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 NumeroBinario somma(NumeroBinario n){
    int dim=Math.max(this.dimensione(),n.dimensione());
    if(this.dimensione()==n.dimensione())
        dim++; //bit di overflow

    Iterator<Integer> iter1=this.destraSinistra();
    Iterator<Integer> iter2=n.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 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 boolean simmetrico(){
    int dim=this.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

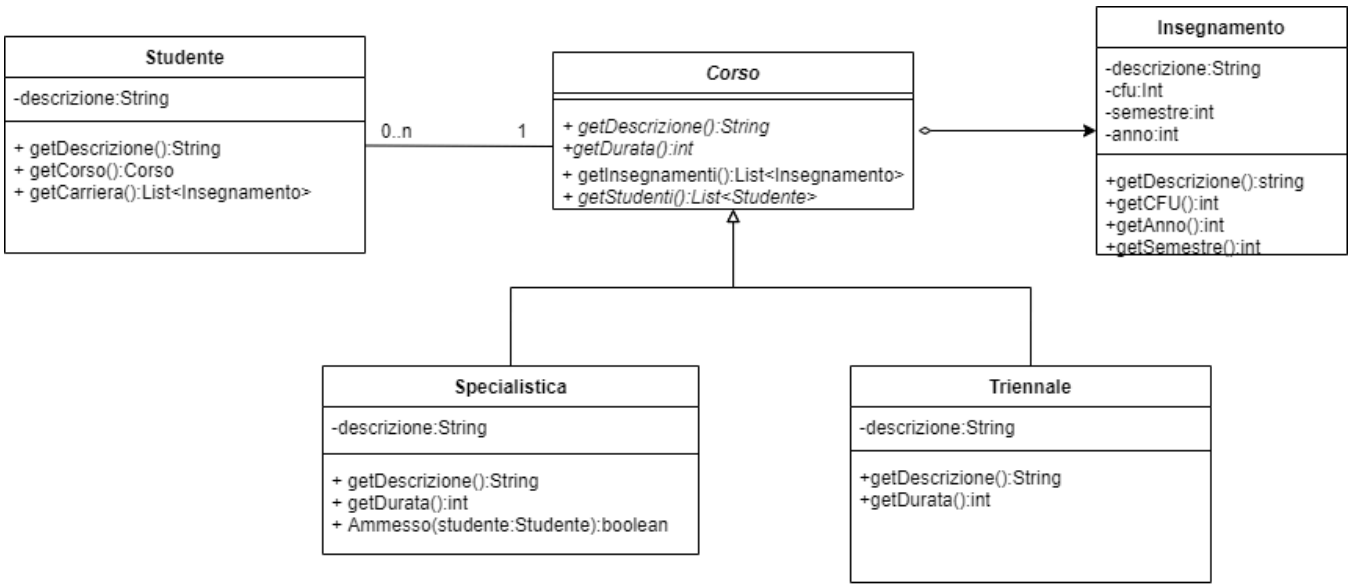
In questo caso, FilmMuto deve essere sottoclasse, poichè la postcondizione di JML sarebbe

