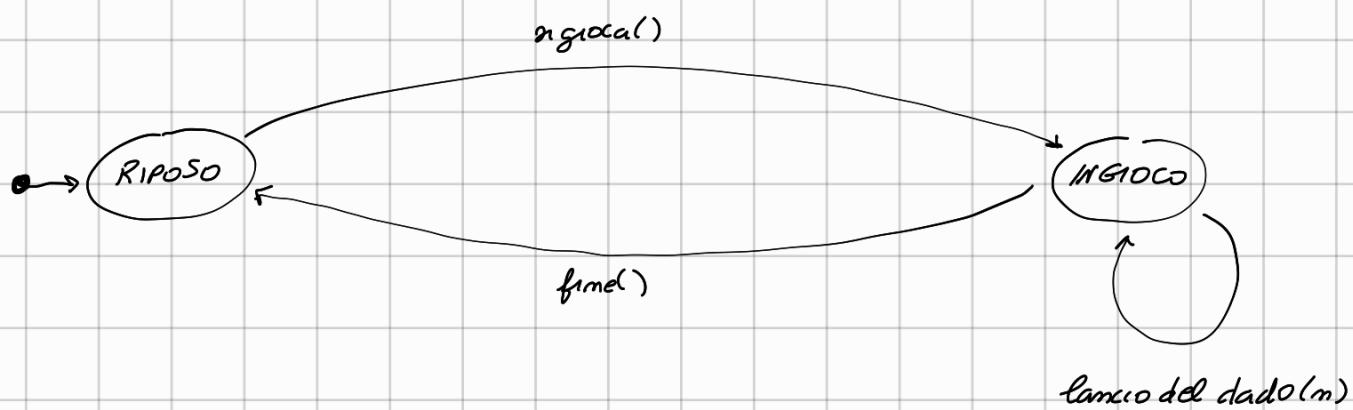
- La classe **Giocatore** con la classe **GiocatoreFired**, le classi JAVA per rappresentare le *associazioni* di cui la classe **Giocatore** ha responsabilità.
- L'*attività principale* e le sue eventuali sottoattività NON atomiche.



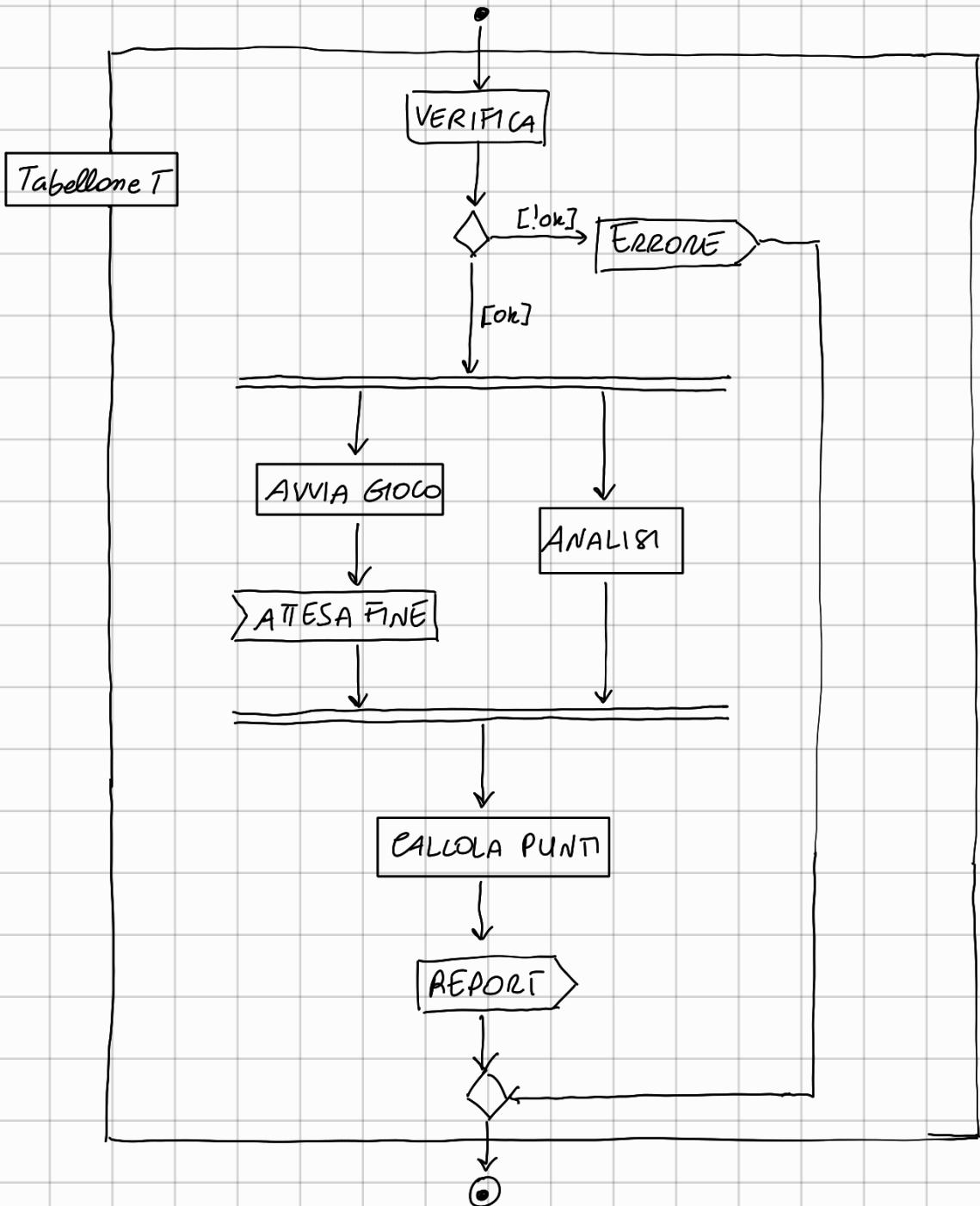
( ➤ verso di lettura )

ASSOCIAZIONE	CLASSE	hARESP
gioca	Giocatore	1 0
	Tabellone	1 0
formato	Tabellone	1 0
	Casella	1
posizione	Giocatore	1 0
	Casella	—
salta	e_SALTO	1 0 R
	Casella	—

TIPO UTL	REALIZZAZIONE
stringa	String
int	int
Insieme	HashSet
lista	LinkedList



Var aux punteggio : inteo



SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
Esame del **12 luglio 2019**  
*Tempo a disposizione: 3 ore*

**Requisiti.** L'applicazione da progettare riguarda una variante del gioco della battaglia navale. Un gioco è caratterizzato da un nome (una stringa), e ha un arbitro e un insieme di giocatori (almeno 2). Dell'arbitro interessa un codice identificativo (un intero). L'arbitro appartiene ad un solo gioco. Ciascun giocatore ha un nome (una stringa) e anche esso appartiene ad un solo gioco. Inoltre ogni giocatore ha un insieme di navi. Le navi hanno un nome (NB: non è di interesse data una nave sapere a quali giocatori appartiene). Le navi sono divise in due categorie: semplici e speciali. Le navi semplici hanno una “posizione annotata”. Una posizione annotata è costituita da due interi (le coordinate) e un booleano che indica se la posizione è stata colpita. Le speciali hanno un insieme non vuoto di posizioni annotate (non necessariamente adiacenti).

Siamo interessati al comportamento dei giocatori. Un giocatore è inizialmente a riposo. Se riceve il comando di si gioca da parte dell'arbitro del gioco annota a false tutte le posizioni delle proprie navi e passa in gioco. Quando in gioco riceve l'evento colpo con payload (due interi) verifica se essi corrispondono alla posizione di una delle sue navi, in caso affermativo mette a true l'annotazione della posizione, altrimenti manda a sua volta un colpo scegliendo in due interi secondo una funzione predefinita i cui dettagli non interessano). Se quando in gioco il giocatore riceve il comando fine da parte dell'arbitro del gioco torna a riposo.

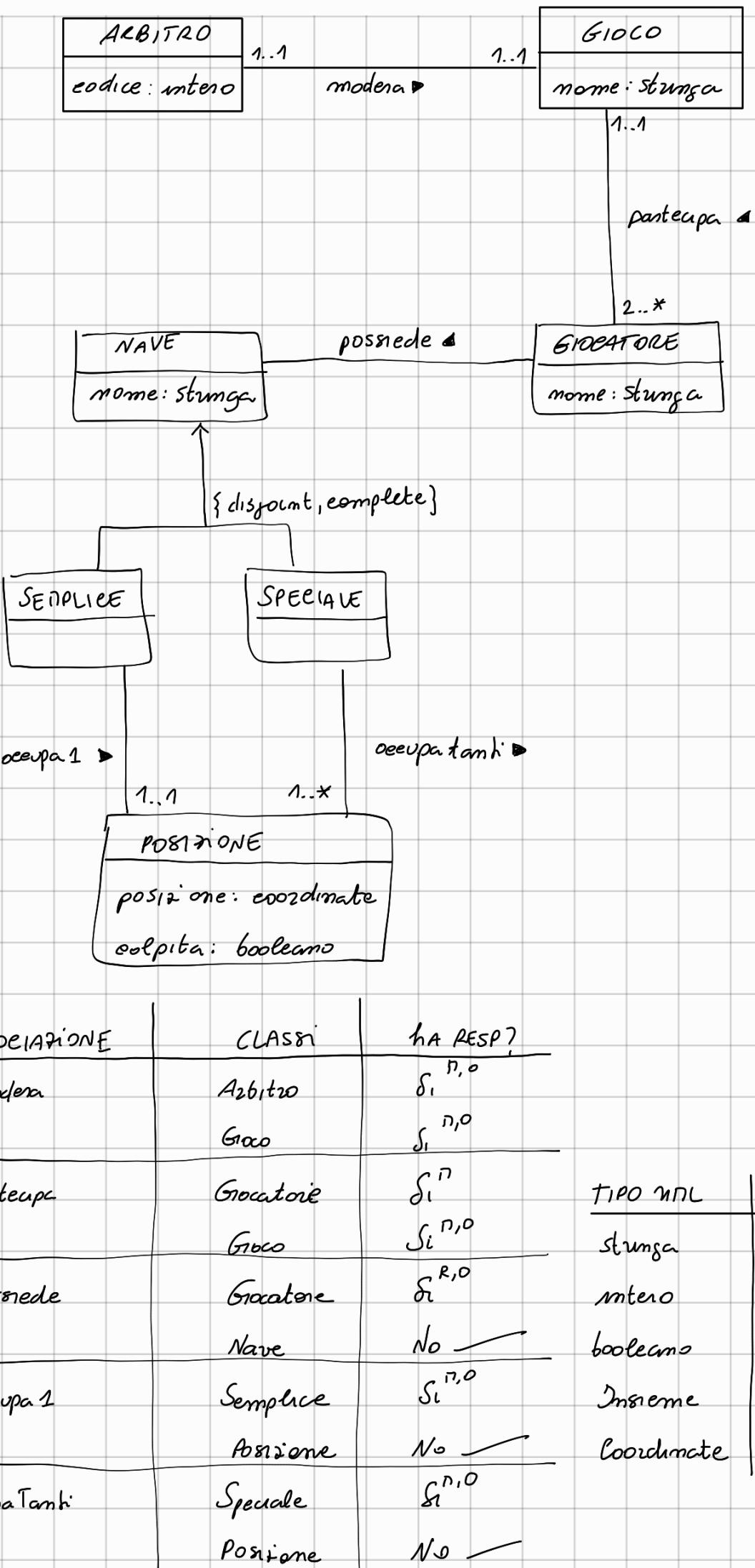
Siamo interessati alla seguente attività principale. L'attività prende in input un (gioco G) e verifica che non ci siano sovrapposizioni tra le posizioni delle navi dei vari giocatori. Se la verifica non va a buon fine, l'attività termina segnalando in output un errore. Altrimenti concorrentemente esegue le seguenti due sottoattività: (i) gioco e (ii) analisi. La sottoattività di gioco (i) avvia il gioco attivando tutti i giocatori di G e l'arbitro mandando opportuni eventi (i dettagli non interessano). Poi si mette in attesa del segnale di fine-gioco che interrompe il gioco stesso. La sottoattività di analisi (ii) calcola un report (una stringa) con l'elenco dei nomi delle navi dei vari giocatori di G. Una volta che tali sottoattività sono state completate, l'attività principale calcola il vincitore che ha vinto, cioè colui le cui navi hanno complessivamente meno posizioni colpite, poi manda un segnale di output con il report calcolato dalla sottoattività di analisi con in aggiunta il nome del vincitore.

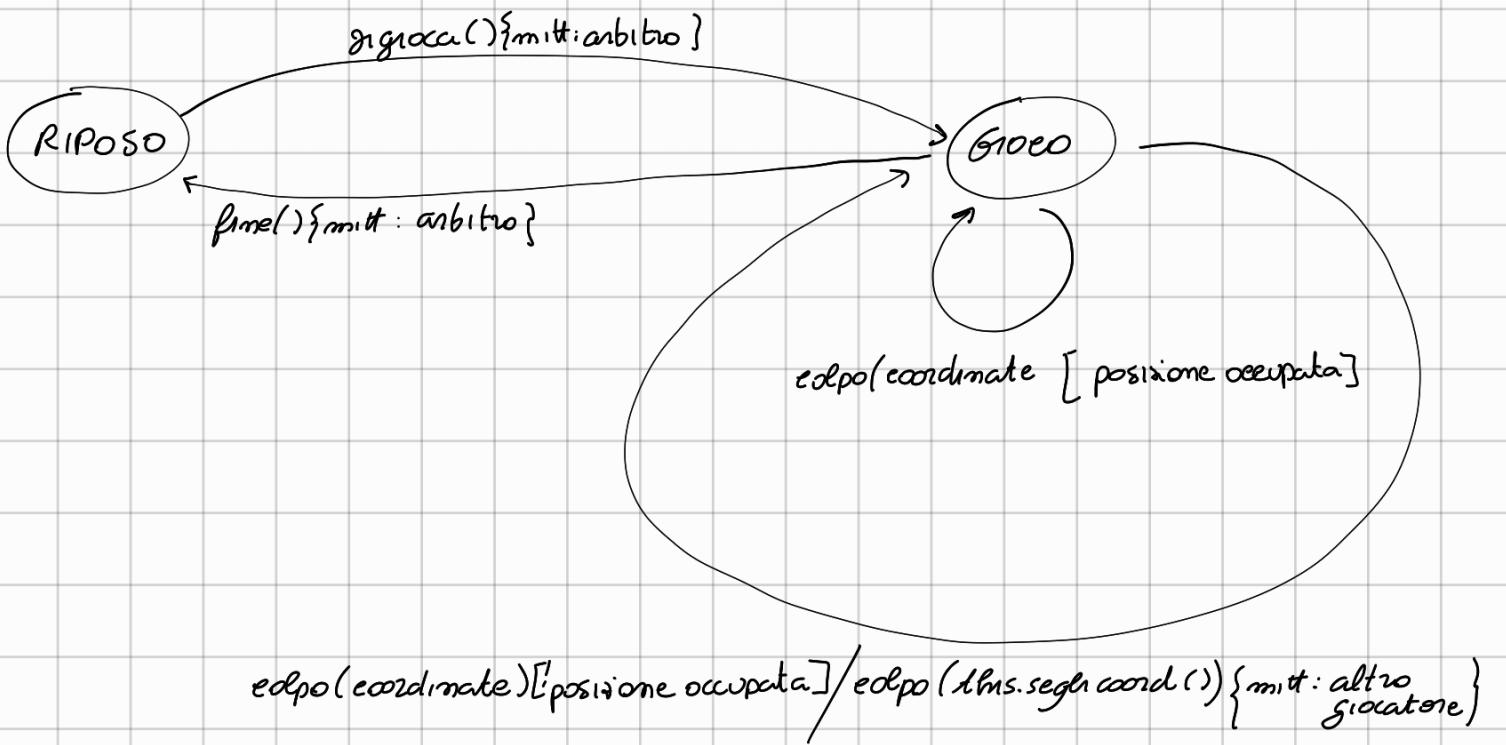
**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi eventuali vincoli non esprimibili in UML); diagramma stati e transizioni per la classe Giocatore; diagramma delle attività; specifica del diagramma stati e transizioni; segnatura dell'attività principale, sottoattività non atomiche, atomiche e segnali di input/output. Si noti che NON è richiesta la specifica delle attività. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

