Progettazione del Software

simulazione di esame III

Domenico Fabio Savo

Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti

Requisiti

L'applicazione da progettare riguarda la gestione annuale di squadre di manutenzione per piattaforme petrolifere ed i loro interventi. Di una squadra interessa il codice (una stringa), il suo ingegnere capo (uno ed uno solo) con l'informazione sul suo anno di nomina nella squadra (un intero positivo), ed i tecnici che vi fanno parte. Una squadra è formata da almeno 5 ed al massimo 7 tecnici, ma se la squadra è una squadra di intervento speciale vi devono appartenere esattamente 7 tecnici. Di una squadra di intervento speciale interessa conoscere anche le specializzazioni (una stringa) che possono essere di un numero qualunque. Di un tecnico interessa il nome (una stringa), l'età (un intero positivo) ed il tipo di patente posseduta (una stringa). Ogni tecnico appartiene ad una ed una sola squadra. Di un ingegnere capo interessa conoscere il nome (una stringa), l'età (un intero positivo) ed il suo stipendio (un intero positivo). Un ingegnere capo non può essere anche un tecnico ed dirige una ed una sola squadra. Di un intervento interessa conoscere la data ed il costo (un intero positivo). Ogni intervento è eseguito da due squadre: la squadra primaria e la squadra secondaria. Per motivi amministrativi, date due squadre, A e B, l'intervento in cui A è la squadra primaria e B la secondaria deve essere unico (in altri termini, non può esistere più di un intervento in cui A è la squadra primaria e B è la squadra secondaria).

L'utente dell'applicazione è interessato ad effettuare alcuni controlli. In particolare:

- Data una squadra S ed una data D, restituire l'insieme di squadre di intervento speciale con cui S ha eseguito interventi come squadra primaria prima della data D;
- Dato un tecnico, verificare se la sua età è maggiore dell'età media dei tecnici della squadra a cui appartiene.

Domande

DOMANDA 1: Basandosi sui requisiti riportati sopra, svolgere la fase di analisi producendo lo schema concettuale in UML per l'applicazione e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

DOMANDA 2 : Svolgere la fase di progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

È obbligatorio solo progettare gli algoritmi e definire le responsabilità sulle associazioni (indicando i criteri con cui sono stabilite le responsabilità).