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

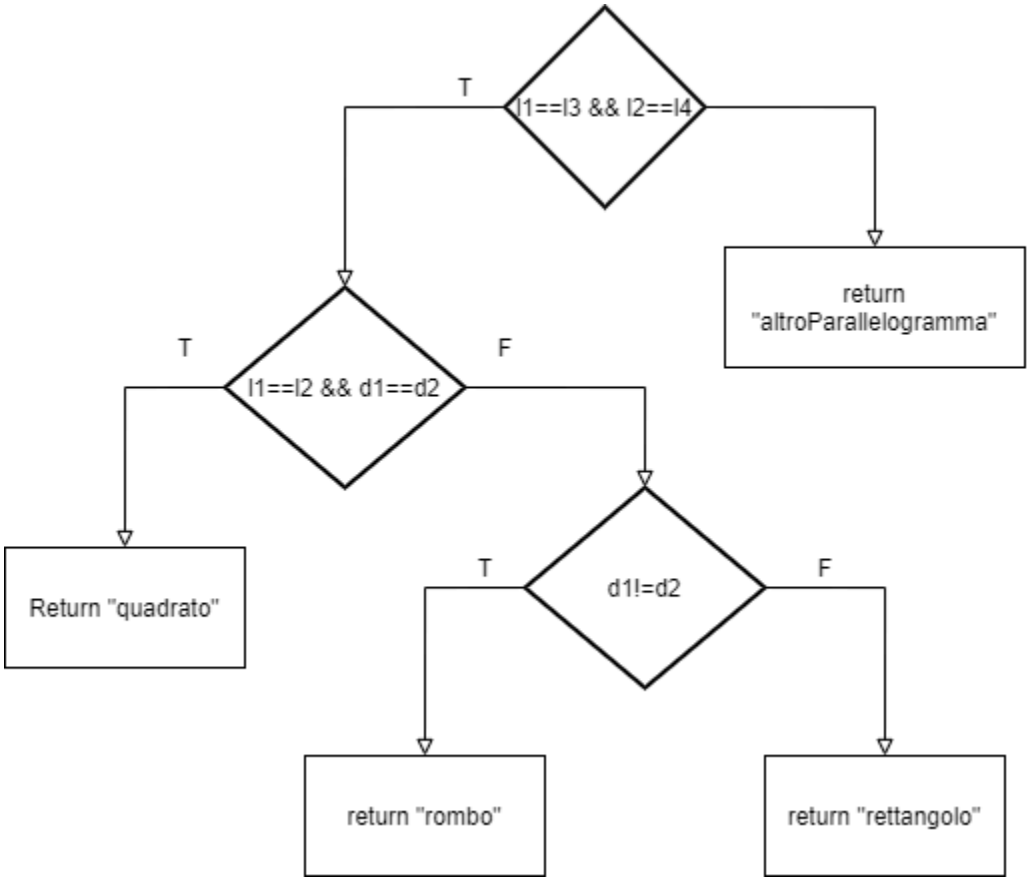
```
public static List<String> startWithA(List<String> list){
    return list.stream().filter(x->x.charAt(0)='a').collect(Collections.toList());
}
```

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

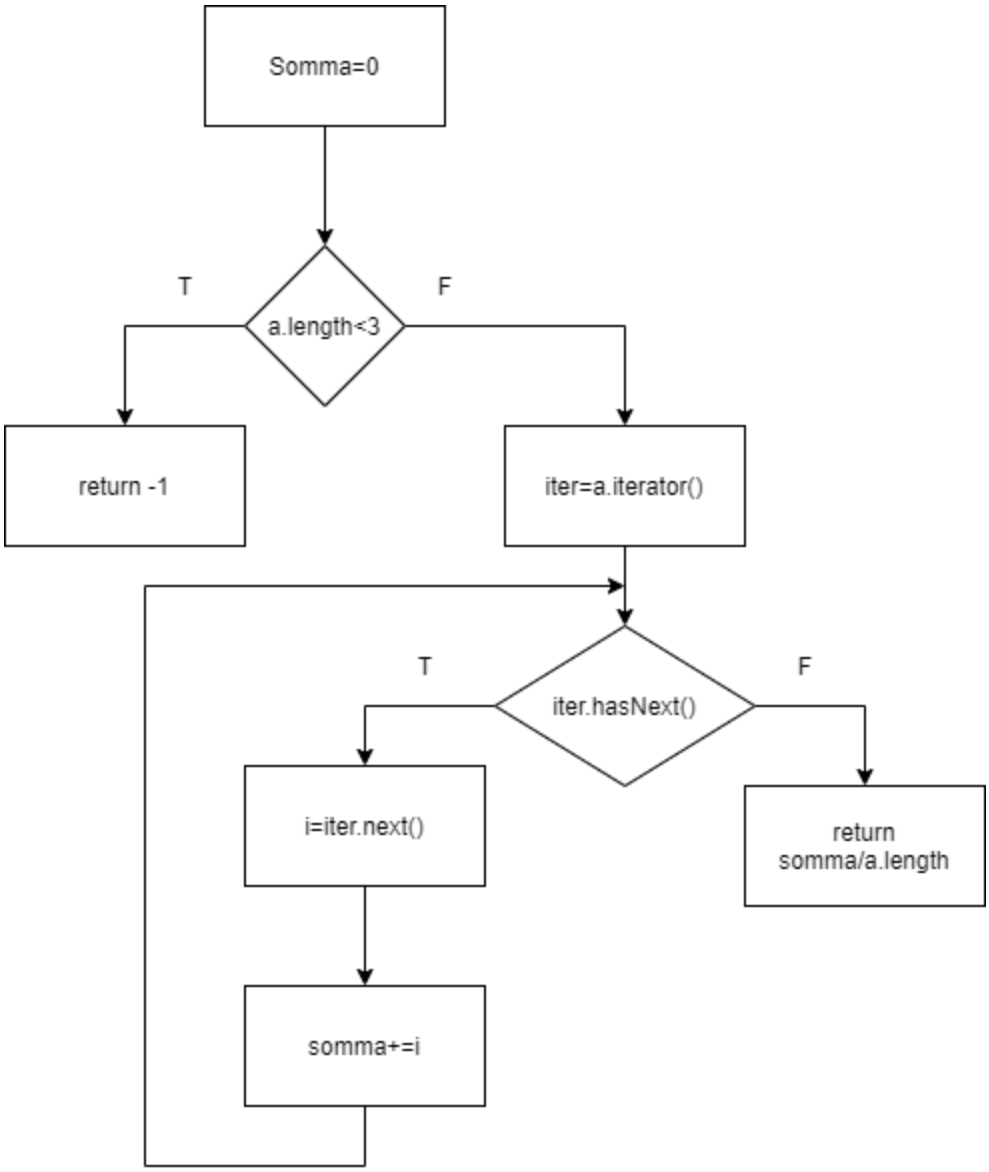
```
public class SistemaCasse{
    private int casseOccupate;
    private int clienti;
    private int turno;
    private ArrayList<Cassa> casse;

    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){
            try{
                wait();
            }
            catch (InterruptedException e){}
        }
        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()
            .mapToInt(element->reduceString(element))
            .reduce(0, (acc,element) -> acc+element);
}
public static int reduceString(String element){
    return Arrays.stream(element.split(""))
            .filter(x->!"aeiouAEIOU".contains(x))
            .mapToInt(x->x.length())
            .reduce(0,(a,e)->a+e);

}
```

