Esame 11 Settembre 2020

Es 1

Quesito 1

```
//@requires nums!=null && n>0 &&
//@n<nums.length &&
//@(\forall int i;0<=i && i<= nums.length;
//@ !(\exists int k,0<=k && k<nums.length && k!=i;nums[i]!=nums[k]))

//@ensures \result <==> (\forall int i;0<=i && i<n;
//@(\forall int k;n<= k && k<nums.length; nums[i]<nums[k]))
```

Quesito 2

```
//@requires n>0 && n<nums.length &&
//(@\forall int i;0<=i && i<= nums.length;
//@ !(\exists int k,0<=k && k<nums.length && k!=i;nums[i]!=nums[k]))
//@ensures nums!= null &&
//@ \result<==> (\forall int i;0<=i && i<n;
//@ (\forall int k;n<= k && k<nums.length; nums[i]<nums[k]))
//@signals (NullException e) nums == null
```

Es 2

```java
public static NumeroBinario somma(NumeroBinario n1,NumeroBinario n2){
    int dim=Math.max(n1.dimensione(),n2.dimensione());
    if(n1.dimensione()==n2.dimensione())
        dim++; //bit di overflow

    Iterator<Integer> iter1=n1.destraSinistra();
    Iterator<Integer> iter2=n2.destraSinistra();

    int posizione=1;//Si ipotizza che la posizione parta da 1
    int riporto=0;

    NumeroBinario result=new NumeroBinario(dim);

    while(iter1.hasNext() && iter2.hasNext()){
        int s1=iter1.next();
        int s2=iter2.next();
        riporto=fullAdder(s1,s2,riporto,result);
        pos++;
    }
    //Si è usciti dal while o per iter1 finito, o per iter 2 finito o per entrambi e
    //verrà eseguito solo il while relativo al numero ancora da scorrere
    while(iter1.hasNext()){
        int s1=iter1.next();
        riporto=fullAdder(s1,0,riporto,result);
        pos++;
    }
    while(iter2.hasNext()){
        int s2=iter2.next();
        riporto=fullAdder(s2,0,riporto,result);
        pos++;
    }
    //Caso dell'overflow
    if(dim==pos){
        result.cambiaBit(pos,riporto);
    }
    return result
}

private static int fullAdder(int s1,int s2,int riporto,NumeroBinario result){
    if((s1+s2+riporto)==0)){
        result.cambiaBit(pos,0);
    }
    else if((s1+s2+riporto)==1){
        result.cambiaBit(pos,1);
        riporto=0;
    }
```

```
    else if((s1+s2+riporto)==2){
        riporto=1;
        result.cambiaBit(pos,0);
    }
    else if((s1+s2+riporto)==3){
        riporto=1;
        result.cambiaBit(pos,1);
    }
    return riporto;
}
```

```
public static boolean simmetrico(NumeroBinario nb){
    int dim=nb.dimensione();
    int index=1;
    Iterator<Integer> leftRight=nd.destraSinistra();
    Iterator<Integer> rightLeft=nd.sinistraDestra();

    while(index<dim/2){
        if(leftRight.next()!=rightLeft.next())
            return false;
    }
    return true;
}
```

Es 3

Quesito 1

Poichè FilmMuto ha un metodo in meno rispetto a film, FilmMuto è la **superclasse** e Film la **sottoclasse**

Quesito 2

Per rispondere a questa domanda bisogna osservare le precondizioni, le precondizioni sono uguali, quindi non viola il Require no more, ma, la postcondizione di FilmMuto è più debole, quindi viola il Promise no less, allora FilmMuto è ancora superclasse e Film sottoclasse