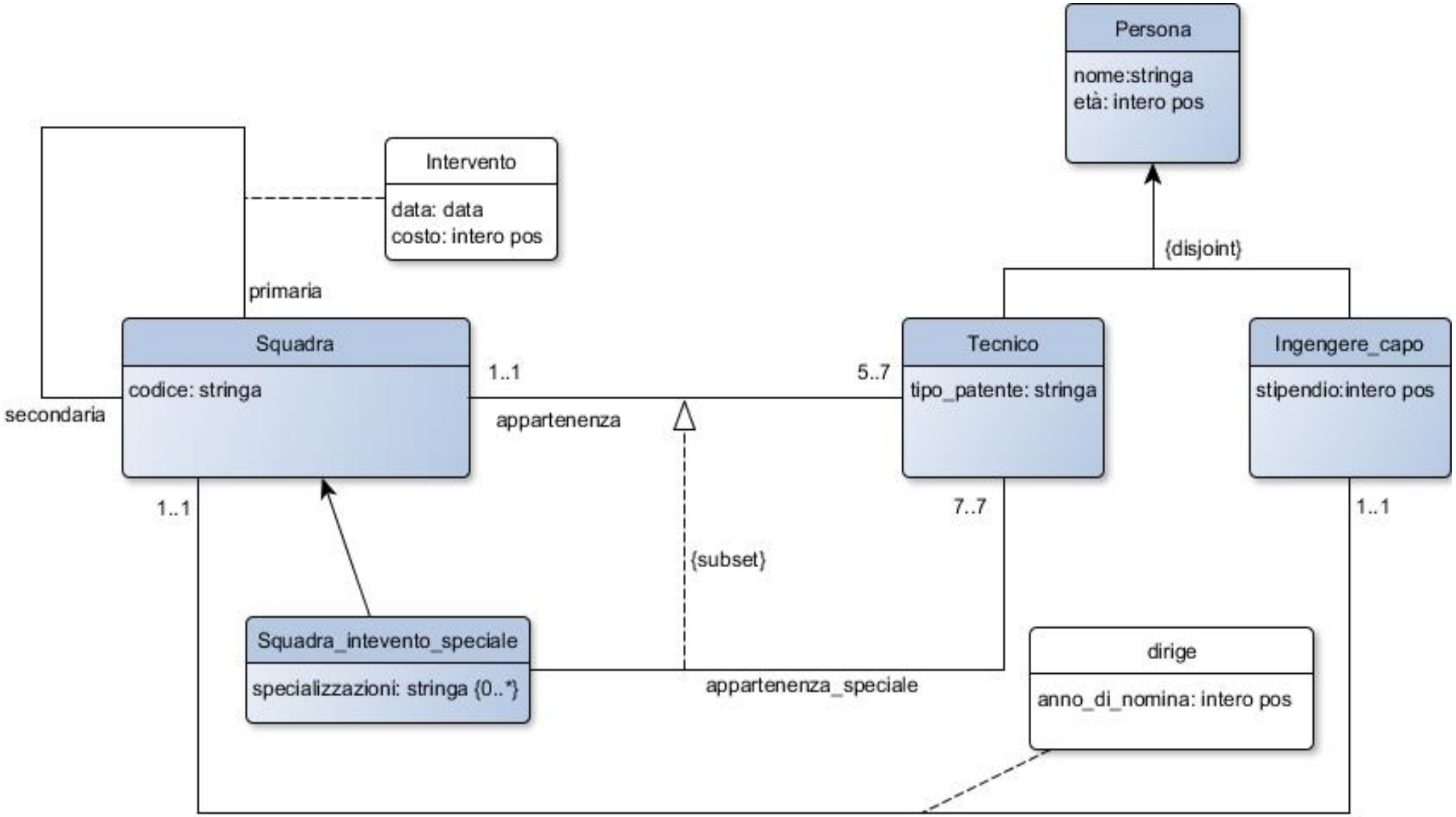
DOMANDA 3: Svolgere la fase di realizzazione, producendo un programma Java e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

È obbligatorio realizzare in Java solo i seguenti aspetti dello schema concettuale:

- Gli aspetti dello schema che sono relativi alle **squadre** ed agli **interventi**, realizzando tutte le classi (con eventuali sottoclassi e/o superclassi) ed associazioni coinvolte in questa parte dello schema;
- La prima operazione dello use case;
- Il codice di classi eccezione eventualmente usate per la realizzazione dei punti precedenti.

Fase di analisi

Diagramma delle classi



Casi d'uso

Specifica del caso d'uso

InizioSpecificaUseCase Controlli

squadreSecondarie(S: Squadra, D: data): Insieme(Squadra_intervento_speciale)

pre: nessuna

post: *result* è l'insieme delle istanze della classe Squadra_intervento_speciale che partecipano come squadra secondaria ad un link L di tipo intervento la cui data è precedente a D e dove L.primaria = S.

tecnicoAnziano(T: tecnico): Boolean

pre: nessuna

post: sia **a** il link di tipo *appartenenza* a cui partecipa **T**. Sia **S** l'insieme dei tecnici **t** per cui esiste un link **a'** di tipo *appartenenza* tale che **a'.Squadra = a.Squadra** e **a'.Tecnico = t**. Sia infine

Media= (somma(età(t)) per ogni **t** in **S**) **/ |S|**

result = età(T) > Media.