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

Esame 12 Giugno 2020

Es1

Parte A

```
//@requires in!= null && (\forall int i;0<=i && i<in.size(); in.get(i)!=null)
//@ensures (\forall int i; 0<=i && i<in.size();
//@ (in.get(i).length()%2==0 && (\num_of int k;0<=k && k<in.get(i).length(); in.get(i).charAt(k)==
//@ \forall int i; 0<= i && i<\result.size(); \exists( int k; 0<=k && k<in.size(); in.get(k).equal

public static ArrayList<String> m1(ArrayList<String> in);

//@requires c!=null
//@ ensures \result <==> (\exists int k; 0<=k && k<c.length; (\exists int i; 0<=i && i<c.length &&
public static boolean m2(char[] c);

//@requires in!= null && (\forall int i;0<=i && i<in.size(); in.get(i)!=null)
//@ensures (\forall int i; 0<=i && i<=\old(in.size());
//@ (\num_of int k;0<=k && k<\old(in.size()); \old(in.get(k))==\old(in.get(i)))==(\num_of int k;0<
//@ && (* ogni elemento vecchio è contenuto ancora in "in" nella stessa quantità*)
//@ !(\exists int i;0<=i && i<=\old(in.size()); \old(in.get(i))%n==0) => ((\exists( int i;0<=i &&
//@ && (* Il resto della divisione =0 comprende già l'esistenza di n e quindi la lugnhezza aumenta
//@ (\exists int i;0<=i && i<=\old(in.size()); \old(in.get(i))%n==0) => (in.size()==\old(in.size()
public static void m3(ArrayList<Integer> in, int n);
```

Parte B

```
public class Rettangolo{
    private double height;
    private double width;

    //@private invariant this.height==Heigth() && this.width==Width()
    public Height(){...}
    public Width(){...}

    //@requires newWidth>0 && newWidth!=Height()
    //@ensures Width()=newWidth
    public void cambiaLarghezza(int newWidth){
    }
```