Quesito 3

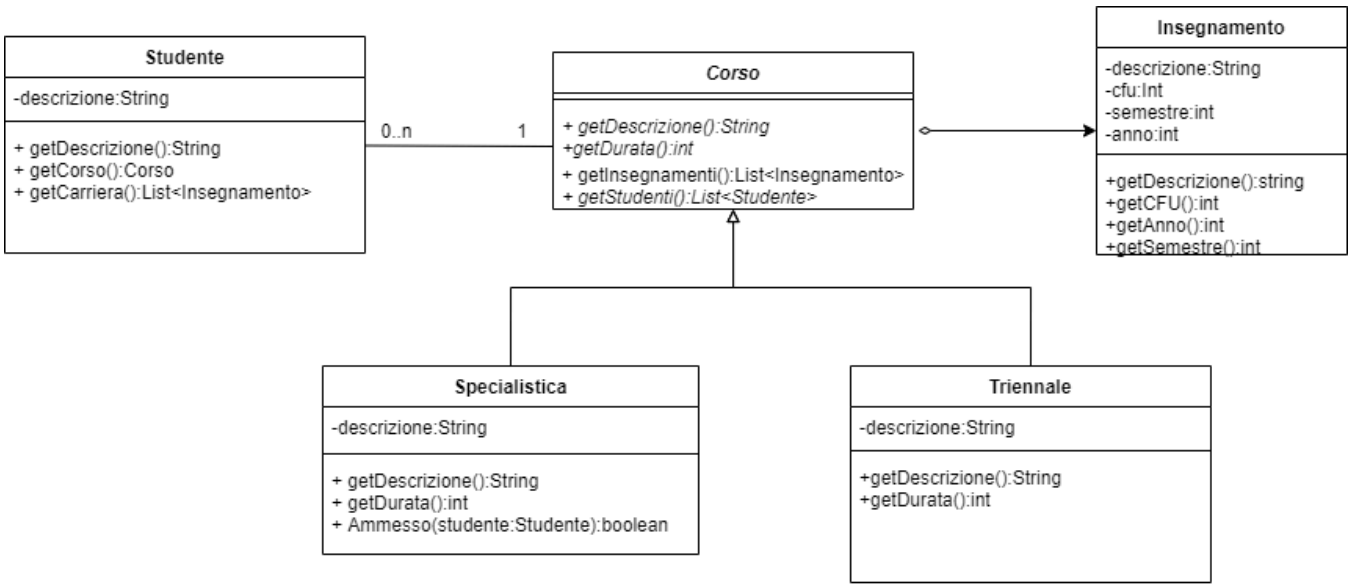
In questo caso, FilmMuto deve essere sottoclasse, poichè la postcondizione di JML sarebbe

calcolata nel seguente modo:
true=>(hasVideo() && hasAudio()) && true=>(hasVideo() && !hasAudio())
Si evince subito che se FilmMuto fosse superclasse la postcondizione risultante sarebbe sempre hasVideo() && !hasAudio()