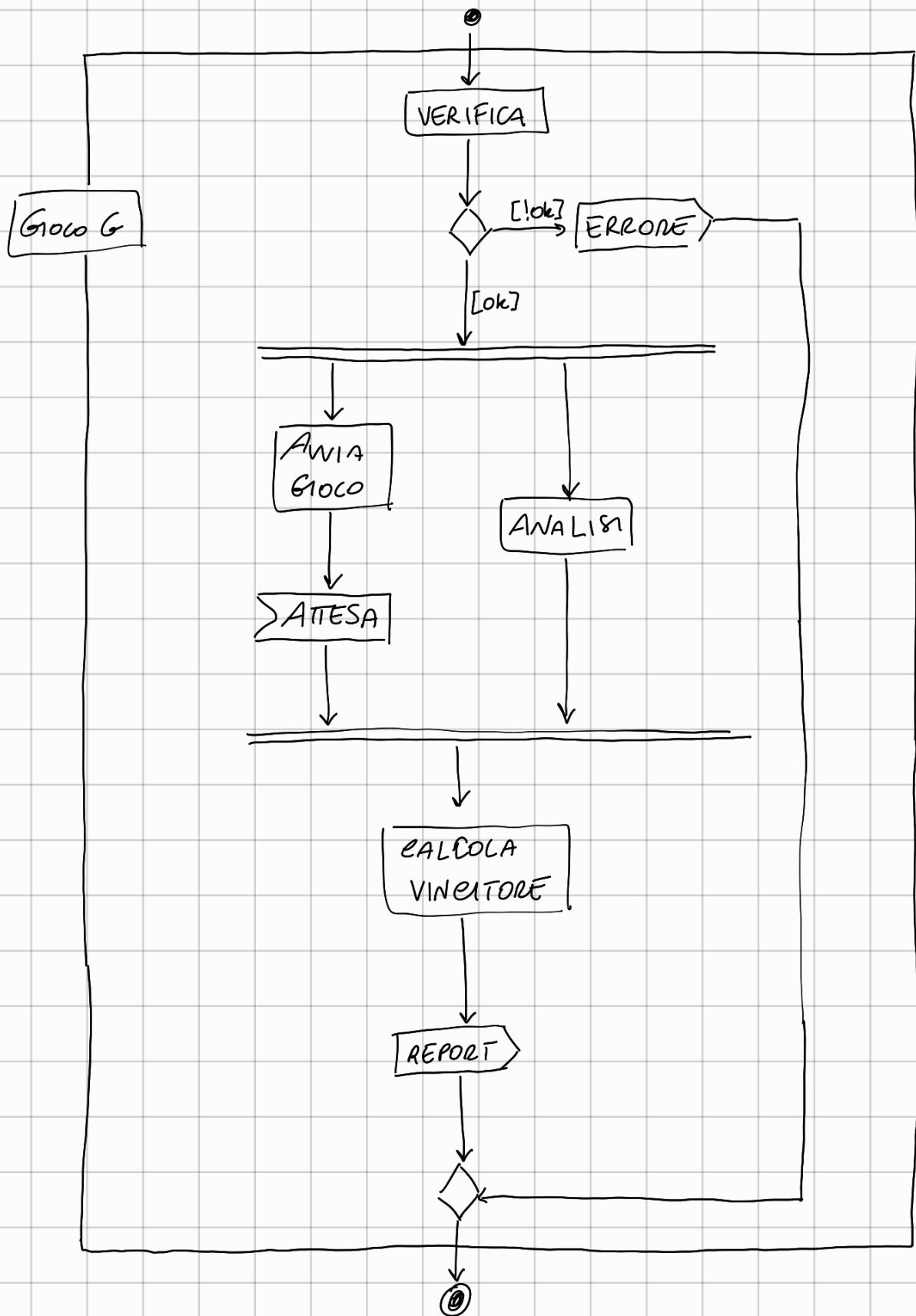
**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi.

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

- La classe Giocatore con la classe GiocatoreFired, le classi JAVA per rappresentare le associazioni di cui la classe Giocatore ha responsabilità.
- L'attività principale e le sue eventuali sottoattività NON atomiche.







SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
Esame del **10 settembre 2019**  
*Tempo a disposizione: 3 ore*

**Requisiti.** L'applicazione da progettare riguarda un sistema ferroviario. Ogni vagone di un treno è collegato al precedente e al successivo. Il vagone di testa non ha un precedente e l'ultimo non ha un successivo. Di ogni vagone interessa il codice (una stringa). Un treno di cui interessa il nome (una stringa) contiene una serie di vagoni, di cui interessa solo il primo dal momento che i successivi sono semplicemente quelli che seguono. Non è di interesse risalire dal vagone al treno a cui appartiene, solo l'opposto. Alcuni vagoni sono trainanti, e di questi interessa il numero di CV di potenza (un intero). Altri sono vagoni letto e di questi interessa il numero di posti letto. Ogni treno ha un certo numero arbitrario di dipendenti ad esso assegnati. Di ogni dipendente interessa il nome e gli anni di servizio. Ogni treno ha un dipendente con il ruolo di capotreno.

Una parte specifica dell'applicazione riguarda il movimento dei vagoni. In particolare, un vagone può trovarsi in sosta o in movimento. La transizione dalla sosta al movimento viene comunicata con un evento partenza, mentre da movimento a sosta dall'evento arrivo. Un vagone in sosta che non sia attaccato a nessun altro può venire attaccato in coda a un altro vagone. Questo evento si chiama sposta con payload il(vagone)a cui attaccarsi e causa la verifica che quest'ultimo sia effettivamente un vagone di coda, ossia non ha già un successivo. La verifica avviene attraverso il seguente protocollo: il vagone da attaccare invia un evento arrivo al vagone a cui va attaccato; questo risponde libero oppure occupato. Solo nel primo caso lo spostamento avviene (con gli opportuni aggiornamenti del diagramma delle classi), nel secondo no.

Siamo interessati alla seguente attività principale. L'attività prende in input un(treno T) e un(insieme V) di vagoni e verifica che ciascun vagone non sia collegato ad altri e che sia presente almeno un vagone motorizzato. Se la verifica non va a buon fine, l'attività termina segnalando in output un errore. Altrimenti concorrentemente esegue le seguenti due sottoattività: (i) formazione e (ii) orario. La sottoattività di formazione del treno (i) forma il treno eseguendo spostamenti dei vagoni in V (i dettagli non interessano). La sottoattività chiede l'orario di orario di partenza e la tratta da percorrere e calcola l'orario di arrivo dei T (i dettagli non interessano). Quando entrambe le sottoattività sono terminate, viene stampato l'orario di arrivo del treno.

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi eventuali vincoli non esprimibili in UML); diagramma stati e transizioni per la classe Vagone; diagramma delle attività; specifica del diagramma stati e transizioni; segnatura dell'attività principale, sottoattività non atomiche, atomiche e segnali di input/output. Si noti che NON è richiesta la specifica delle attività. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

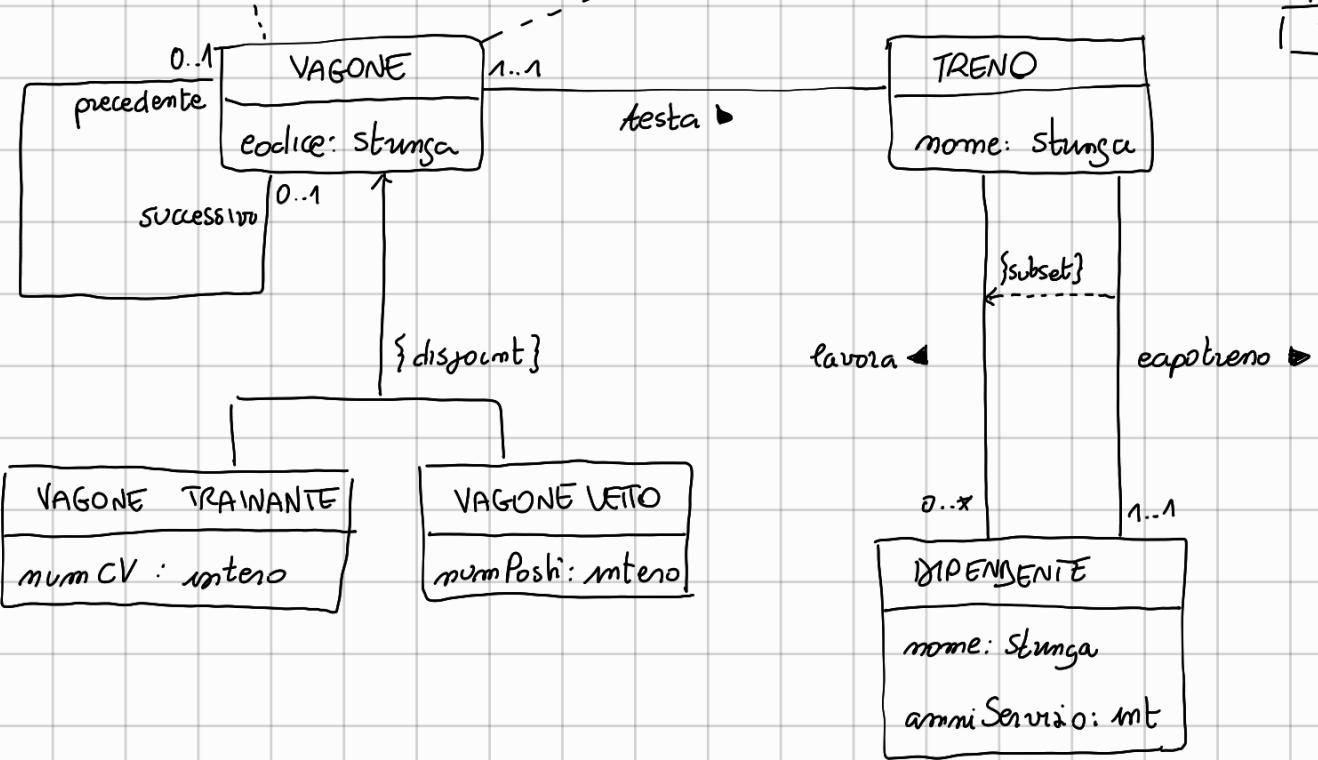
**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi (nella tabella, inserire anche il motivo di ognuna delle responsabilità).

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

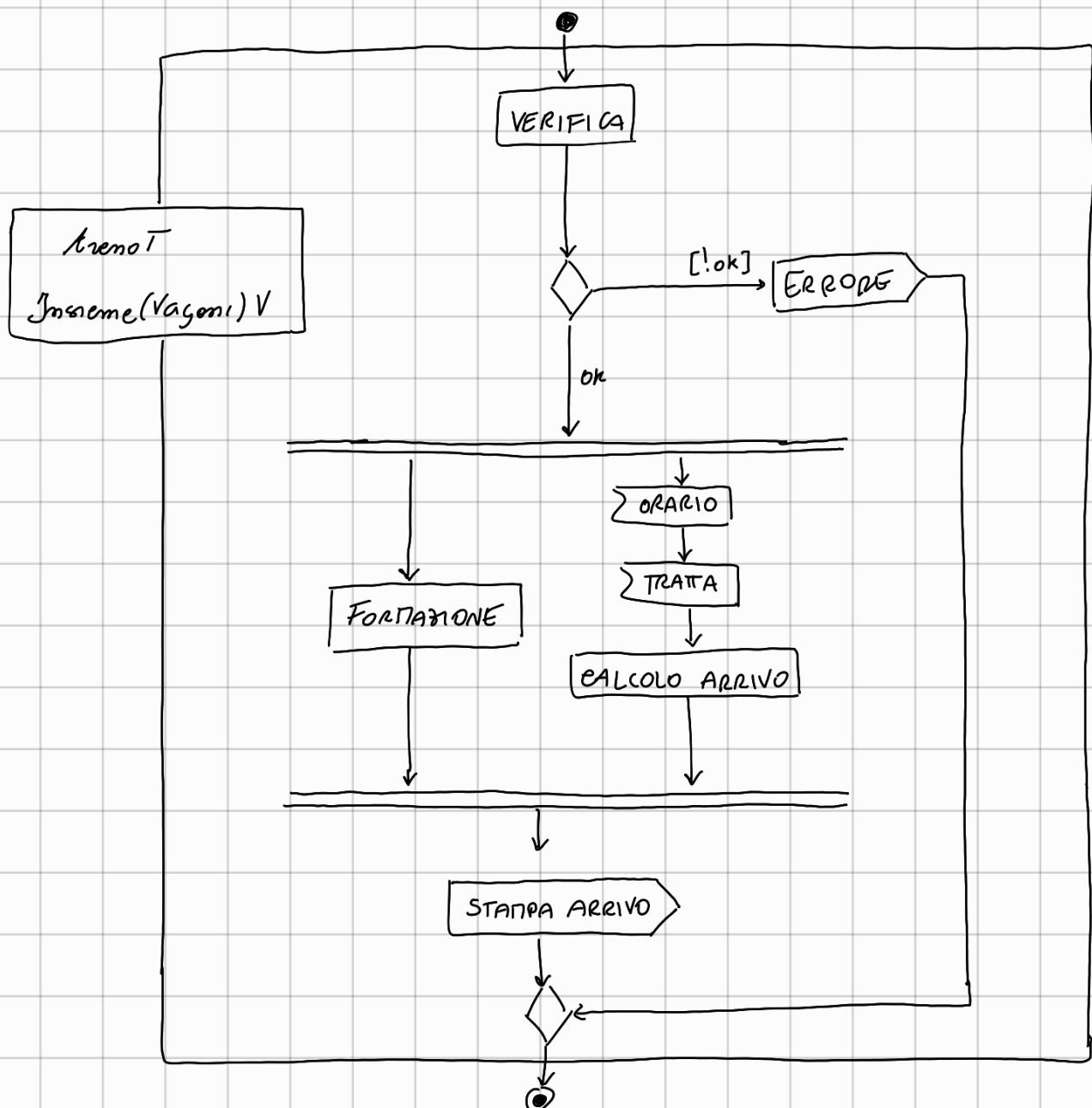
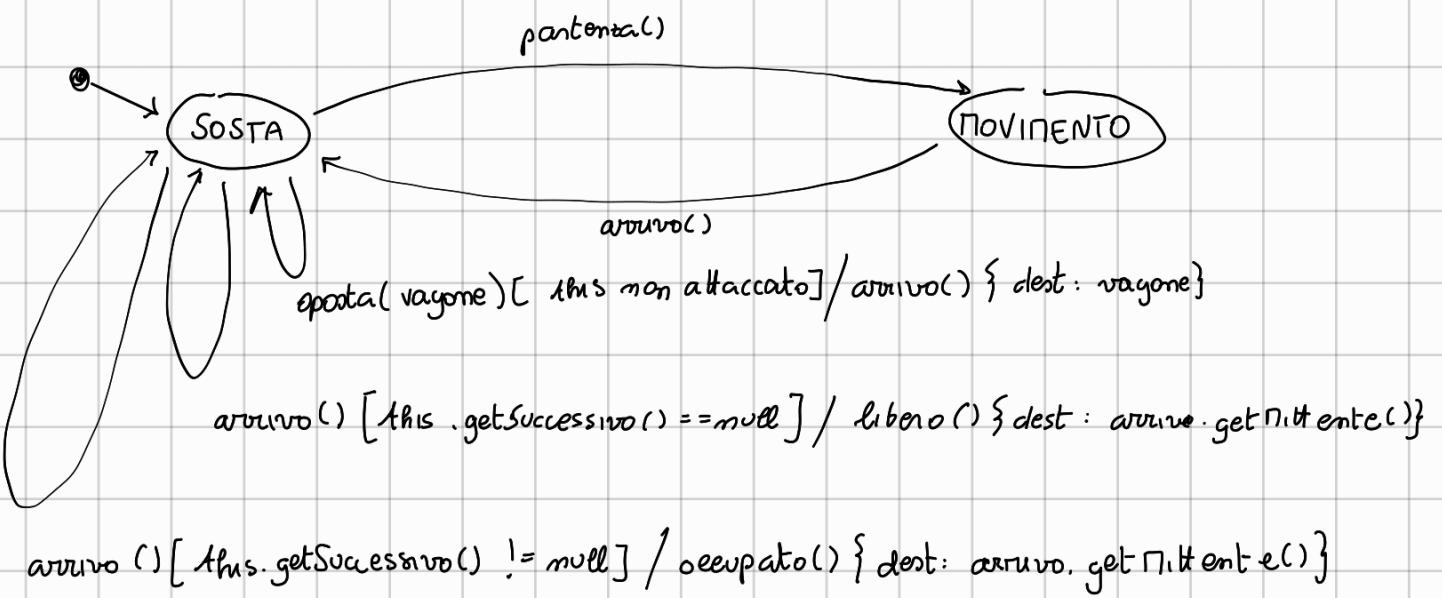
- La classe Vagone con la classe VagoneFired, le classi JAVA per rappresentare le associazioni di cui la classe Vagone ha responsabilità.
- L'attività principale e le sue eventuali sottoattività NON atomiche.

il primo vagone non ha precedente, l'ultimo non ha successivo

vedi diagramma Stati e Transizioni



ASSOCIAZIONE	CLASSE	HA RISP?
testa	Vagone	No <sup>R</sup>
	Treno	Sì <sup>R, I, O</sup>
lavora	Dipendente	No
	Treno	Sì <sup>R</sup>
capotreno	Dipendente	No
	Treno	Sì <sup>I</sup>
connessione	Vagone precedente	Sì <sup>I, O</sup>
	Vagone successivo	Sì <sup>I, O</sup>



SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
Esame del **29 gennaio 2020**  
*Tempo a disposizione: 3 ore*

**Requisiti.** L'applicazione da progettare riguarda una parte di un videogioco di biliardo robotizzato. Un gioco è costituito da almeno due giocatori e un tavolo. Ciascun **giocatore** ha un nome (una stringa) ed un insieme non vuoto di palle robotizzate. Un **tavolo** è rettangolare ed è caratterizzato da una lunghezza e una larghezza ed un insieme non vuoto di buche, ciascuna in una posizione del tavolo (determinate da coordinate cartesiane). Ciascuna buca appartiene esclusivamente ad un tavolo e può essere prenotata da una palla. Ciascuna **palla robotizzata** appartiene ad esattamente un giocatore, ha un nome e una posizione corrente sul tavolo (in coordinate cartesiane) e può muoversi per cambiare posizione. Le palle sono divise in **“pesanti”** e **“leggere”** e quelle leggere hanno un colore (una stringa), quelle pesanti invece non hanno un colore perché sono cromate. Tutti i movimenti sono caratterizzati dalle leggi della fisica, ma i dettagli per questo compito non interessano. Tuttavia la massa delle palle leggere è trascurabile rispetto a quelle pesanti e in una collisione queste ultime NON modificano la propria traiettoria (mentre quelle leggere sì).