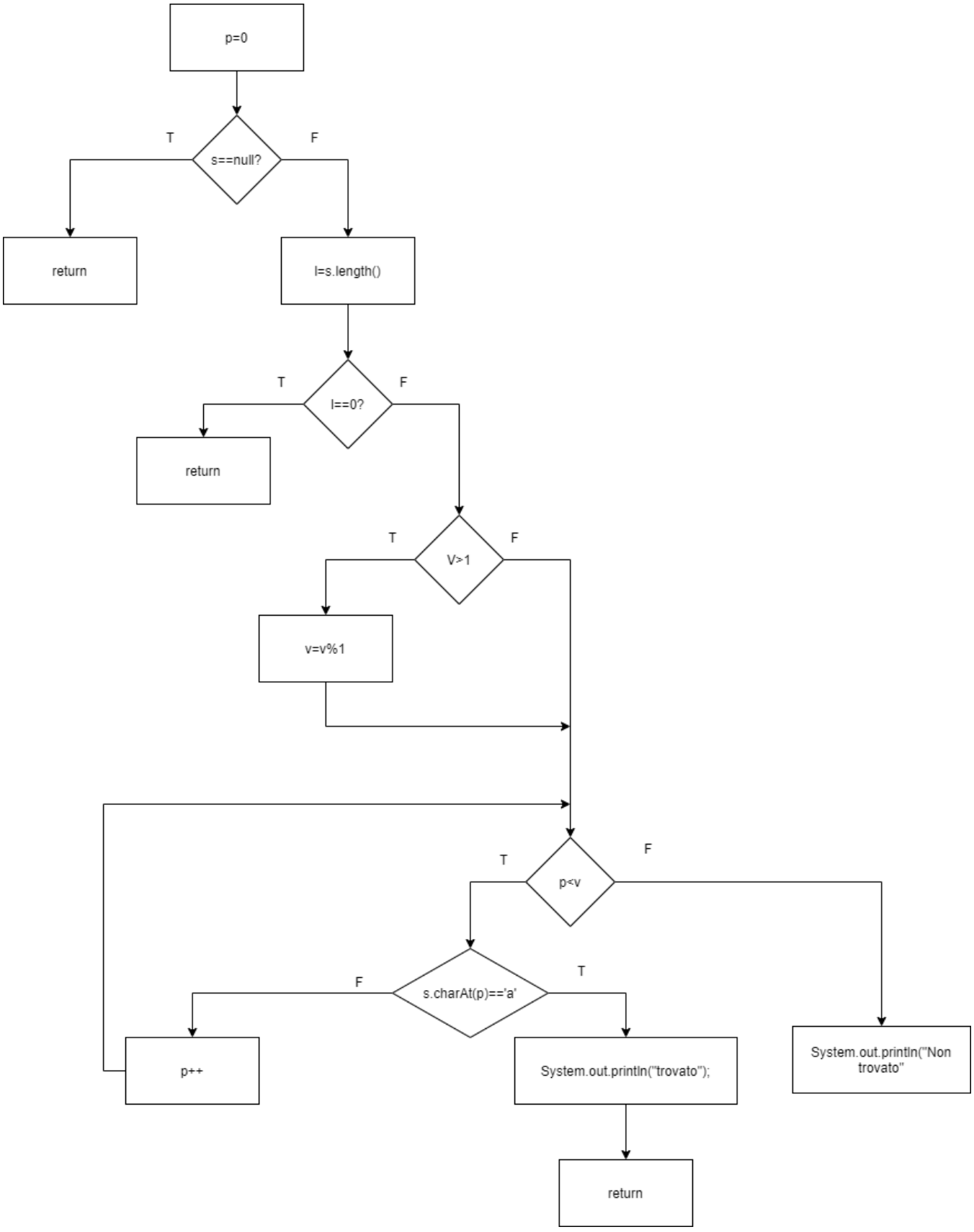
Es3

```
public class Museo {
    public static final int MAX=10;
    private int numTuristi;
    public Museo(){
        numTuristi=0;
    }
    public synchronized void Entra() throws InterruptedException {
        while(numTuristi==MAX) wait();
        numTuristi++;
        System.out.println("Entrato 1 turista, num Turisti= "+numTuristi);
    }
    public synchronized void Esci(){
        numTuristi--;
        System.out.println("Uscito 1 turista, num Turisti= "+numTuristi);
        notifyAll();
    }
}
public class Main {
    public static final int MAX_ATTESA=10*60*1000; //mm*ss*ms
    public static void main(String[] args){
        Museo m=new Museo();
        for(int i=0;i<100;i++){
            Thread turista=new Thread(new Runnable(){
                public void run(){
                    try {
                        Thread.sleep((int)(MAX_ATTESA*Math.random()));
                    } catch (InterruptedException e) {
                    }

                    try {
                        m.Entra();
                        int minuti=(int)(Math.random()*3); //0,1 or 2
                        Thread.sleep(minuti*1000*60);
                        m.Esci();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
            turista.start();
        }
    }
}
```

Es3

```
public static int methodFunz(int v, List<String> values){
    if (values == null || v == 0) return 0;
    return values.stream().filter(x->x.length()>=3).mapToInt(x->x.charAt(0)!='A'?x.length():1)
}
```