Es 4

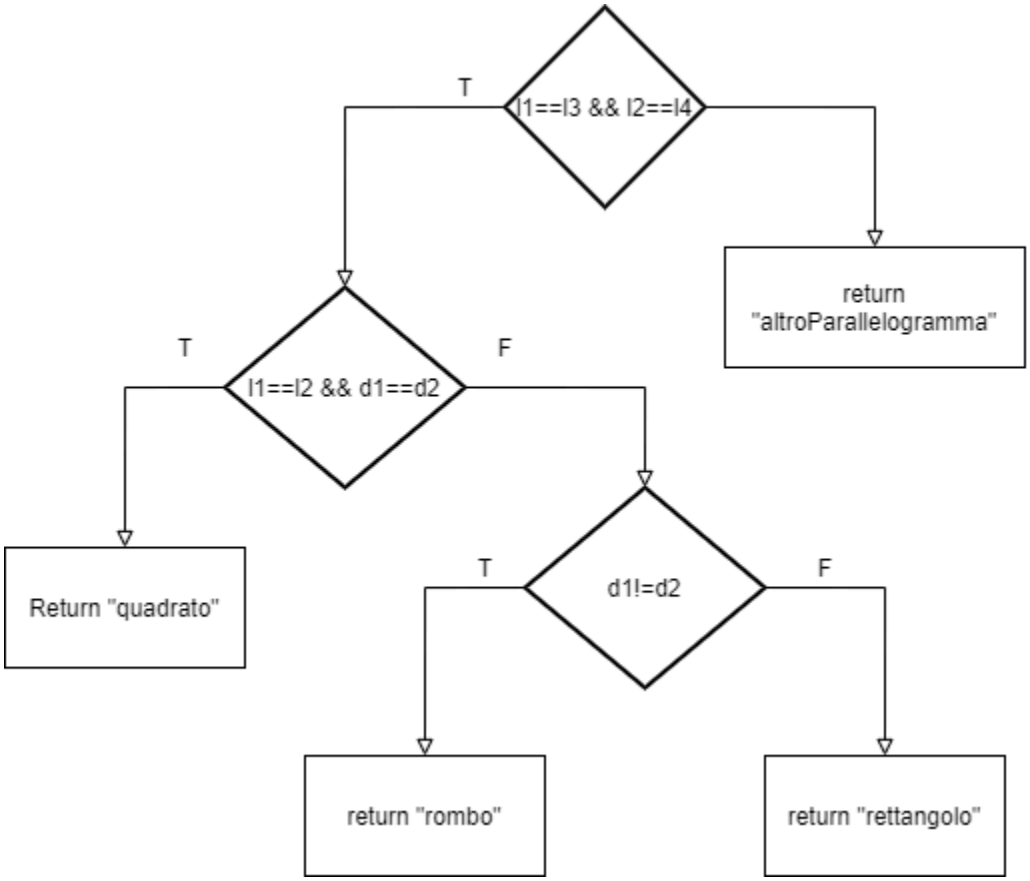
```
public static List<String> startWithA(List<String> list){
    return list.stream().filter(x->x.charAt(0)='a').collect(Collections.toList());
}
```

Es 5



Es 6

Quesito 1



Quesito 2

l1	l2	l3	l4	d1	d2	return atteso
0	1	1	1	1	1	altroParallelogramma
1	1	1	1	1	1	quadrato
1	2	1	2	1	2	rombo
1	2	1	2	1	1	rettangolo

Quesito 3

Per eseguire il metodo delle condizioni, basta aggiungere un test a quelli precedenti, per

esempio:

l1	l2	l3	l4	d1	d2	return attesa
1	1	2	2	1	1	rettangolo

La risposta attesa sarebbe rettangolo, quella ricevuta è però "altroParallelogramma"

Esame 7 luglio 2020

Es1

```
//@assignable \nothing
//@requires a1!=null && n>0
//@ensures (\forall int i;0<=i && i<\result.size();
//@ n==(\num_of int k;0<=k && k<a1.size(); a1.get(k)==\result.get(i)) &&
//@ (\forall int i;0<=i && i<a1.size();
//@   (n==(\num_of int k;0<=k && k<a1.size();
//@     a1.get(k)==a-get(i)))=>(\exists int m;0<=m && \result.size();
//@       \result.get(m)==a1.get(i)) &&
//@ \forall int i;0<=i && i<\result.size();
//@   !(\exists int k;0<=k && k<\result.size && k!=i; \result.get(i)==\result.get(k))
public static ArrayList<Integer> m1(ArrayList<Integer> a1,int n);

//@requires cArr!=null
//@ensures result<==> (\forall int i;0<=i && (cArr.length-1);cArr[i]<cArr[i+1])
public static boolean m2(char[] cArr);

//@requires sArr!=null && s!=null
//@ensures \forall int i;0<=i && i<\old(sArr.length);
//@  (\exists int k;0<=k && k<sArr.length; \old(sArr[i])=sArr[k]) &&
//@!(\exists int i;0<=i && i<\old(sArr.length); \old(sArr[i].charAt(0))==s.charAt(0))=>
//@  (\exists int k;0<=k && k<sArr.length; sArr[k].equals(s)  &&
//@    \forall int j;0<=j && j<sArr.length; sArr[j]==sArr[k]=>j==k &&
//@      (*aggiunta solo 1 volta*)) &&
//@ \forall int i;0<=i && i<sArr.length;  !sArr[i].equals(s) =>
//@   (\exists int k;0<=k && k<\old(sArr.length); sArr[i].equals(\old(sArr[k])
public static void m3(String[] sArr, String s)
```