Siamo interessati al comportamento delle palle. Una palla è inizialmente ***inattiva***. Se riceve dal proprio giocatore il comando **vai in buca** con payload la buca in cui deve andare, chiede alla buca di prenotarla mettendosi in **attesa** della risposta. Se la buca gli risponde **prenotata** (questo significa che era libera e ora è prenotata per la palla in questione), allora si mette in **movimento** verso la buca; altrimenti torna ***inattiva*** segnalando al proprio giocatore il fallimento. Quando la buca gli dice che l'ha **raggiunta**, mette come propria posizione corrente la posizione della buca, e diviene di nuovo ***inattiva***. Se mentre è in **movimento** riceve un evento **collisione** con payload un'altra palla, se lei stessa è una palla pesante e l'altra palla è leggera, ignora l'evento; altrimenti si mette in uno stato **temporaneo** e quando si **ferma** (un evento esterno con payload la posizione), modifica la propria posizione corrente nella nuova e si mette nello stato di ***inattiva***.

Siamo interessati alla seguente attività principale. L'attività prende in input un gioco **G** e **verifica** che il numero di palle pesanti sia al massimo il 10% del numero di palle leggere. Se la verifica non va a buon fine, l'attività termina segnalando in output un **errore**. Altrimenti concorrentemente esegue le seguenti due sottoattività: *(i)* **gioco** e *(ii)* **analisi**. La sottoattività di gioco *(i)* **avvia il gioco** attivando tutti i giocatori, palle e buche di **G** mandando opportuni eventi (i dettagli non interessano). Poi si mette in **attesa del segnale di fine-gioco** che interrompe il gioco stesso. La sottoattività di analisi *(ii)* calcola i colori delle palle leggere presenti nel gioco e per ciascuno il numero di palle di quel colore e prepara un report con questi dati (una stringa). Una volta che tali sottoattività sono state completate, l'attività principale manda un **segnale di output** con il report calcolato nella sottoattività di analisi e termina.

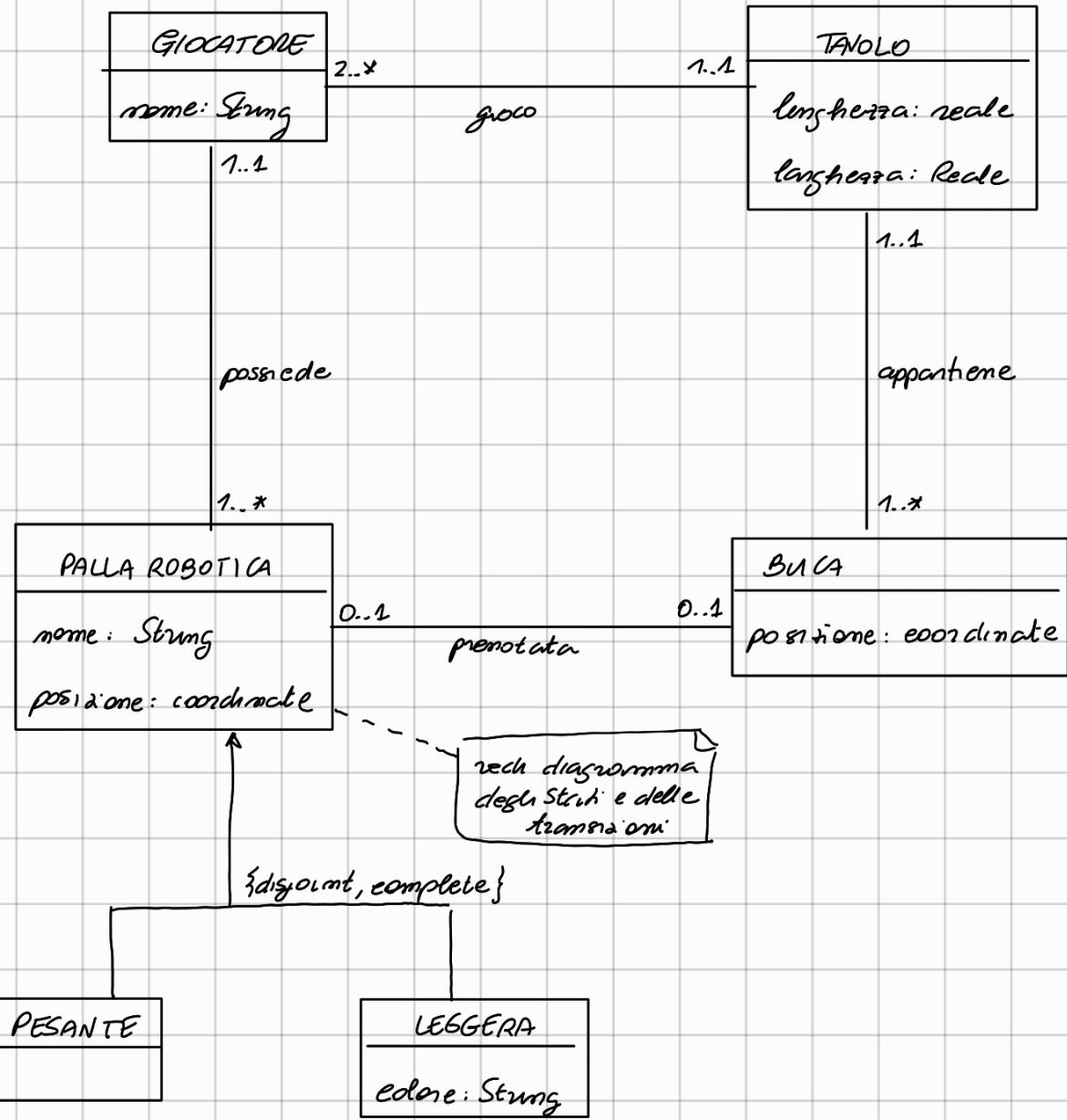
**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi vincoli non esprimibili in UML); diagramma stati e transizioni per la classe **Palla**; diagramma delle attività; specifica del diagramma stati e transizioni; segnatura dell'attività principale, sottoattività non atomiche, atomiche e segnali di input/output. Si noti che NON è richiesta la specifica delle attività. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi, con i relativi motivi (molteplicità, operazioni, requisiti).

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

- La classe **Palla** con classe **PallaFired**, le eventuali sottoclassi, e le classi JAVA per rappresentare le **associazioni** di cui la classe **Palla** ha responsabilità.
- L'**attività principale** e le sue eventuali sottoattività NON atomiche.

## Diagramma delle classi

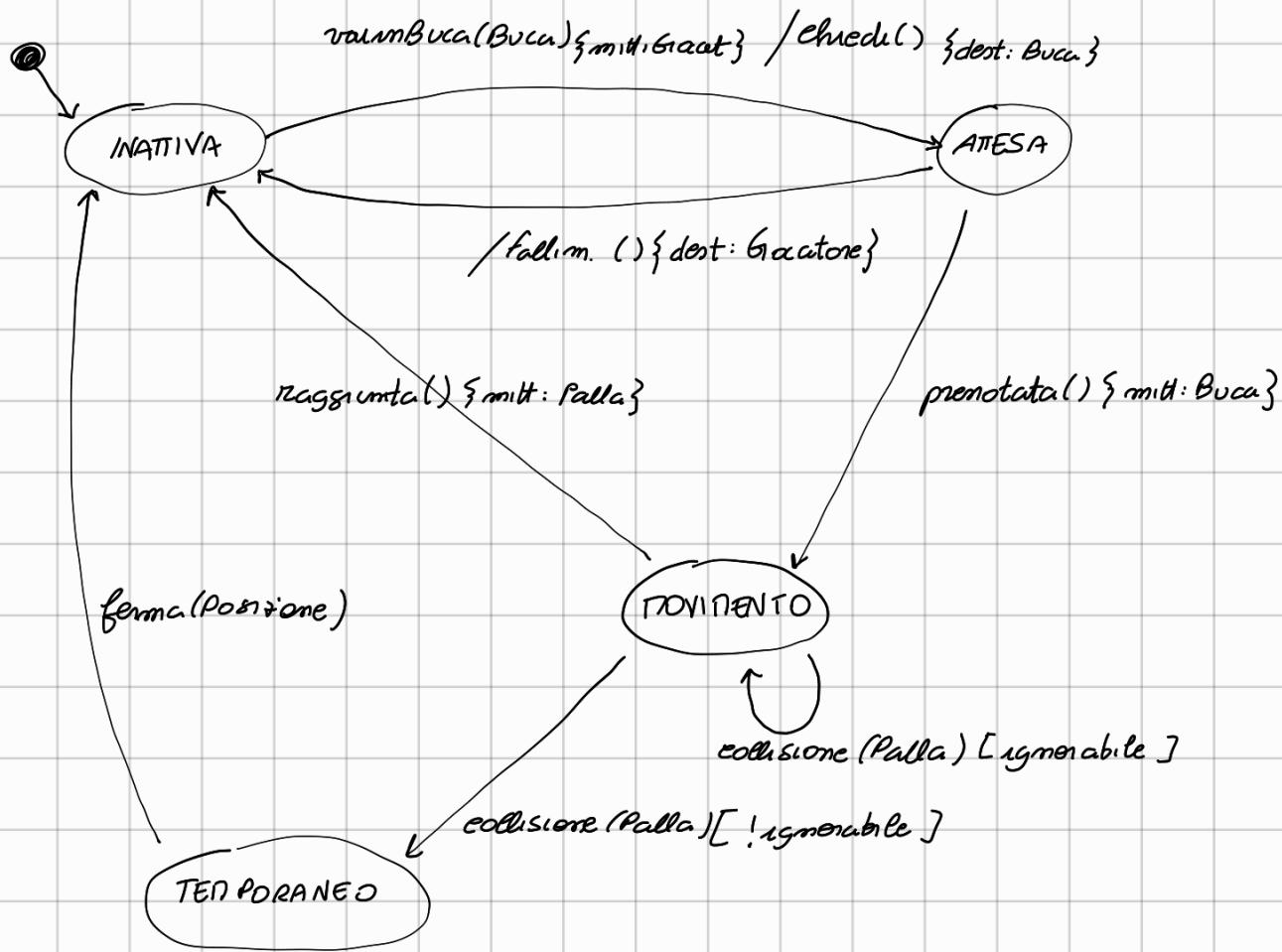


## Tabella responsabilità

ASSOCIAZIONE	CLASSI	RESPONSABILITÀ
gioco	Giocatore	▷
	Tavolo	▷
possiede	Giocatore	▷ 0
	Palla_Robotica	▷ 0
prenotata	Buca	▷ 0
	Palla_Robotica	▷ 0
appartenente	Buca	▷
	Tavolo	▷

O perché:  
per inviare i messaggi (eventi)  
bisogna poter ricevere  
l'ossoetto collegato

Diagramma Stati e transizioni classe PALLA

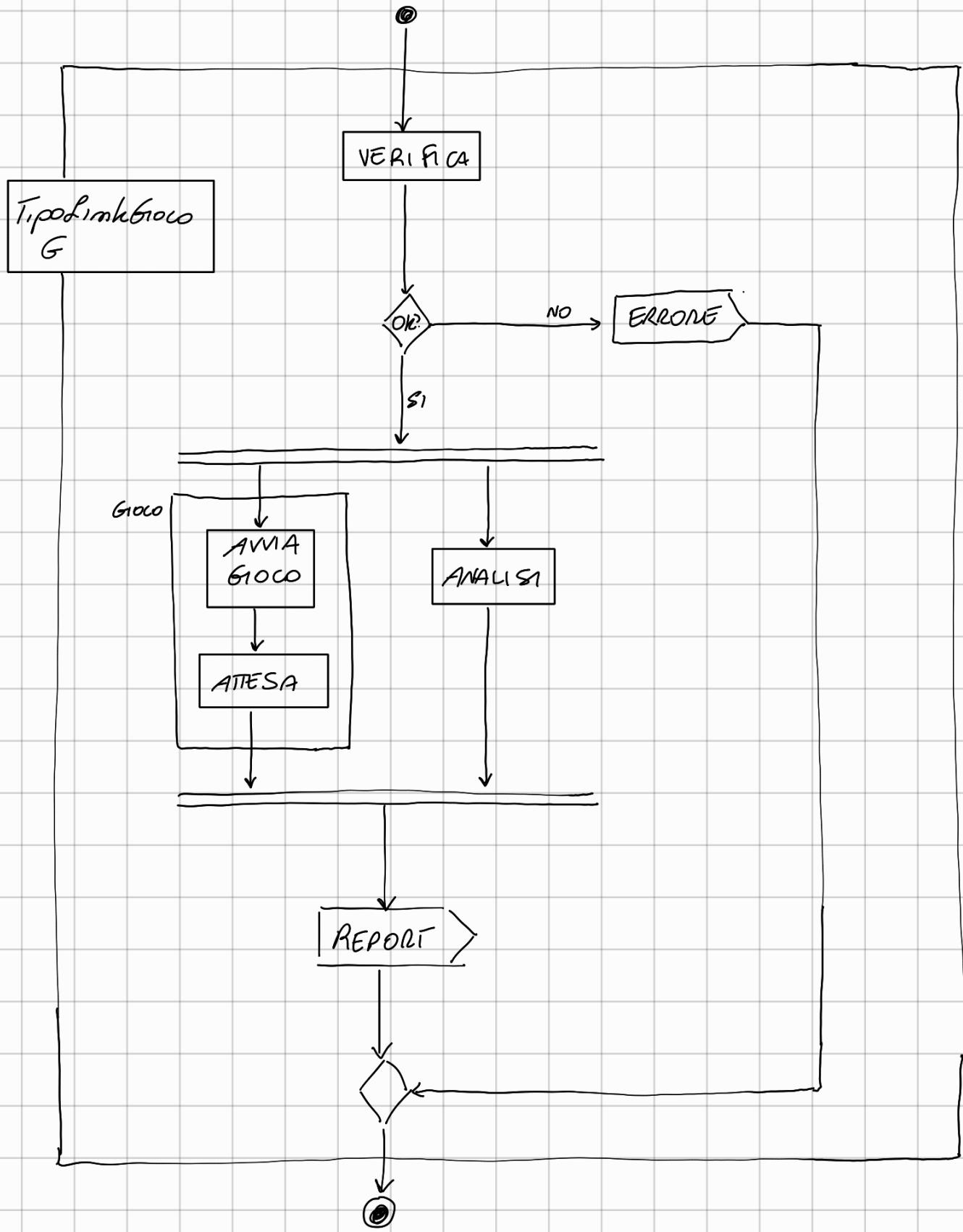


Var Aux: Palla p

NB ignorabile = (*this* == PallaPesante || Palla == PallaLeggera)

TIPO UML	REALIZZ. IN JAVA
Stringa	String
intero	int
reale	double
Coordinate	double, double

diagramma delle attività



SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
Esame del **26 giugno 2020**  
*Tempo a disposizione: 3 ore*

**Requisiti.** L'applicazione da progettare riguarda un sistema di sviluppo software cooperativo.

Gli elementi centrali del sistema sono gli **utenti** e i **progetti software** che sviluppano. Degli utenti interessa il nome e l'email. Dei progetti interessa il nome e la data di creazione. Ogni utente può creare un numero arbitrario di progetti (anche nessuno). Ogni progetto ha un unico **creatore** ma può avere un arbitrario di altri manutentori. Il creatore è egli stesso un manutentore. I manutentori di un progetto sono al massimo venti. Un progetto contiene dei **file** (inizialmente nessuno); ogni file appartiene a un solo progetto. A un progetto possono venire fatte delle richieste di modifica. Ognuna di queste **richieste** è costituita da un codice (un intero) e una descrizione (una stringa). Una richiesta viene creata da un utente qualsiasi (che può o meno essere un manutentore del progetto), ed è associata a uno o più file; dato un file, non è di interesse sapere in quali richieste è coinvolto. Le richieste possono essere bloccanti o meno. Le richieste bloccanti hanno una priorità (un intero).