Es4


p=0
s==null?
T
F
return
l=s.length()
l==0?
T
F
return
V>1
T
F
v=v%1
p<v
T
F
s.charAt(p)=='a'
F
T
p++
System.out.println("trovato");
System.out.println("Non trovato"
return

Copertura tutte istruzioni e branch (Non ci sono differenze):
foo(null,0)
foo("",0)
foo("a",2)
foo("b",1)
foo("a",1)

Non è possibile eseguire il ciclo while 2 volte, infatti se viene fornito un valore di v>1 (2,3...) si ottiene che v viene messo =0 ed esce sempre dal while, quindi, l'unico valore assegnabile a v per eseguire il ciclo while è 1, ma così facendo il ciclo viene eseguito solo 1 volta.

4 Febbraio 2020

Es1

```
public enum Tipo{
    Carta,Vetro,Metallo,Indifferenziata
}
public class Contenititore{
    private Tipo tipologia;
    private double capienza;
    private double spazioOccupato;

    //@private invariant tipologia==getType() &&  capienza==getCapienza()
    //@ && spazioOccupato== capienza-getSpazioLibero()

    //@requires cap>0
    //@ensures getType==type && getCapienza==cap && getSpazioLibero()==cap
    public Contenitore(double cap, Tipo type){
        capienza=cap;
        tipologia=tipe;
        spazioOccupato=0;
    }

    //@public invariant getCapienza>0 && SpazioLibero>=0
    public Tipo getType(){
        return this.tipologia;
    }
    public double getCapienza(){
        return this.capienza;
    }
    public double getSpazioLibero(){
        return this.capienza-spazioOccupato;
    }

    //@ensures (s.getType==this.getType() && \old(getSpazioLibero())>=s.getVolume())=>
    //@ this.getSpazioLibero==(\old(this.getSpazioLibero())-s.getVolume())
    //@signals (WrongTypeException e) s.getType()!=this.getType()
    //@signals (NoSpaceExcpetion e) \old(getSpazioLibero())<s.getVolume()
    public void gettaVia(Spazzatura s) throws NoSpaceExcepetion, WrongTypeExcpetion
    {
        ...
    }
}
public class Spazzatura{
    private Tipo tipologia;
    private double volume;

    //@private invariant tipologia==getType() &&  volume==getVolume()

    //@requires vol>0
```

```
    //@ensures getVolume()==vol && getType()==type
    public Contenitore(double vol, Tipo type){
        volume=vol;
        tipologia=tipe;
    }

    //@public invariant getVolume()>0
    public Tipo getType(){
        return this.tipologia;
    }
    public double getVolume(){
        return this.volume;
    }

}
```