Es2

Quesito 1

```
public class Porta{
    private int height;
    private int width;
    private boolean isOpen;

    //@private invariant Height()==height && Width()==height &&
    //@isOpen==isOpen() && isOpen==!isClosed()

    //@public invariant Height()>0 && Width()>0
    //@isOpen()==!isClosed()
    public Porta(int height;int width){
        this.height=height;
        this.width=width;
    }

    //@ensures isOpen()==true
    public void apri(){
        isOpen=true;
    }

    //@ensures isOpen()==true
    public void chiudi(){
        isOpen=false;
    }

    public int Height(){
        return height;
    ]

    public int Width(){
        return width;
    }

    public boolean isOpen(){
        return isOpen;
    }

    public boolean isClosed(){
        return !isOpen;
    }
}
```

Quesito 2

La porta magica dovrebbe essere superclasse, infatti, se fosse sottoclasse violerebbe la regola dei metodi e un utilizzatore si troverebbe sorpreso se, aspettandosi una porta normale, dovesse dire una parola magica.

Quesito 3

```
//@requires pm!=null
//@ensures isOpen()<==>(\old(isOpen() || pm.equals(getParolaMagica())
public void apri(String pm);
```

Es 3

```
public class SistemaCasse{
    private int casseOccupate;
    private int clienti;
    private int turno;
    private ArrayList<Cassa> casse;

    private class Cassa{
        private boolean occupata;
        public void Occupa(){
         occupata=true;
        }
        public void isOccupata(){
            return occupata;
        }
        public void Libera(){
            occupata=false;
        }
    }

    public SistemaCasse(int n){
        casseOccupate=0;
        turno=1;
        clienti=1;
        casse=new ArrayList<Cassa>();
        for(int i=0;i<n;i++){
            casse.add(new Cassa());
        }
    }

    public synchronized int Accodati(){
        int numero=clienti++;
        while(casseOccupate==casse.size() || turno!=numero){
            wait();
        }
        turno++;
        return assegnaUnaCassa();
    }

    private synchronized int assegnaUnaCassa(){
        int numeroCassa=0;
        while(casse[numeroCassa].isOccupata(){
            numeroCassa++;
        }
        casse[numeroCassa].Occupa();
        casseOccupate++;
        return numeroCassa;
```