Una parte specifica dell'applicazione riguarda il meccanismo delle richieste. Un utente crea la richiesta, che si trova inizialmente nello stato di **creata**. Lo stesso utente può decidere di chiuderla oppure di inviarla. Nel primo caso va nello stato **chiusa**, altrimenti **inviata**. Quando è nello stato **inviata**, un manutentore del progetto può decidere di chiuderla (non interessa sapere se viene accettata o meno), oppure può mandare un suggerimento di modifica all'autore della richiesta. In questo caso, la richiesta passa nello stato **attesa**. L'autore della richiesta può decidere di chiuderla oppure di effettuare la modifica e far tornare la richiesta nello stato di **inviata**; in entrambi i casi, viene informato il manutentore che ha mandato il suggerimento.

Siamo interessati all'attività di rilascio di una versione del progetto software. In questa attività viene prima **verificato** che non ci siano richieste bloccanti. Se questo controllo fallisce viene stampato un messaggio di **errore** e l'attività termina. Se invece il controllo riesce vengono eseguite in parallelo due sottoattività; nella prima, il progetto viene **compilato** e poi **testato**; nel secondo, il progetto viene **assemblato** in un singolo archivio. Quando entrambe le sottoattività sono concluse, se il test è andato a buon fine il pacchetto viene rilasciato e viene stampato un messaggio di **conferma**. In caso contrario viene stampato un messaggio di **errore**.

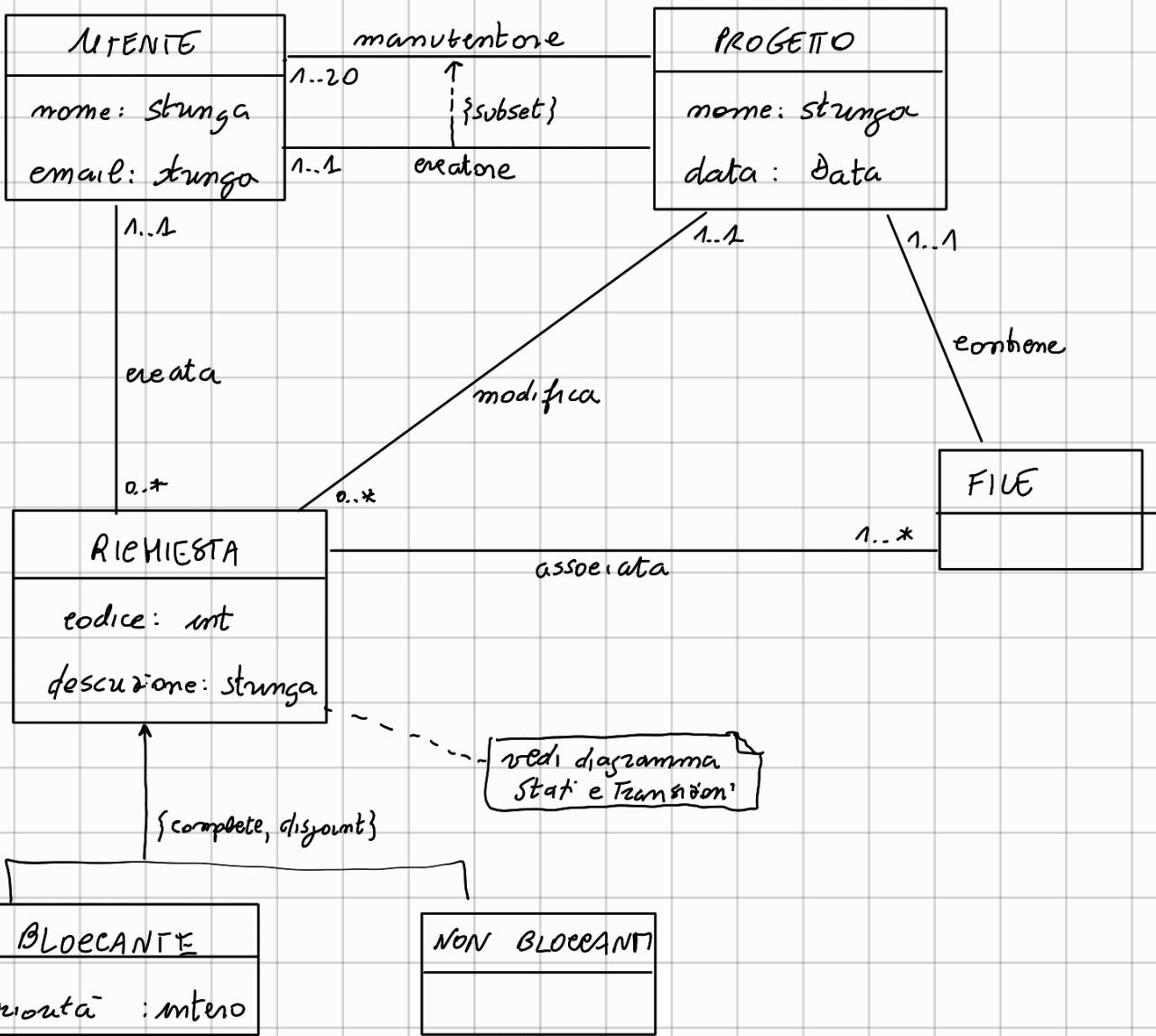
**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi eventuali vincoli non esprimibili in UML); diagramma stati e transizioni per la classe *Richiesta*; diagramma delle attività di rilascio; specifica del diagramma stati e transizioni; specifica completa dell'attività di rilascio, sottoattività non atomiche, atomiche e segnali di input/output. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi (nella tabella, inserire anche il motivo di ognuna delle responsabilità).

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

- La classe **Richiesta** con la classe **RichiestaFired**, le classi JAVA per rappresentare le **associazioni** di cui la classe **Richiesta** ha responsabilità.
- L'**attività di rilascio** e le sue eventuali sottoattività NON atomiche.

## diagramma classi

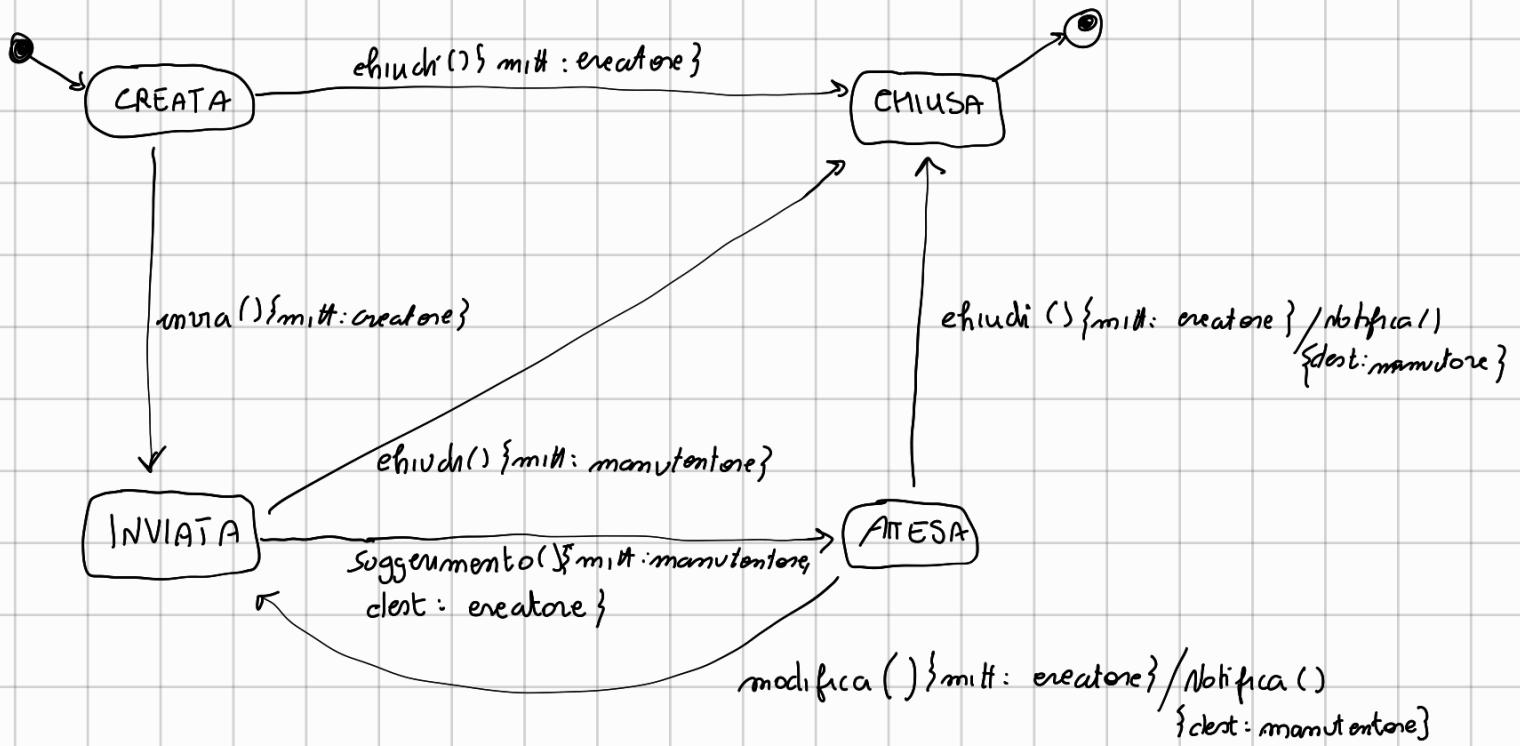


TIPO UTIL	REALIZZ. JAVA
stringa	String
intero	int
Data	int, int, int

## Tabella responsabilità

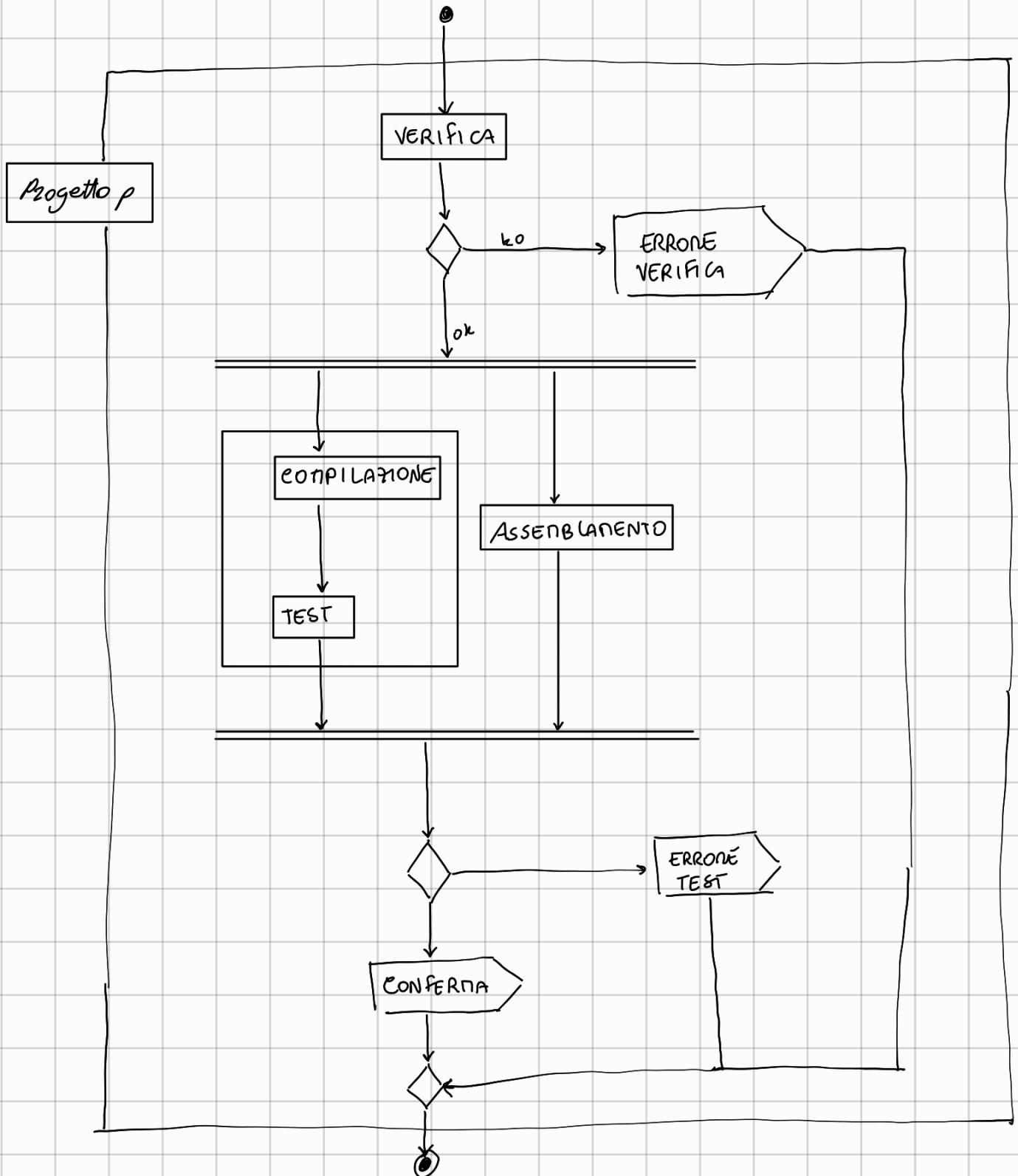
ASSOCIAZIONE	CLASSI	RESPONSABILITÀ
manutentore	Utente	O
	Progetto	M
creatore	Utente	/
	Progetto	M
contiene	Progetto	/
	File	M
modifica	Richiesta	M
	Progetto	O
associata	Richiesta	M
	File	/
creata	Richiesta	M O
	Utente	O

diagramma degli Stati e delle transizioni Richiesta



Var Aux  
(manutentore: Utente)

diag attività



SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
Esame del **24 luglio 2020**  
*Tempo a disposizione: 3:30 ore*

**Requisiti.** L'applicazione da progettare riguarda un sistema per gestire campagne pubblicitarie.

Delle campagne pubblicitarie interessano il nome, una descrizione (una stringa) e i creativi che ci lavorano. Di questi interessano il nome e l'email. Tra i creativi che lavorano ad una campagna pubblicitaria, uno è il responsabile della campagna. Ogni campagna pubblicitaria ha esattamente un responsabile ed un numero arbitrario di altri creativi che ci lavorano. Le campagne pubblicitarie sono costituite da diverse pubblicità. Ogni pubblicità è associata ad esattamente una campagna pubblicitaria. Ad una campagna pubblicitaria possono venire associate delle idee, costituite da un titolo (una stringa) ed una descrizione (una stringa). Ogni idea è associata ad esattamente una campagna pubblicitaria ed è creata da esattamente un creativo, che può o meno lavorare alla campagna pubblicitaria stessa. Ogni creativo può creare al più 100 idee. Alcune idee sono dirompenti in quanto richiedono un cambiamento profondo della campagna pubblicitaria stessa ed in tal caso devono indicare una descrizione aggiuntiva che giustifichi il cambiamento proposto (una stringa).

Siamo interessati allo gestione delle idee. Una idea si trova inizialmente nello stato *creata*. Il suo creatore può decidere di chiuderla oppure di sostenerla. Nel primo caso va nello stato *chiusa*, altrimenti va nello stato *sostenuta*. Quando è nello stato *sostenuta*, un lavoratore della campagna pubblicitaria può decidere di chiuderla, oppure può mandare una richiesta di approfondimenti al suo creatore. In questo caso, l'idea passa nello stato *attesa*. Il creatore dell'idea può decidere di chiuderla oppure di aggiornare la descrizione dell'idea stessa, in tal caso l'idea manda una notifica al lavoratore che lo ha richiesto e torna nello stato *sostenuta*.