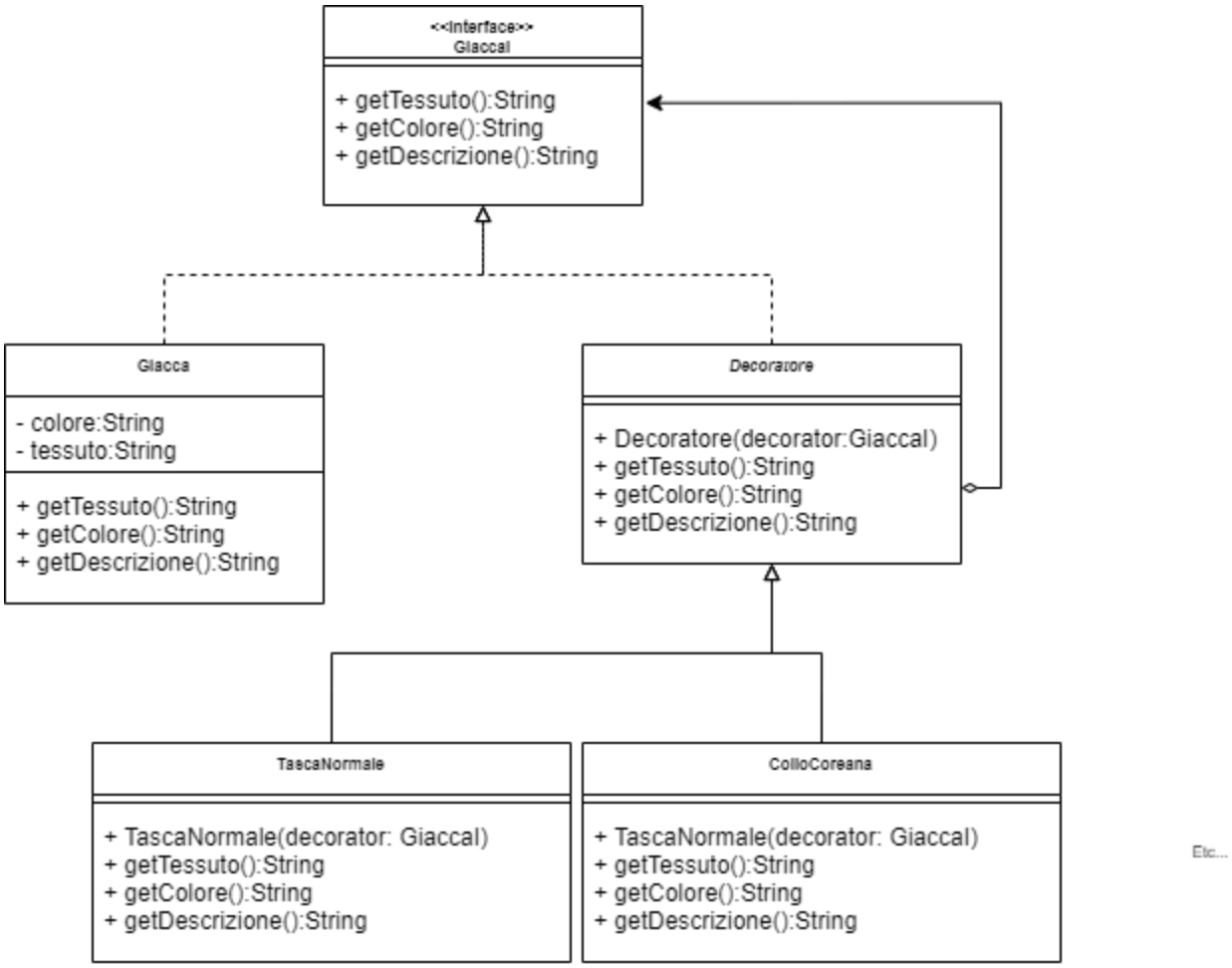
Es2

```
public class Store {
    public static final int MAX_PANETTONI_CLIENTE=3;
    public static final int MAX_NUM_CLIENTI=5;
    private Object lockOnNumPanettoni;
    private int numPanettoni;
    private int numClienti;

    public Store(int n){
        lockOnNumPanettoni=new Object();
        numPanettoni=n;
        numClienti=0;
    }

    public synchronized void Entra() throws InterruptedException{
        while(numClienti>=MAX_NUM_CLIENTI) {
            wait();
        }
        numClienti++;
        //System.out.println("Clienti nello store: "+numClienti);
    }

    public boolean Acquista(int np){
        if(np>3){
            //System.out.println("Troppi panettoni richiesti");
            return false;
        }
        else{
            synchronized (lockOnNumPanettoni){
                if(numPanettoni<np){
                   // System.out.println("Acquisto rifiutato,  panettoni insufficienti \n Panetton
                    return false;
                }
                numPanettoni-=np;
                System.out.println("Acquisto effettuato,\n Panettoni rimasti:" +numPanettoni+"\n R
            }
        }
        return true;
    }
    public synchronized void Esci(){
        numClienti--;
        //System.out.println("Cliente uscito, Clienti nello store: "+numClienti);
        notifyAll();
    }
}

public class Main {
```

```
    public static void main(String[] args){
        Store s=new Store(50);
        for(int i=0;i<10000;i++) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    try{
                        Thread.sleep((int)(Math.random()*2000));
                        s.Entra();
                        Thread.sleep((int)(Math.random()*2000));
                        s.Acquista((int)(Math.random()*5+1));
                        s.Esci();
                    }
                    catch (Exception e)
                    {

                    }
                }
            }).start();
        }

    }
}
```