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use-case

Per l'operazione squadreSecondarie(S: Squadra, D: data)

```
Sia result un insieme inizialmente vuoto di Squadra_intervento_speciale;
Per ogni link l di tipo intervento in cui S è coinvolto come squadra primaria
   Se l.data < D
        Se l.secondaria è istanza della classe Squadra_intervento_speciale
               result.add(l.secondaria);
return result;
```

Per l'operazione tecniocoAnziano(T: Tecnico)

```
Sia l il link di tipo appartenenza in cui T è coinvolto;
Sia M un reale positivo inizialmente uguale a 0;
Sia N un intero positivo inizialmente uguale a 0;
Per ogni link l' di tipo appartenenza a cui l.Squadra partecipa
        M = M + l'.Tecnico.età;
        N = N + 1;
return T.età > M/N;
```

Responsabilità sulle associazioni

La seguente tabella delle responsabilità si evince da:

1) i requisiti,
2) la specifica degli algoritmi per le operazioni di classe e use-case,
3) i vincoli di molteplicità nel diagramma delle classi.

Associazione	**Classe**	**Ha responsabilità**
Intervento	Primaria (Squadra)	Si (2)
	Secondaria (Squadra)	No
Appartenenza	Tecnico	Si(2,3)
	Squadra	Si (2,3)
Appartenenza_speciale	Tecnico	Si (3)
	Squadra_intervento_speciale	Si (3)
Dirige	Ingegnere_capo	Si (3)
	Squadra	Si (3)

Strutture di dati

Abbiamo la necessità di rappresentare collezioni omogenee di oggetti per rappresentare:

- Responsabilità su associazioni con vincoli di molteplicità 0..*
- Attributi multi-valore
- Strutture dati necessarie per vari algoritmi.

Per fare ciò, utilizzeremo l'interfaccia **Set**, e la classe **HashSet** del *collection framework* di Java.

Corrispondenza fra tipi UML e Java

UML	**Java**
Intero positivo	Int (con gestione opportuna delle precondizioni)
Reale positivo	Float
Data	Supponiamo di avere una classe Java per la gestione delle date come ad esempio java.util.GregorianCalendar
Insieme	HashSet
Stringa	String

Tabelle di gestione delle proprietà di classi UML

Riassumiamo le nostre scelte differenti da quelle di default mediante la tabella delle proprietà immutabili e la tabella delle assunzioni sulla nascita.

Classe UML	**Proprietà immutabile**
Persona	nome
Squadra	codice

	Proprietà	
Classe UML	**Nota alla nascita**	**Non noto alla nascita**
--	**--**	**--**

Si ricorda che di default:

- tutte le proprietà sono mutabili;
- le proprietà singole sono note alla nascita
- le proprietà multiple non sono note alla nascita.

Fase di realizzazione

Fase di realizzazione

Il compito richiede di realizzare solo quanto segue:

- la classe UML *Squadra*
- la classe UML *Squadra_intervento_speciale*
- l'associazione UML *Intervento*
- la prima operazione dello use case;
- la classe Java EccezionePrecondizione.

Procediamo quindi ignorando tutte le altre classi, entità e casi d'uso che non sono fra quelli menzionati sopra.

Struttura dei file e dei package

\---***squadraDiManutenzione***
        | TipoLinkIntervento.java
        | TipoLinkDirige.java
        | Controlli.java
        | EccezionePrecondizioni.java
        | EccezioneMolteplicita.java
        +-----***squadra***
        | Squadra.java
        +-----***squadraInterventoSpeciale***
        | SquadraInterventoSpeciale.java
        +-----***persona***
        | Persona.java
        +-----***tecnico***
        | Tecnico.java
        \-----***ingengereCapo***
                         IngegnereCapo.java

La classe Squadra

```
package squadraDiManutenzione.squadra;
import java.util.*

import squadraDiManutenzione.TipoLinkIntervento;

public class Squadra {
        protected final String codice;
        protected HashSet<TipoLinkIntervento> interventiPrimaSquadra;

        public Squadra(String codice) {
                this.codice = codice;
                interventiPrimaSquadra = new HashSet<TipoLinkIntervento>();
        }
```

```
	public String getCodice(){	return codice;	}

	public void inserisciLinkInternentoSquadraPrimaria(TipoLinkIntervento t) {
		if (t != null && t.getSquadraPrimaria() == this)
			interventiPrimaSquadra.add(t);		}

	public void eliminaLinkInternentoSquadraPrimaria(TipoLinkIntervento t) {
		if (t != null && t.getSquadraPrimaria() == this)
			interventiPrimaSquadra.remove(t);			}

	public Set<TipoLinkIntervento> getLinkHaLavorato() {
		return (Set<TipoLinkIntervento>) interventiPrimaSquadra.clone();
	}
}
```