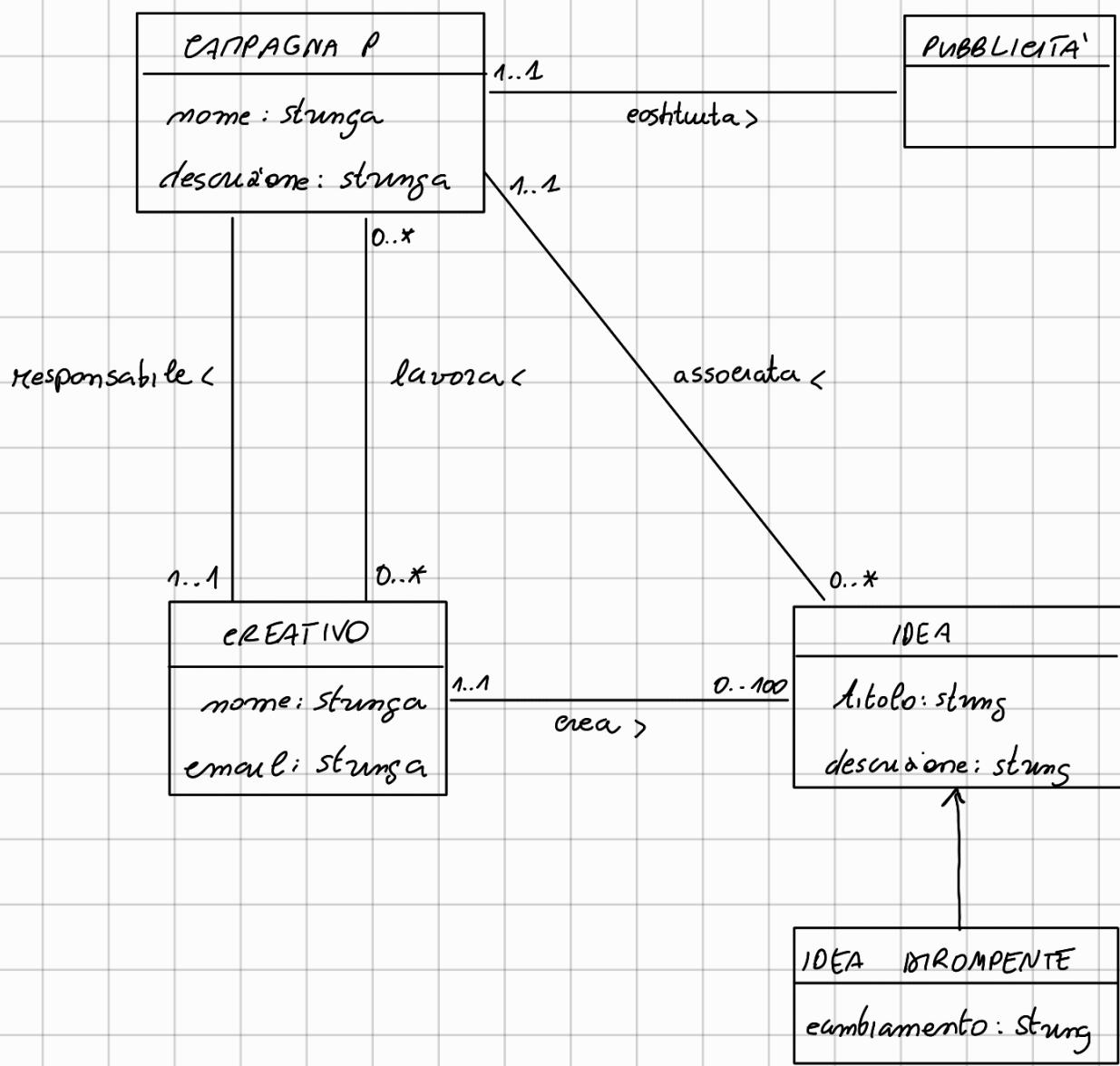
Siamo interessati all'attività di rilascio di una campagna pubblicitaria. In questa attività viene prima verificato che non ci siano idee dirompenti. Se questo controllo fallisce, viene stampato un messaggio di errore e l'attività termina. Se invece il controllo riesce, vengono eseguite in parallelo due sottoattività (i cui dettagli non interessano); nella prima, la campagna pubblicitaria viene pubblicata per avere nuove idee per un certo periodo al termine del quale viene preparato un elenco delle idee proposte (una stringa); nella seconda, si inseriscono le nuove pubblicità associate ad essa, stilandone un elenco delle pubblicità inserite (una stringa). Quando entrambe le sottoattività sono concluse, viene stampato un rapporto finale con entrambi gli elenchi proposti.

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi eventuali vincoli non esprimibili in UML); diagramma stati e transizioni per la classe Idea; diagramma delle attività di rilascio; specifica del diagramma stati e transizioni, riportando solo gli stati, e le variabili di stato ausiliarie, ma non la specifica delle transizioni; la segnatura delle attività complesse, delle attività atomiche e dei segnali di input/output. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi (nella tabella, inserire anche il motivo di ognuna delle responsabilità).

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

- La classe Idea con la classe Idea Fired, le eventuali sottoclassi e le classi JAVA per rappresentare le associazioni di cui la classe Idea ha responsabilità.
- L'attività di rilascio, ma non le sue sotto-attività, e l'attività atomica di verifica.



### tabella responsabilità

ASSOCIAZIONE	CLASSE	ha RESP.
eostituta	Campagna P	—
	Pubblicità	n
lavora	Creativo	o
	Campagna P	R
responsabile	Creativo	—
	Campagna P	n
crea	Creativo	n o
	Idea	n

TIPO UNL	REALIZZ JAVA
stringa	String
Insieme	HashSet

Criteria

n - molteplicità

o - operazioni

a - regole

associata

Idea

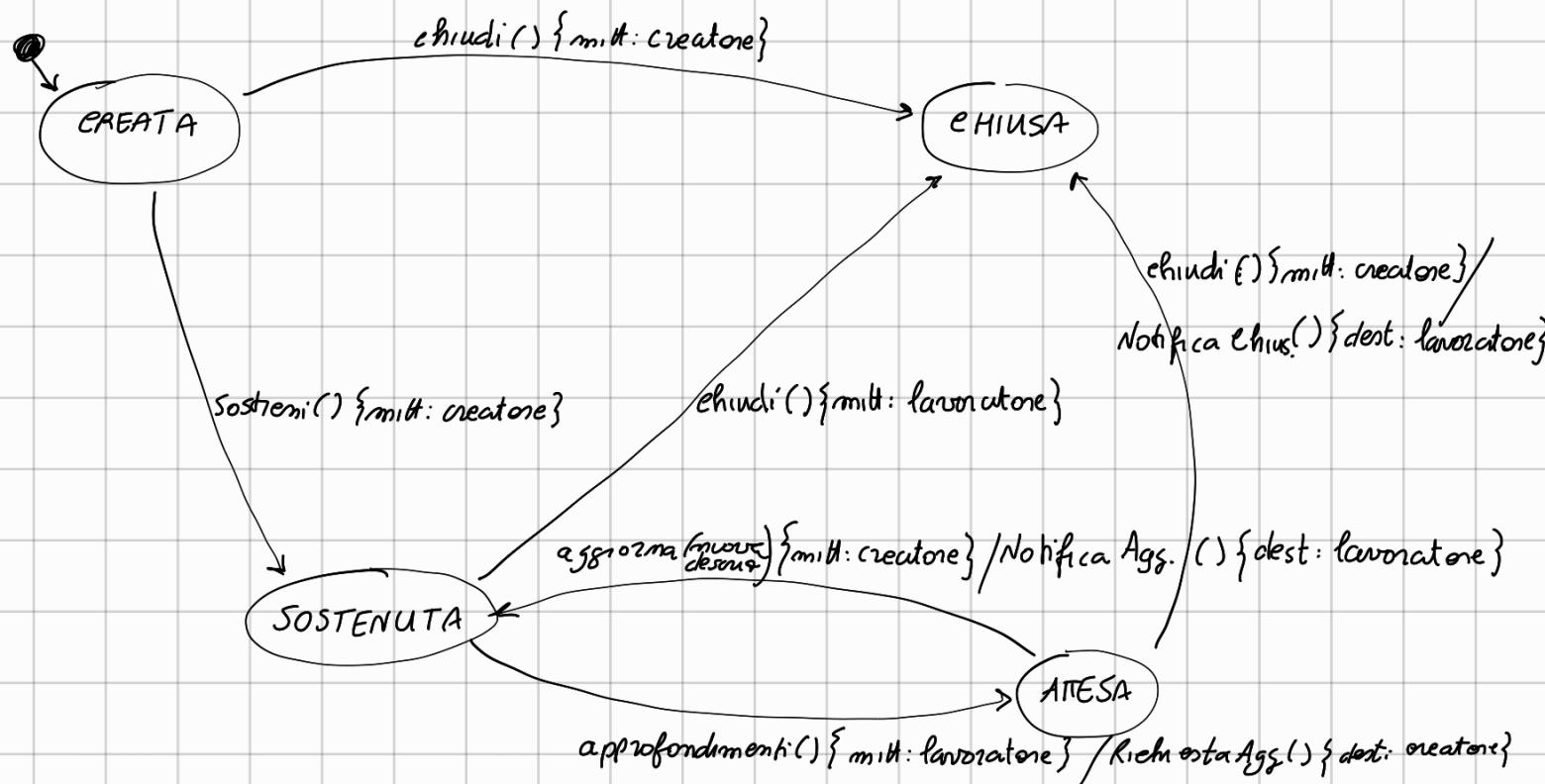
n

Campagna P

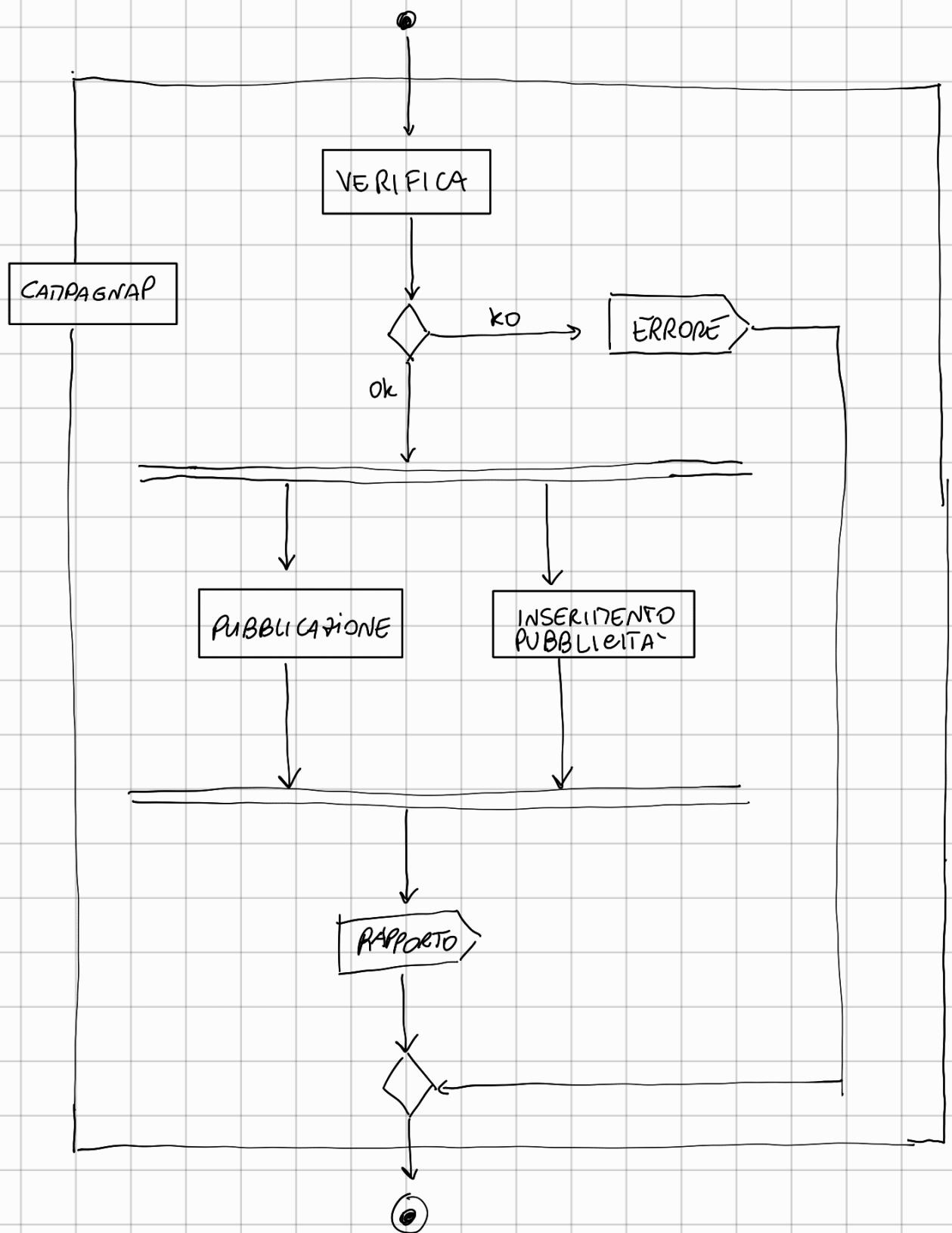
O

Se un lavoratore della campagna ha accesso ad un'idea (per poter richiedere assegnamenti su di essa), allora il creativo ha responsabilità su lavoro e compagnia ha responsabilità in associata.

Inoltre Campagna P ha responsabilità su associata anche per l'operazione di verifica del diag. delle Attività.



Var aux: lavoratore, Creativo



SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corso di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
Esame del **18 settembre 2020**  
*Tempo a disposizione: 3:30 ore*

**Requisiti.** L'applicazione da progettare riguarda un sistema di prenotazione di auto private.

Sono di interesse gli **autisti** che hanno un nome ed una matricola (una stringa). Ogni autista possiede diverse auto (almeno una) e le auto sono possedute da un unico autista. **Alcuni autisti** possono guidare limousine; di questi interessa sapere quale è la lunghezza massima delle limousine che possono guidare. Delle **auto** interessa la targa (una stringa). Sono poi di interesse i **clienti** con nome e numero di telefono. I clienti effettuano prenotazioni e ogni prenotazione è effettuata da un solo cliente. Le **prenotazioni** hanno un codice (una stringa) ed una data, e coinvolgono un cliente (che ha effettuato la prenotazione), un autista ed una macchina (che deve ovviamente essere in possesso dell'autista stesso).

Siamo interessati al comportamento dell'autista. L'autista è inizialmente a **riposo**. Quando riceve una **richiesta** da parte del cliente (con la **data**), se non ha altre prenotazioni per la stessa data, mostra le macchine che ha disponibili e si mette in **attesa** della scelta del cliente. Se invece ha già una prenotazione per la data indicata risponde che è occupato rimanendo a riposo. Quando l'autista è in attesa ed arriva il messaggio da parte del **cliente** che aveva richiesto la prenotazione con la macchina scelta, crea una prenotazione e la memorizza nel sistema (vedi diagramma UML) e torna a riposo; se il messaggio per errore arriva da un altro cliente, l'autista risponde segnalando l'errore senza cambiare stato.

Siamo interessati alla seguente attività principale. L'attività prende un insieme di prenotazioni e verifica se ci sono prenotazioni il cui autista non possiede l'auto indicata dalla prenotazione e in caso segnala l'errore terminando. Se la verifica non da errore, allora procede eseguendo in parallelo due sottoattività complesse A e B i cui dettagli non interessano se non per il fatto che entrambe le sottoattività A e B producono un risultato una stringa. Al termine di entrambe A e B, i due risultati prodotti vengono integriti (producendo una nuova stringa) e stampati in output.