Es3

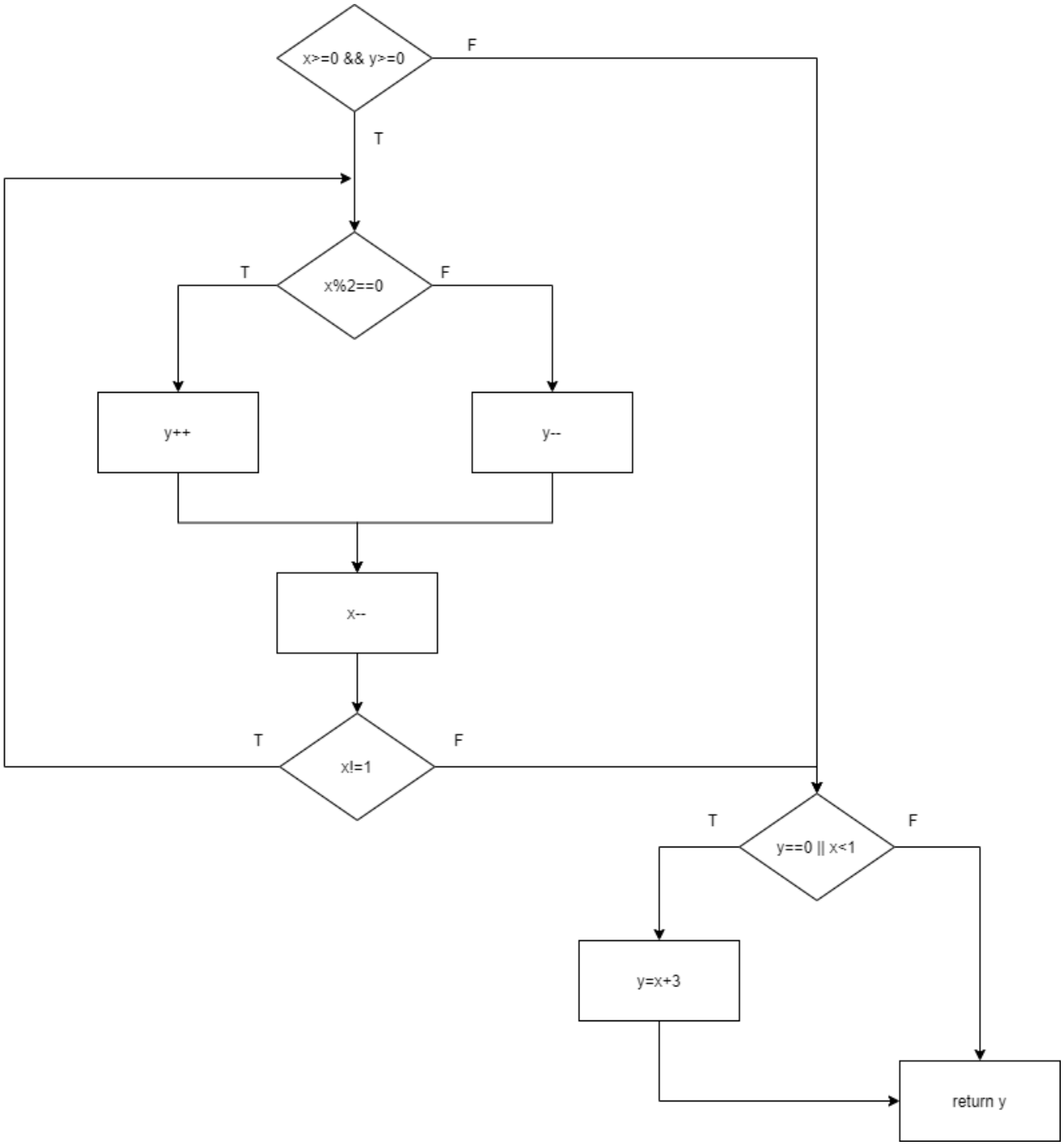
Il design pattern decorator è quello più flessibile utilizzabile in questo caso, permette infatti di aggiungere eventuali nuove decorazioni alla giacca senza modificare il codice esistente



Es4

```
public static int fooFunz(Integer v,List<Integer> values){
    if (values == null || v == null ) return 0;
    return values.stream().map(x-> x%3==0? v:x%3).reduce(0,(acc,x)->acc+x);
}
```