La classe SquadraInternventoSpeciale

```
package squadraDiManutenzione.squadraInterventoSpeciale;

import java.util.HashSet;
import java.util.Set;
import squadraDiManutenzione.squadra.Squadra;

public class SquadraInternventoSpeciale extends Squadra {

	protected HashSet<String> specializzazioni;

	public SquadraInternventoSpeciale(String codice) {
		super(codice);
		specializzazioni = new HashSet<String>();
	}
```

```
    public void inserisciSpecializzazione(String s) {
        if(s != null)
            specializzazioni.add(s);                    }

    public void eliminaSpecializzazione(String s) {
        specializzazioni.remove(s);                 }

    public Set<String> getSpecializzazioni() {
        return (Set<String>) specializzazioni.clone();      }

}
```

La classe TipoLinkIntervento

```
package squadraDiManutenzione;

import java.util.GregorianCalendar;
import squadraDiManutenzione.squadra.Squadra;

public class TipoLinkIntervento {
        private final Squadra primaria, secondaria;
        private final GregorianCalendar data;
        private final int costo;
```

```
public TipoLinkIntervento(Squadra primaria, Squadra secondaria,  GregorianCalendar data, int costo)
                        throws EccezionePrecondizioni {
        if (primaria == null || secondaria == null) // CONTROLLO PRECONDIZIONI
                        throw new EccezionePrecondizioni("Gli oggetti devono essere inizializzati");
        if (costo < 0) // CONTROLLO PRECONDIZIONI
                        throw new EccezionePrecondizioni("il costo deve essere un intero positivo");
        this.primaria = primaria;
        this.secondaria = secondaria;
        this.data = data;
        this.costo = costo;
}
public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
                        TipoLinkIntervento b = (TipoLinkIntervento) o;
                        return b.primaria == this.primaria && b.secondaria == this.secondaria; }
        else return false;
}
```

```
    public int hashCode() {
            return primaria.hashCode() + secondaria.hashCode(); }

    public Squadra getSquadraPrimaria() { return primaria; }

    public Squadra getSquadraSecondaria() { return secondaria; }

    public int getCostoIntervento() { return costo; }

    public GregorianCalendar getDataIntevento() {return data; }

}
```

La classe EccezionePrecondizioni

```
package squadraDiManutenzione;

public class EccezionePrecondizioni extends Exception {
        private String messaggio;
        public EccezionePrecondizioni(String m) {
                messaggio = m;
        }
        public EccezionePrecondizioni() {
                messaggio = "Si e’ verificata una violazione delle precondizioni";
        }
        public String toString() {
                return messaggio;
        }
}
```

La classe Controlli

```
package squadraDiManutenzione;

import java.util.GregorianCalendar;
import java.util.*;
import squadraDiManutenzione.squadra.Squadra;
import squadraDiManutenzione.squadraInterventoSpeciale.SquadraInternventoSpeciale;

public final class Controlli {

        private Controlli() {           }
```

```
    public static Set<SquadraInternventoSpeciale> squadreSecondarie(Squadra s,
                                                                    GregorianCalendar data){
        HashSet<SquadraInternventoSpeciale> sq =
                                                    new HashSet<SquadraInternventoSpeciale>();
        if(s != null && data != null){
            Iterator<TipoLinkIntervento> i = s.getLinkHaLavorato().iterator();
            while(i.hasNext()){
                TipoLinkIntervento in = i.next();
                 if(in.getDataIntevento().before(data))
                    if(in.getSquadraSecondaria() instanceof SquadraInternventoSpeciale)
                        sq.add((SquadraInternventoSpeciale)in.getSquadraSecondaria());
            }
        }
        return sq;
    }
}
```