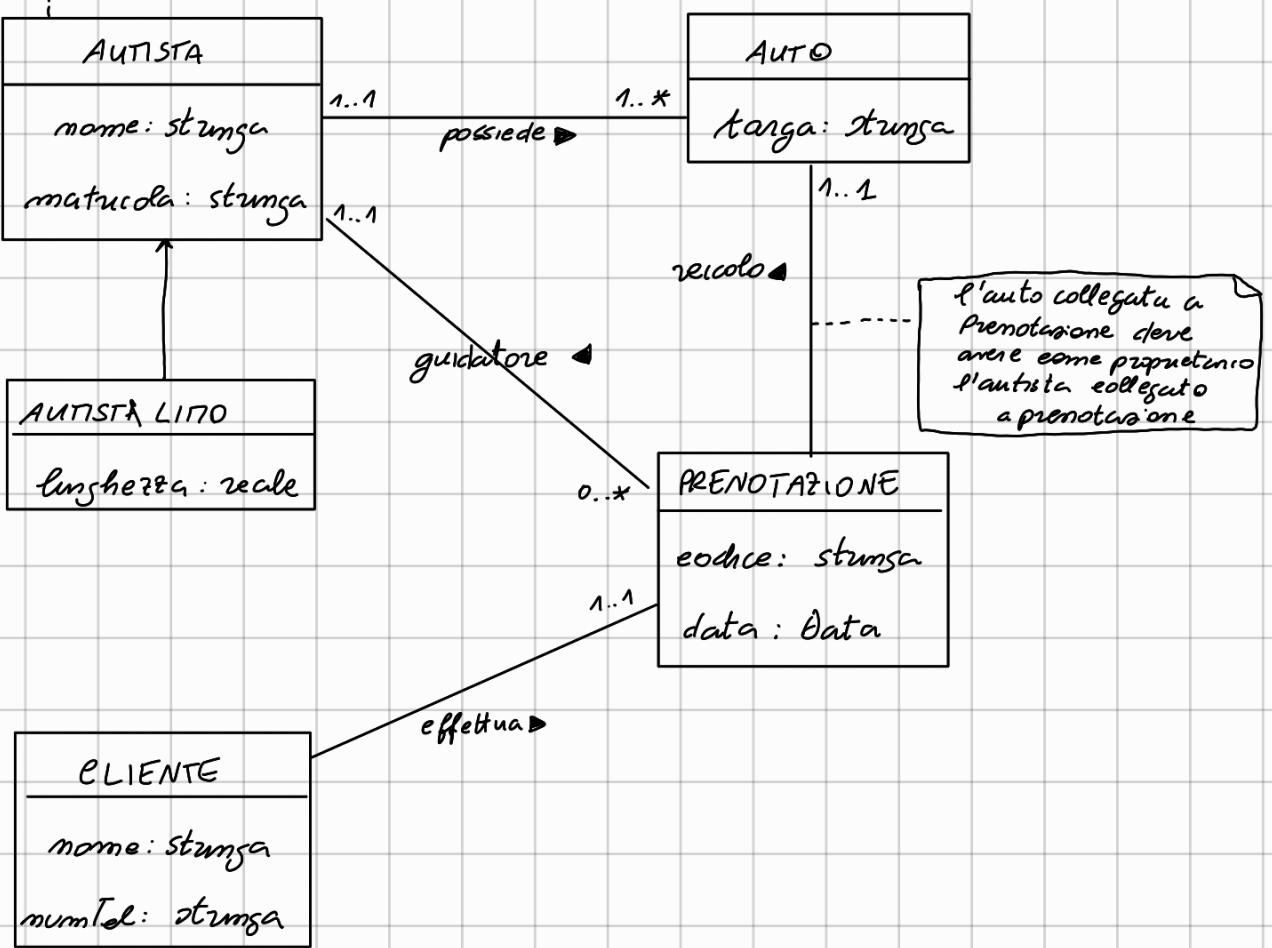
**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi eventuali vincoli non esprimibili in UML); diagramma stati e transizioni per la classe **Autista**; diagramma delle attività principale; specifica del diagramma stati e transizioni, riportando solo gli stati, e le variabili di stato ausiliarie, ma non la specifica delle transizioni; la **segnatura** delle attività complesse, delle attività atomiche e dei segnali di input/output. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le **responsabilità** sulle associazioni del diagramma delle classi (nella tabella, inserire anche il motivo di ognuna delle responsabilità).

**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

- La classe **Autista** con la classe **AutistaFired**, le eventuali sottoclassi e le classi JAVA per rappresentare le *associazioni* di cui la classe **Autista** ha responsabilità (basta un riportare un solo “**TipoLink**” e “**ManagerAssociazione**”).
- L'**attività principale**, l'**attività atomica di verifica**, ma non le sotto-attività A e B e i segnali di input-output.

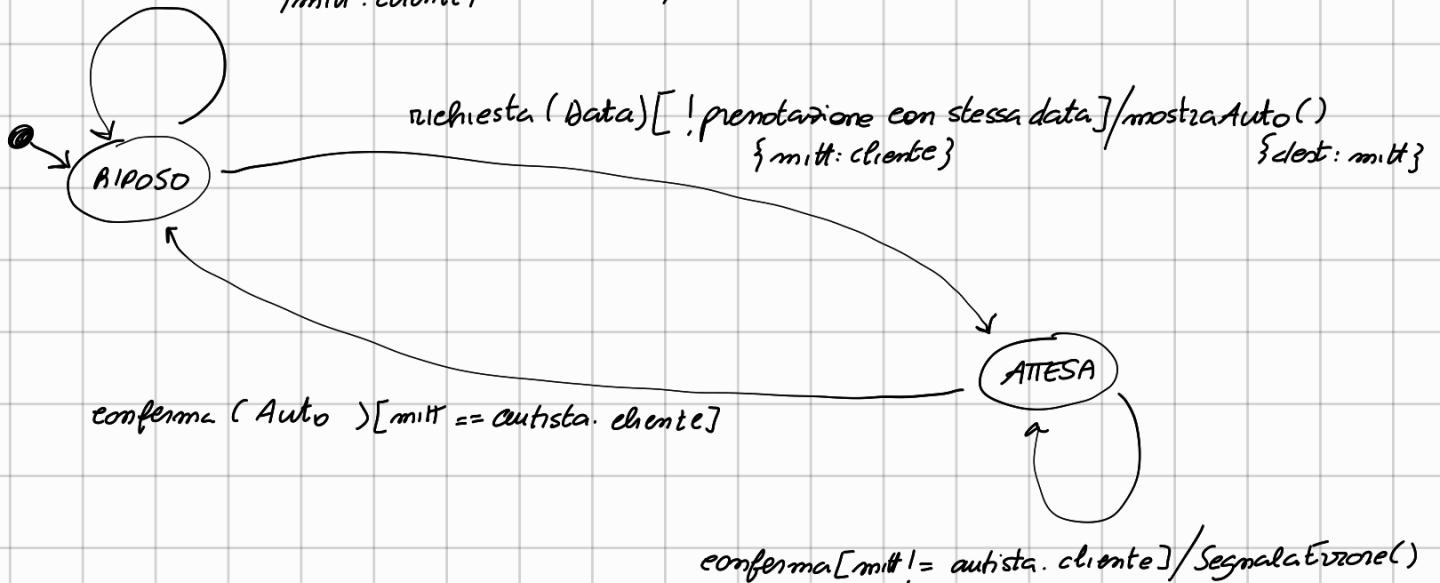
vedi diagramma  
Stati e Transizioni



ASSOCIAZIONE	CLASSI	ha RESP.
possiede	Autista	1
	Auto	1
effettua	Cliente	1
	Prenotazione	—
guidatore	Prenotazione	1
	Autista	0
veicolo	Prenotazione	1 0
	Auto	—

TIPO VNL	REALIZZAZIONE JAVA
string	String
real	double
insieme	HashSet
Date	int, int, int

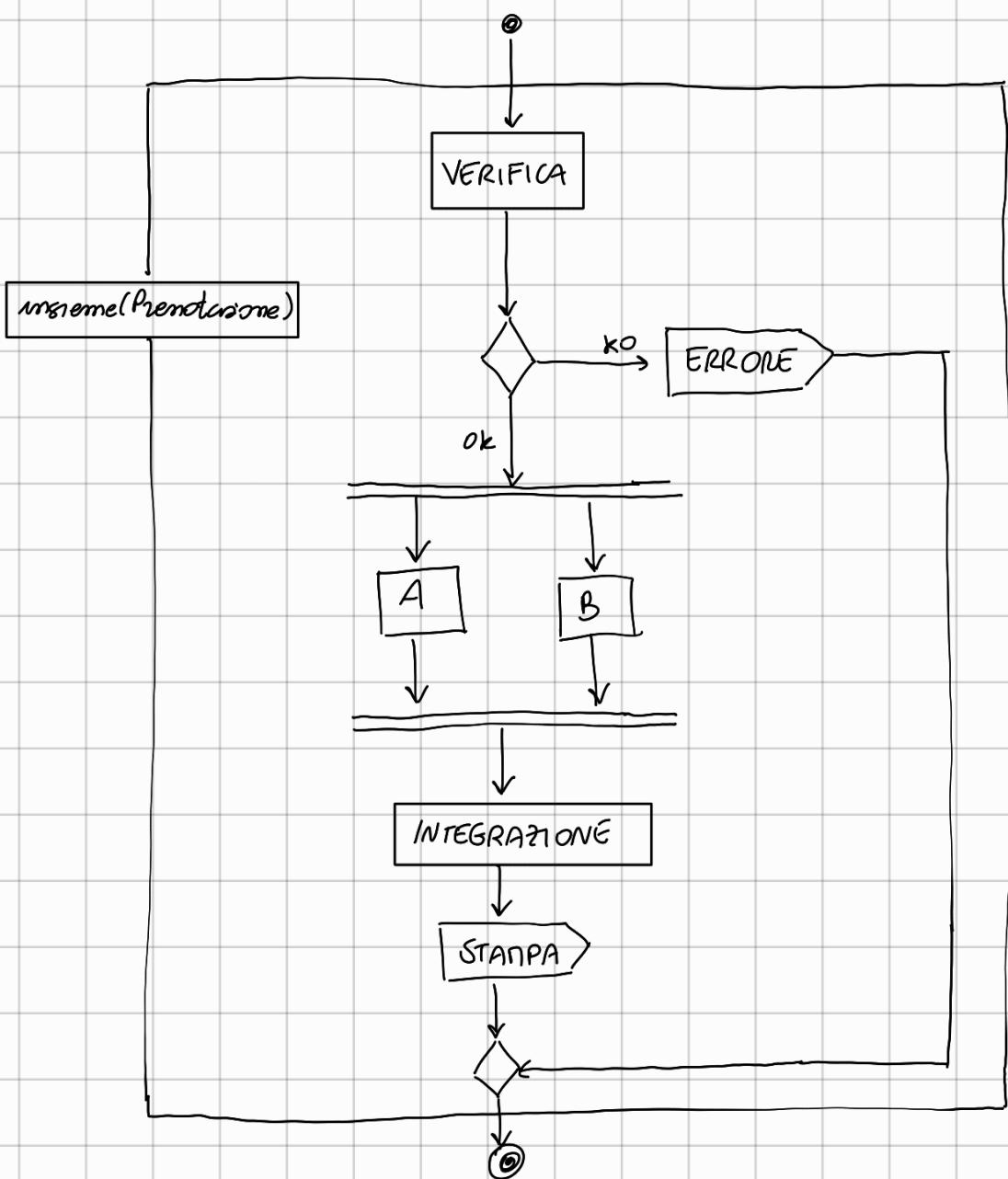
richiesta(Data) [prenotazione con stessa data] / occupato() {dest: cliente}  
 {mitt: cliente}



var Aux

data: Data

cliente: Cliente



SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corsi di Laurea in Ingegneria Informatica ed Automatica  
**Corso di Progettazione del Software**  
**Esame del 19 febbraio 2021**  
*Tempo a disposizione: 3 ore*

**Requisiti.** L'applicazione riguarda un videogioco sul mondo del crimine. Una gilda ha un nome ed è composta da un certo numero di assassini, minimo 3. Un assassino è una persona (vedi dopo) e appartiene a esattamente una gilda. A un assassino può essere commissionato un tentativo di omicidio con un codice (una stringa), una persona che è la vittima ed un'altra persona che è il committente. Un omicidio ha due campi booleani per memorizzare rispettivamente se concluso o meno e in caso sia concluso se è riuscito o fallito. Se è concluso e riuscito allora è di interesse memorizzare il tempo impiegato per eseguirlo, in secondi. Tutte le persone hanno nome, cognome e indirizzo, e tra queste gli assassini hanno anche un'arma preferita (una stringa), e gli assassini esperti hanno anche una seconda arma.

Siamo interessati al comportamento dell'assassino. Un assassino è inizialmente a *riposo*. Se riceve dalla sua gilda un messaggio *uccidi* con payload un omicidio commissionato a se stesso ed il tempo corrente, notifica la gilda con un messaggio e si mette nello stato *missione*. Quando riceve l'evento *fine* inviato dalla propria gilda con payload il tempo corrente e l'esito (omicidio fallito o riuscito) aggiorna le informazioni calcolando, se riuscito, il tempo impiegato nell'omicidio e torna a *riposo*. Per calcolare il tempo impiegato eseguire la differenza tra il tempo corrente nel evento *uccidi* e il tempo corrente nell'evento *fine*.

Siamo interessati alla seguente attività principale che prende come parametro un insieme di omicidi. Questa attività prima verifica che nessun committente di un omicidio nell'insieme sia anche la vittima di un omicidio dell'insieme. Se questa verifica fallisce manda un segnale di errore e termina. Altrimenti, esegue concorrentemente due sottoattività: nella prima vengono tentati tutti gli omicidi, nella seconda vengono richiesti i pagamenti ai committenti. I dettagli di queste due sottoattività non interessano, salvo il fatto che la prima attività produce il sottoinsieme degli omicidi riusciti e la seconda il totale dei soldi raccolti. Quando entrambe le sottoattività si concludono, si esegue un'attività che calcola la differenza fra i soldi raccolti e quelli previsti come pagamento degli omicidi riusciti e la si stampa.

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare la fase di analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo del diagramma delle classi (inclusi vincoli non esprimibili in UML), diagramma stati e transizioni per la classe *Assassino*, diagramma delle attività, specifica del diagramma stati e transizioni, riportando solo gli stati, e le variabili di stato ausiliarie, ma non la specifica delle transizioni; la segnatura delle attività complesse, delle attività atomiche e dei segnali di input/output. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

**Domanda 2.** Effettuare la fase di progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte effettuate. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi (nella tabella, inserire anche il motivo di ognuna delle responsabilità).

**Domanda 3.** Effettuare la fase di realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte effettuate. È obbligatorio realizzare in JAVA solo i seguenti aspetti dello schema concettuale:

- La classe *Assassino* (con classe *AssassinoFired*), eventuali *superclassi* e *sottoclassi* e le classi per rappresentare le *associazioni* di cui la classe *Assassino* è responsabile.
- L'*attività principale*, e la sua *attività atomica di verifica*.

Diagramma delle classi:

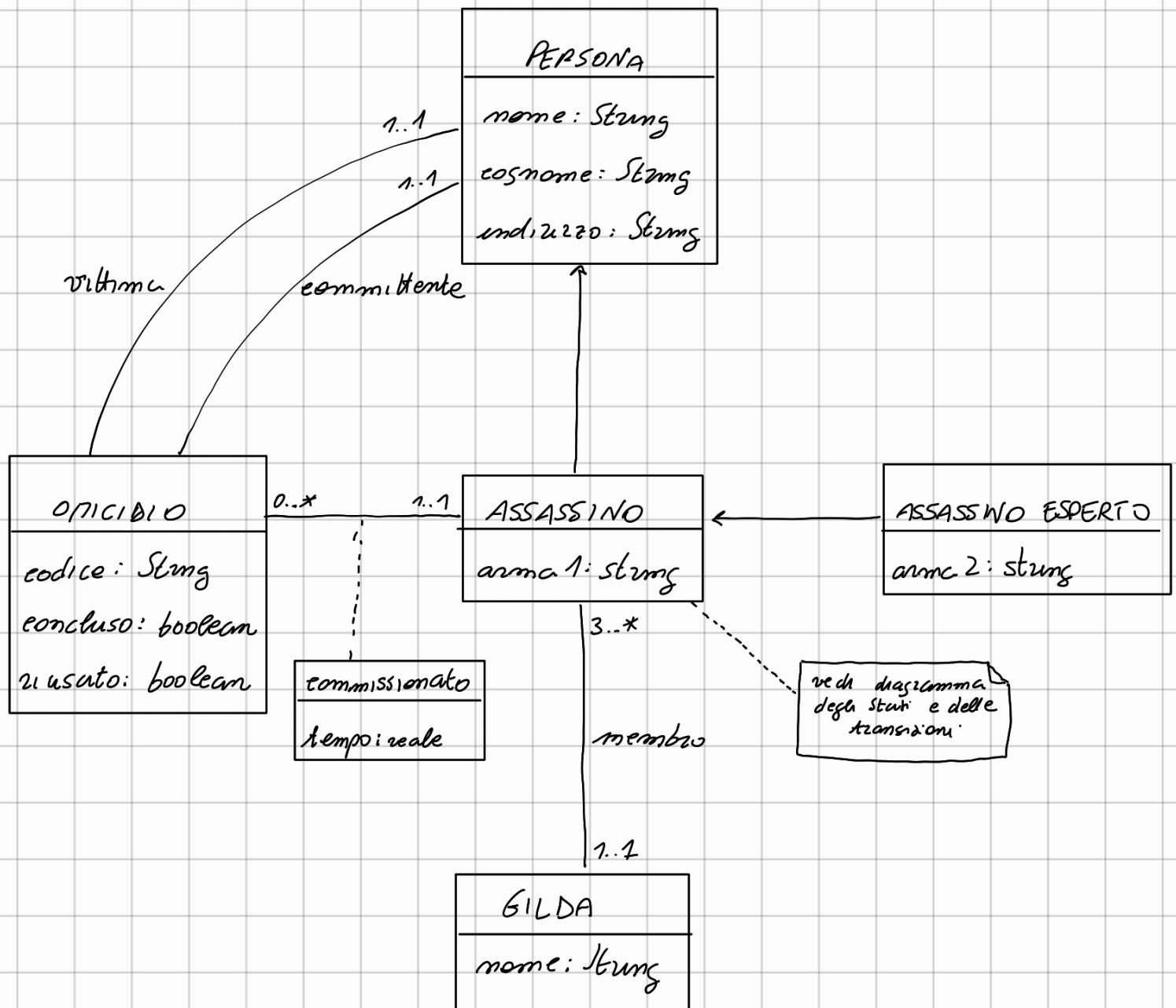
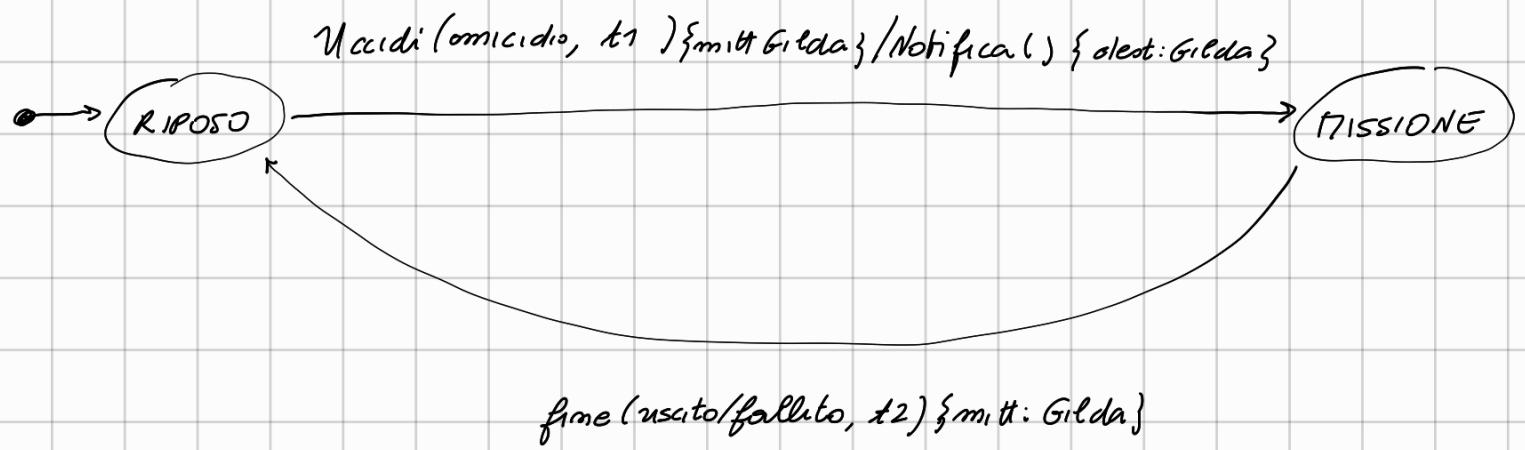