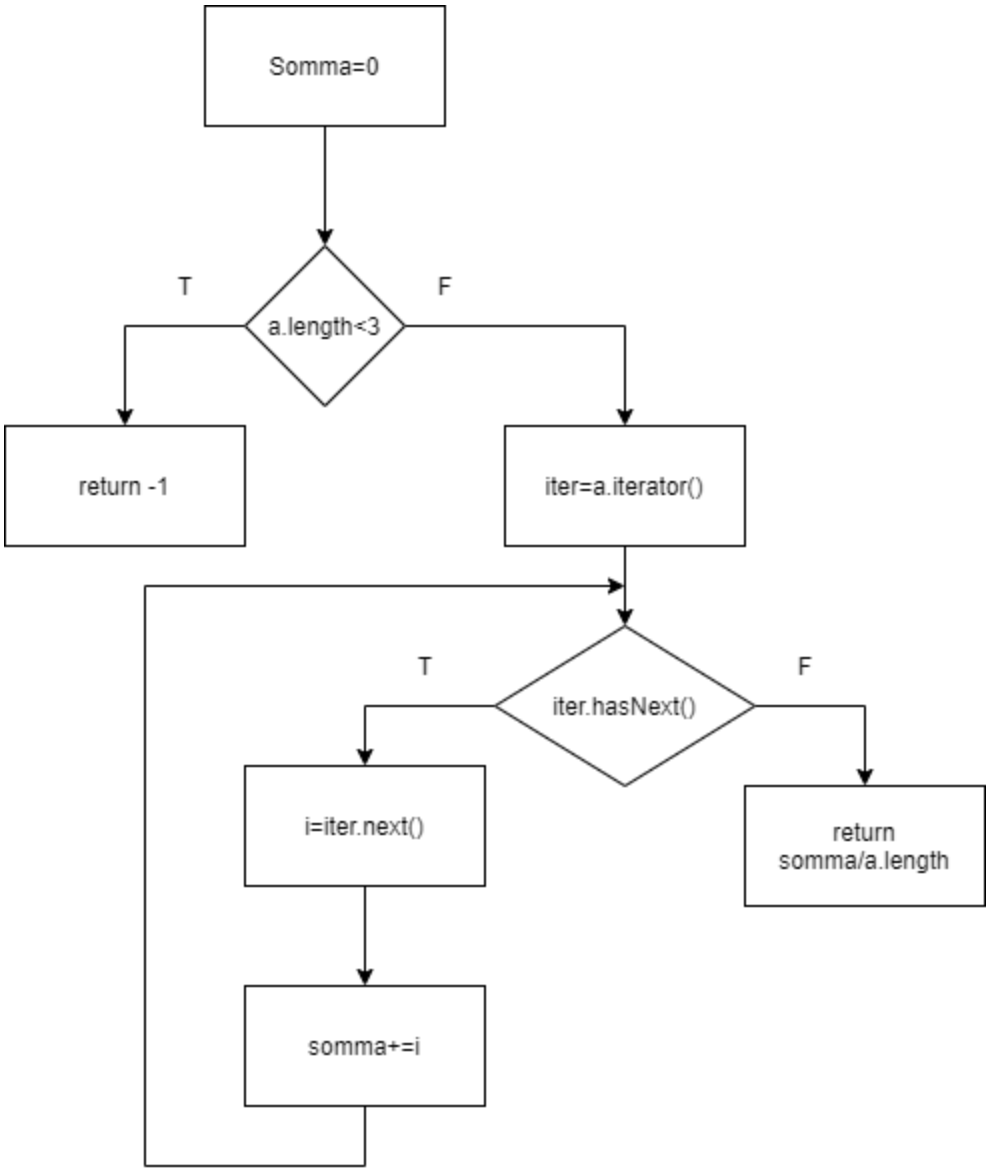
```
    }
    public synchronized void LiberaCassa(int numero){
        casse[numero].Libera();
        casseOccupate--;
        notifyAll();
    }
}

public class MainClass{
    public static void main(String[] args){
        SistemaCasse sc=new SistemaCasse(10);
        for(int i=0;i<30;i++){
            new Thread(new Runnable(){
                public void run(){
                    int cassa=sc.Accodati();
                    try{
                        Thread.sleep((int)(Math.random()*1000));
                    }
                    catch (InterruptedException e){
                    }
                    finally{
                        sc.Libera(cassa);
                    }
                }
            });
        }
    }
}
```

Es 4

```
public static int consonanti(List<String> list){
    return list.stream().reduce(0,(acc,element)->{
        for(int i=0;i<element.length();i++){
            if("aeiouAEIOU".contains(element.charAt(i))==false)
                acc++;
        }
        return acc;
    });
}
```

Es 5



Quesito 2

Test	Copertura	Valore restituito effettivo
{1,2,3}/2	7/8	2
{1,1}/-1	3/8	-1
{1,2,3,4,5}/3	7/8	3
{-1,-2,-3,-4,-1}/-2.2	7/8	-2

Test	Copertura	Valore restituito effettivo
{}/-1	3/8	-1

Quesito 3

Il penultimo test andrebbe in errore, poichè il valore tornato è -2 e non -2.2