Tabella responsabilità:

ASSOCIAZIONE	CLASSI	RESPONSABILITÀ
membro	Gilda	□ ○
	Assassino	□ ○
commissionato	Omicidio	□
	Assassino	○
vittima	Omicidio	□
	Persona	—
committente	Omicidio	□
	Persona	—

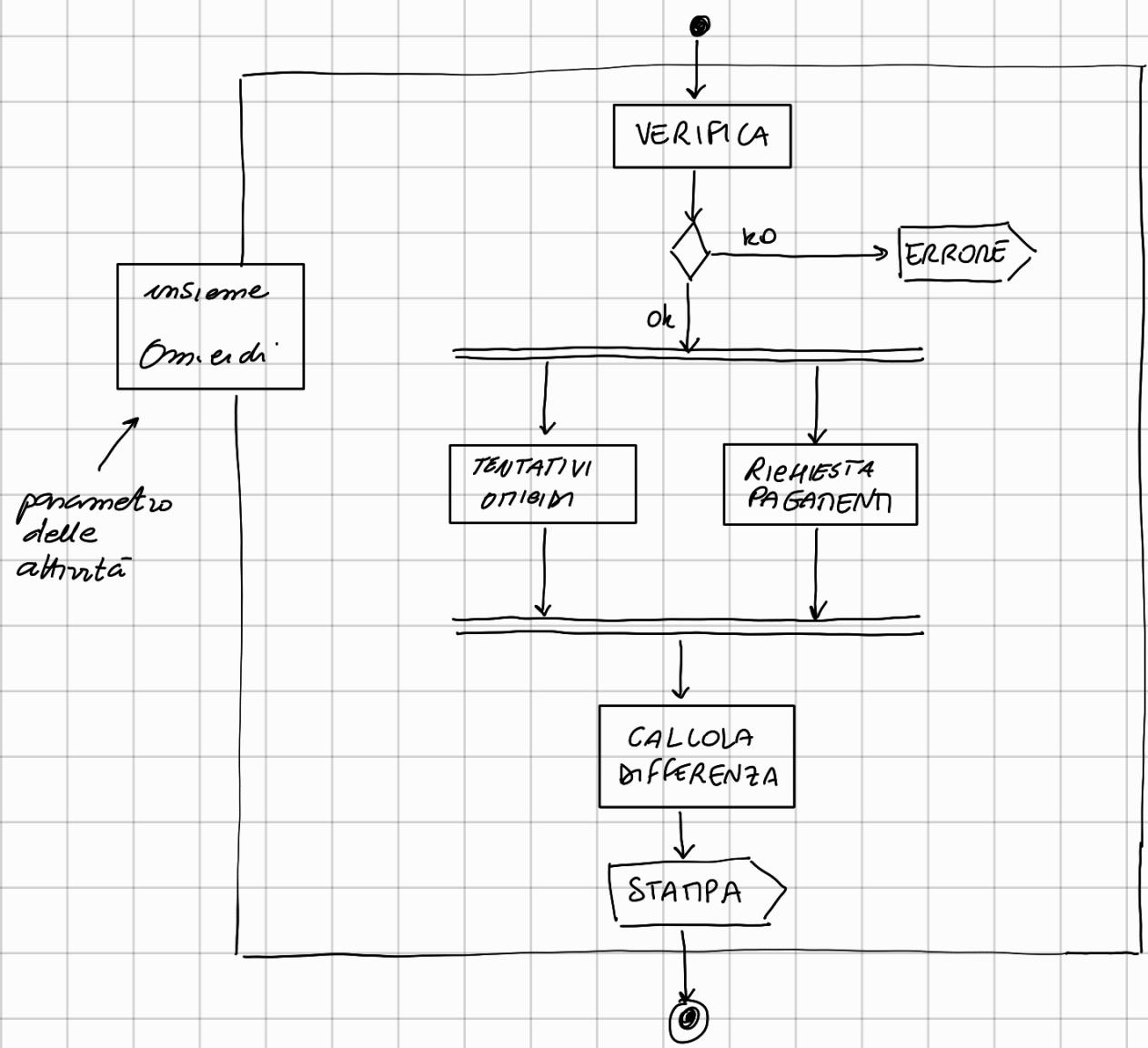
Diagrammi degli stati e delle transizioni  
Assassino



Var Aux: double t1

Omicidio o

Diagramma delle Attività



SAPIENZA Università di Roma  
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Corsi di Laurea in Ingegneria Informatica ed Automatica  
Corso di Progettazione del Software  
Esame del 28 gennaio 2022  
Tempo a disposizione: 3 ore

**Requisiti.** L'applicazione da progettare riguarda la gestione di gruppi musicali da parte di una casa discografica. Di ogni gruppo musicale interessa il nome (una stringa) e l'insieme ordinato delle formazioni (almeno una) che si sono succedute negli anni. Di ogni formazione, che è costituita da almeno un musicista ed è relativa ad uno ed un solo gruppo, sono di interesse l'anno di creazione e l'anno di scioglimento (due interi, dove il valore -1 rappresenta l'assenza di anno di scioglimento). Non è possibile che due formazioni di uno stesso gruppo musicale si sovrappongano ~~temporalmente~~. Di ogni musicista interessa il nome (una stringa) e la data di nascita e deve fare parte di almeno una formazione. Un musicista suona un insieme di strumenti musicale (anche vuoto) ed è di interesse sapere a partire ~~dalla quale anno~~ lo suona. Di ogni strumento musicale sono di interesse il nome (una stringa) e l'anno di introduzione. Tra gli strumenti sono particolarmente di interesse quelli a corda e quelli a fiato (sebbene ne esistano anche altri tipi). Degli strumenti a corda interessa il numero di corde (un intero), mentre di quelli a fiato interessa il numero di ottave coperte (un intero).

Siamo interessati al comportamento dei musicisti. Inizialmente un musicista è in attività. Ad un certo punto può sentire l'impulso (un evento) di andare in pausa di riflessione con una specifica causa (una stringa). Di questo evento (e della causa) sono notificati tutti gli altri membri delle formazioni in cui suona attualmente (non quelle sciolte). Ad un certo punto il musicista ha l'impulso di tornare in attività e di questo sono notificati tutti gli altri membri delle formazioni in cui suona attualmente (non quelle sciolte). Il musicista può ad un certo punto morire. Di questo sono informati tutti gli altri membri delle formazioni in cui suona attualmente. In stato di deceduto il musicista non riceve più eventi.

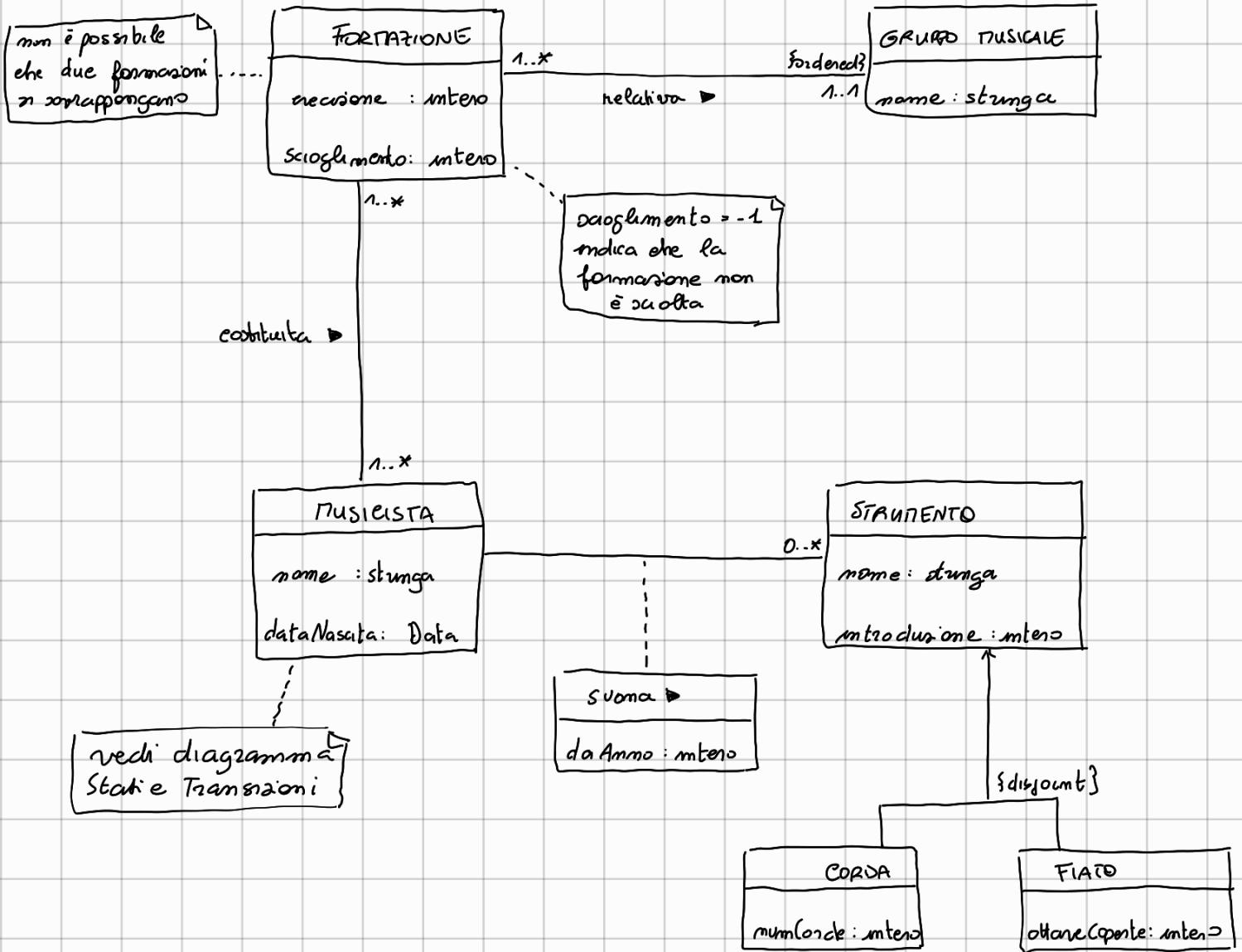
Siamo interessati alla seguente attività: la registrazione di un disco. L'attività prende in ingresso uno specifico gruppo musicale. Come primo passo, l'attività deve verificare che tutti i musicisti che compongono l'ultima formazione attiva siano in stato disponibile. Se ciò non fosse, occorre procedere alla sostituzione temporanea dei musicisti mancanti oppure alla creazione di una nuova formazione (un'unica sotto-attività complessa di cui non interessano i dettagli). Una volta disponibile una formazione completa inizia il processo di produzione del disco che consta di due sotto-attività (complesse ma di cui non si discutono qui i dettagli) e cioè la scrittura dei brani e la loro registrazione. Una volta che entrambe le sotto-attività si sono completate viene rilasciato al sistema un annuncio stampa (una stringa).

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare la fase di analisi producendo lo schema concettuale in Uml per l'applicazione, comprensivo del diagramma delle classi (inclusi vincoli non esprimibili in UML), diagramma stati e transizioni per la classe Musicista, diagramma delle attività, specifica del diagramma stati e transizioni, riportando solo gli stati, e le variabili di stato ausiliarie, ma non la specifica delle transizioni; la segnatura delle attività complesse, delle attività atomiche e dei segnali di input/output. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

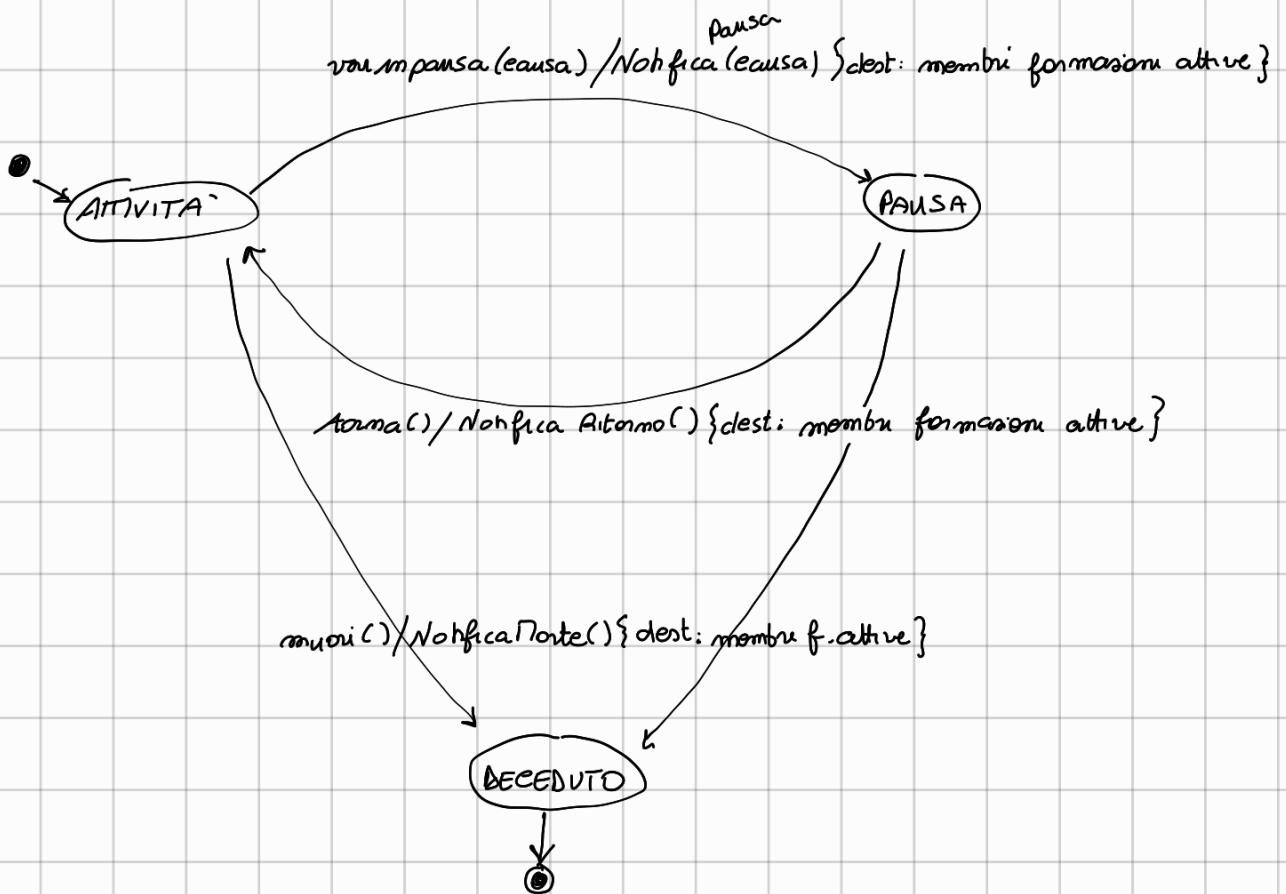
**Domanda 2.** Effettuare la fase di progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte effettuate. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi (nella tabella, inserire anche il motivo di ognuna delle responsabilità).

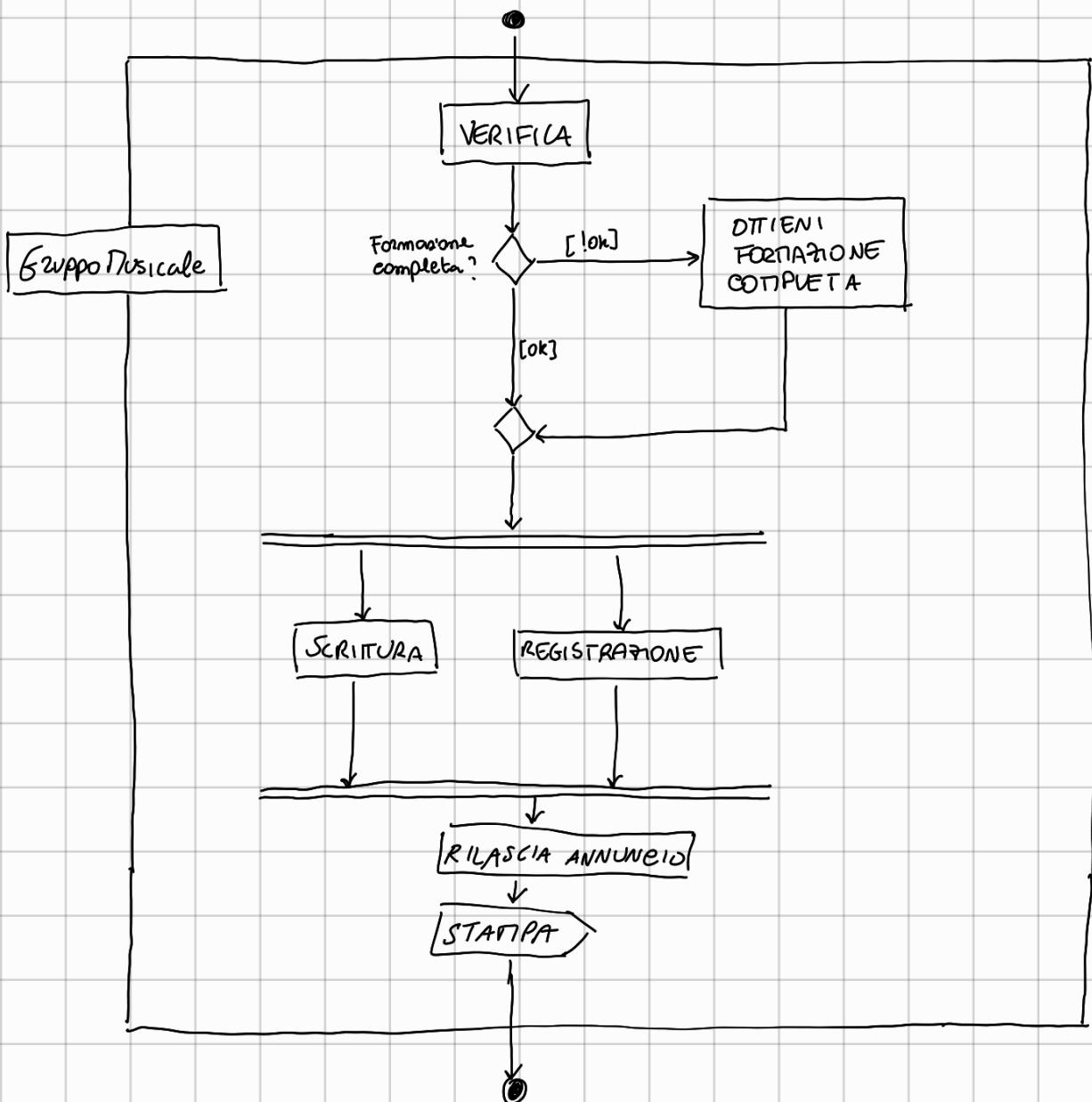
**Domanda 3.** Effettuare la fase di realizzazione, producendo un programma Java e motivando, qualora ce ne fosse bisogno, le scelte effettuate. E obbligatorio realizzare in Java solo i seguenti aspetti dello schema concettuale:

- La classe Musicista (con classe MusicistaFired), eventuali sottoclassi e le classi per rappresentare le associazioni di cui la classe Musicista è responsabile.
- L'attività principale, e la sua attività atomica di verifica.



ASSOCIAZIONE	CLASSI	ha RESP	TIPO UML	REALIZZATI IN JAVA
reflexiva	Formazione	Si <sup>n</sup>	stringa	String
	Gruppo	Si <sup>n, R, O</sup>	intero	int
costituita	Formazione	Si <sup>n, O</sup>	booleano	boolean
	Musicista	Si <sup>n, O</sup>	Lista	LinkedList
Suona	Musicista	Si <sup>R</sup>	Insieme	HashSet
	Strumento	No	Data	int, int, int





**Requisiti.** L'applicazione da progettare riguarda la gestione dei processi di produzione all'interno di uno stabilimento industriale. Sono di interesse i processi produttivi, dei quali interessa conoscere il codice ed il nome. Ogni processo produttivo richiede l'utilizzo in sequenza di almeno due macchinari industriali. Di ogni macchinario industriale interessano il numero seriale, la marca, il modello, la data di acquisto e la data di ultima manutenzione (se mai avvenuta). Un macchinario industriale può partecipare ad un numero qualunque di processi produttivi che sono di interesse per l'applicativo. I macchinari industriali si dividono in esattamente due categorie disgiunte: quelli completamente automatizzati e quelli che richiedono intervento umano. Del personale impiegato interessano il nome, il cognome e la data di nascita. Esistono diverse categorie di impiegati, ma in particolar modo per l'applicazione sono di interesse i supervisori alla produzione e gli operatori. Ogni operatore è assegnato al massimo ad un macchinario ad intervento umano. A sua volta un macchinario ad intervento umano richiede almeno un operatore per funzionare. Ogni processo produttivo è supervisionato da uno ed un solo supervisore. A sua volta un supervisore può supervisionare un numero qualsiasi (anche zero) di processi produttivi.

Siamo interessati al comportamento dei macchinari. Un macchinario è inizialmente a riposo. Se riceve il comando di avvio per un certo (processo) produttivo passa allo stato in funzione e di questo sono avvisati (nel caso di macchinario ad intervento umano) gli operatori coinvolti. Alla fine di una operazione, nel qual caso riceve un evento di completamento, ritorna nello stato di riposo. Se qualcosa si rompe mentre è in funzione può ricevere un evento di rottura ed in questo caso notifica il supervisore del processo produttivo per il quale era attualmente in funzione e passa nello stato rotto. Un evento di manutenzione riporta il macchinario nello stato di riposo aggiornando la data di ultima manutenzione (si assuma che la data odierna sia nota come il risultato di un metodo statico della classe Macchinario).

Siamo interessati alla seguente attività principale. L'attività prende in input un (supervisore di processo S) e gli chiede quale processo produttivo eseguire. A questo punto l'attività deve utilizzare nella sequenza prestabilita dal processo produttivo i macchinari richiesta. Per ogni macchinario utilizzato, vengono avviate due attività complesse concorrenti: (i) utilizzo del macchinario e (ii) raccolta delle statistiche. L'attività di raccolta di statistiche consta di una sola operazione atomica (di cui non interessano i dettagli) che restituisce (una stringa) L'attività complessa di utilizzo del macchinario, verifica prima che il macchinario sia nello stato a riposo. Se il macchinario non è a riposo termina il processo produttivo. Se il macchinario è a riposo lo mette in funzione (i dettagli di questa attività atomica non interessano a parte il fatto che un evento di avvio viene generato). Quando tutti i macchinari sono stati utilizzati viene presentato un report come unione delle stringhe ottenute dalla raccolta delle statistiche. Se invece il processo produttivo è terminato prima che tutti i macchinari siano stati utilizzati (a causa di un macchinario rotto o in funzione), si visualizza un messaggio di errore.

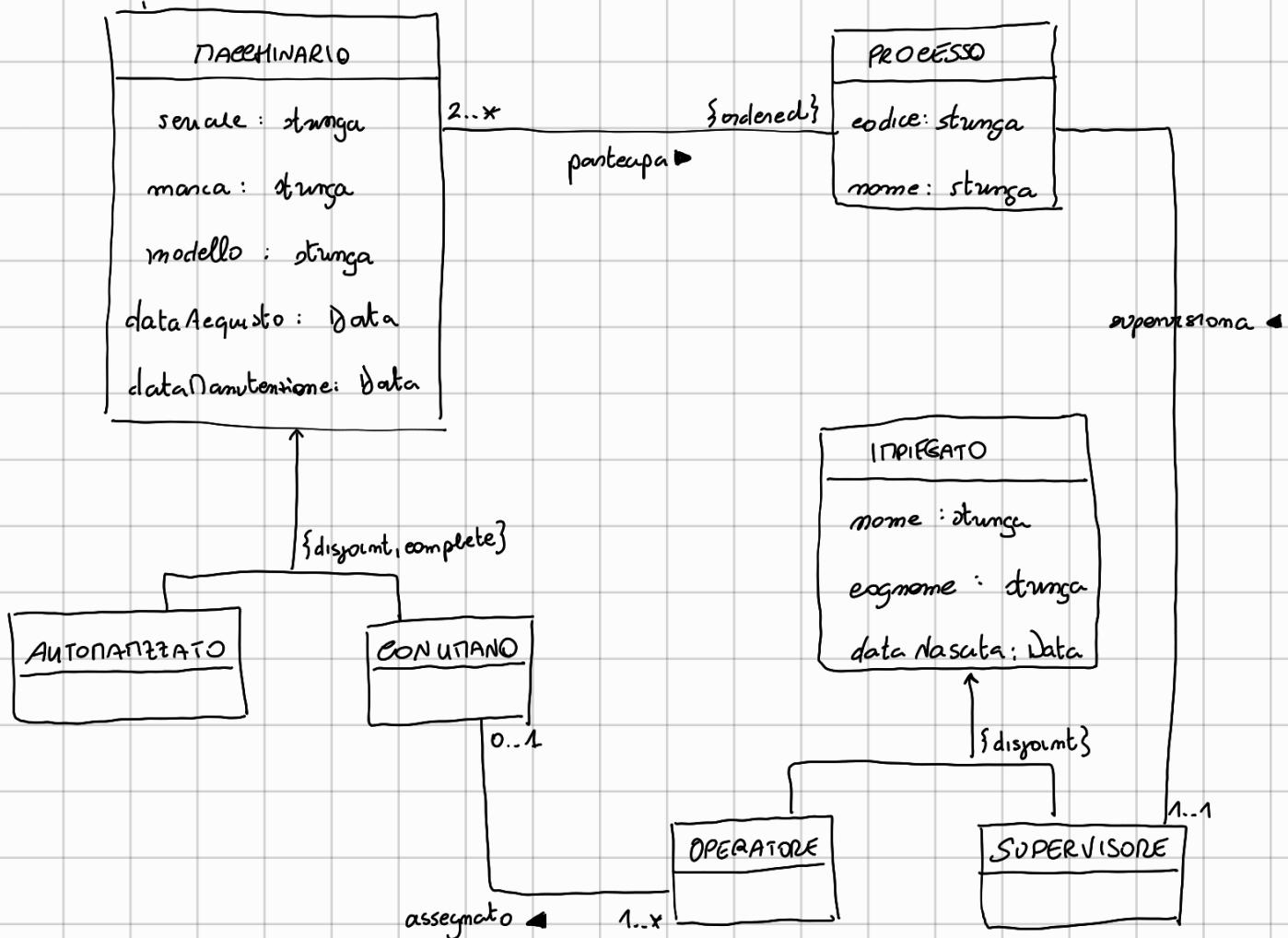
**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi eventuali vincoli non esprimibili in UML); diagramma stati e transizioni per la classe *Macchinario*; diagramma delle attività; specifica del diagramma stati e transizioni; segnatura dell'attività principale, sottoattività non atomiche, atomiche e segnali di input/output. Si noti che NON è richiesta la specifica delle attività. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

**Domanda 2.** Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi.

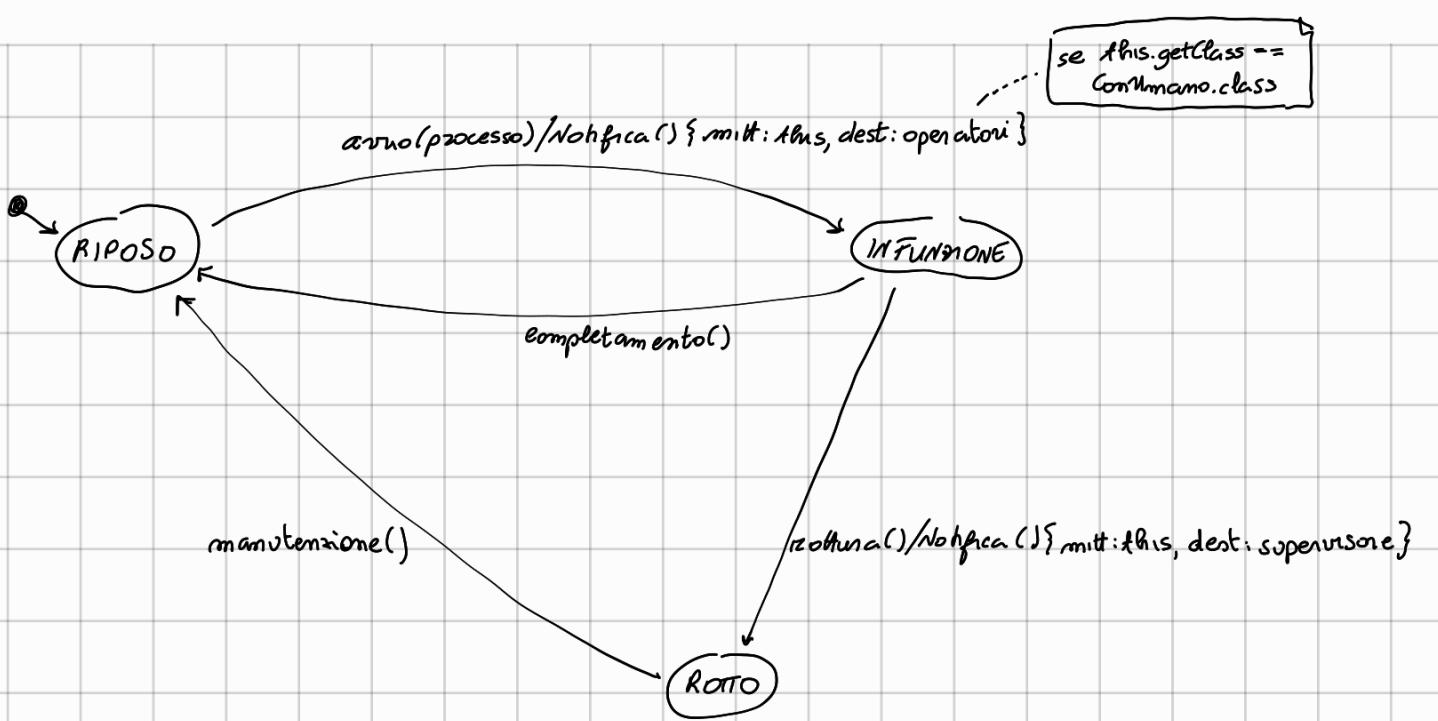
**Domanda 3.** Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare, realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

- La classe *Macchinario* (ed eventuali sottoclassi) con la classe *MacchinarioFired*, le classi Java per rappresentare le associazioni di cui la classe *Macchinario* ed eventuali sottoclassi hanno responsabilità.
- L'attività principale e le sue eventuali sottoattività NON atomiche.

Arch. diag. statiche  
Tramandaioni



ASSOCIAZIONE	CLASSI	ha RESP?
partecipa	Macchinario	Sì (R)
	Processo	Sì (N o R)
assegnato	Operatore	Sì (N)
	Conunitario	Sì (N o R)
supervisiona	Supervisore	Sì (O)
	Processo	Sì (N o )



Var Aux:

$p$  : Processo