Es 5



Test per coprire tutte le istruzioni: `Test(3,0)`

Per coprire tutti i branch basta aggiungere il test `Test(1,-1)`

Per eseguire il cammino 2,4,5,8,9,10 e 12 basta il test `Test(2,0)`

14 Gennaio 2020

Es 1

```
public class Enum Valore{
    Uno,Due,Tre,Quattro,Cinque,Sei,Sette,Otto,Nove,Dieci,Fante,Cavallo,Re
}
public class Enum Seme{
    Picche,Cuore,Quadri,Fiore
}
public class MazzoDiCarte{
    Carta[] mazzo;

    //@private invariant \forall int i;0<=i && i<mazzo.length;
    //@ this.getMazzo()[i].equals(this.mazzo[i])

    //@ensures getMazzo().length=52 &&
    //@ (\forall int i;0<=i && i<getMazzo().length; getMazzo()[i]!=null &&
    //@ !(\exists int k;0<=k && k<getMazzo().length && k!=i;
    //@       getMazzo()[i].equals(getMazzo()[k])))
    public MazzoDiCarte(){
        //....
    }

    public Carta[] getMazzo(){
        Carta[] mazzoBis=new Carta[52];
        int i=0;
        for(Carta c:mazzo){
            mazzoBis[i]=new Carta(mazzo[i].getSeme(),mazzo[i].getValore())
        }
        return mazzoBis;
    }

    //@ensures (\num_of int i; 0<=i && i<getMazzo().length; getMazzo()[i].equals(\old(getmazzo()[i
    public void mescola(){
        //...
    }
}
public class Carta{
    private Valore valore;
    private Seme seme;

    //@private invariant this.valore=getValore() && this.seme=getSeme()

    public class Carta(Seme s,Valore v){
        this.seme=s;
        this.valore=v;
    }

    public Valore getValore(){
```

```
        return valore;
    }
    public Seme getSeme(){
        return seme;
    }

    public boolean /*@ pure @*/ equals(Object a){
        if(!(a instance of Carta)) return false;
        return this.valore==a.valore && this.seme==a.seme;
    }

}
```

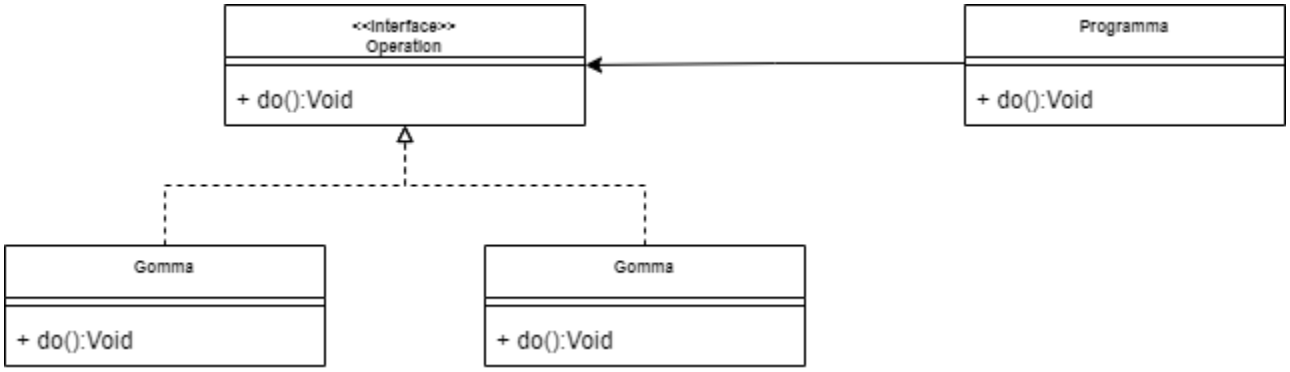
Es 2

```
public class Shop {

    public static final int MAX_PERSONE=5;
    public static final int MAX_BIGLIETTI=3;

    private int numClienti;
    private Object lockBiglietti;
    private int numBiglietti;

    public Shop(int numBiglietti){
        numClienti=0;
        this.numBiglietti=numBiglietti;
        lockBiglietti=new Object();
    }

    public synchronized void Entra() throws InterruptedException{
        while(numClienti==MAX_PERSONE) wait();
        numClienti++;
    }

    public int Compra(int numB){
        if(numB>MAX_BIGLIETTI){
            numB=MAX_BIGLIETTI;
        }
        int acquistati=numB;
        synchronized (lockBiglietti){
            if(numB>numBiglietti){
                acquistati=numBiglietti;
            }
            numBiglietti-=acquistati;
        }
        return acquistati;
    }
    public synchronized void Esci(){
        numClienti--;
        notifyAll();
    }
public class Main {
    public static void main(String[] args){
        Shop s=new Shop(500);
        for(int i=0;i<1000;i++){
            new Thread(new Runnable() {
                @Override
                public void run() {
                    try {
                        Thread.sleep((int)(Math.random()*5000));
```

```
                        s.Entra();
                        Thread.sleep((int)(Math.random()*5000));
                        s.Compra((int)(Math.random()*5+1));
                        s.Esci();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
        }
    }
}
```

Es 3

Il pattern strategy risolverebbe il problema del cambio a run-time delle operazioni da eseguire, si otterebbe quindi un UML così:

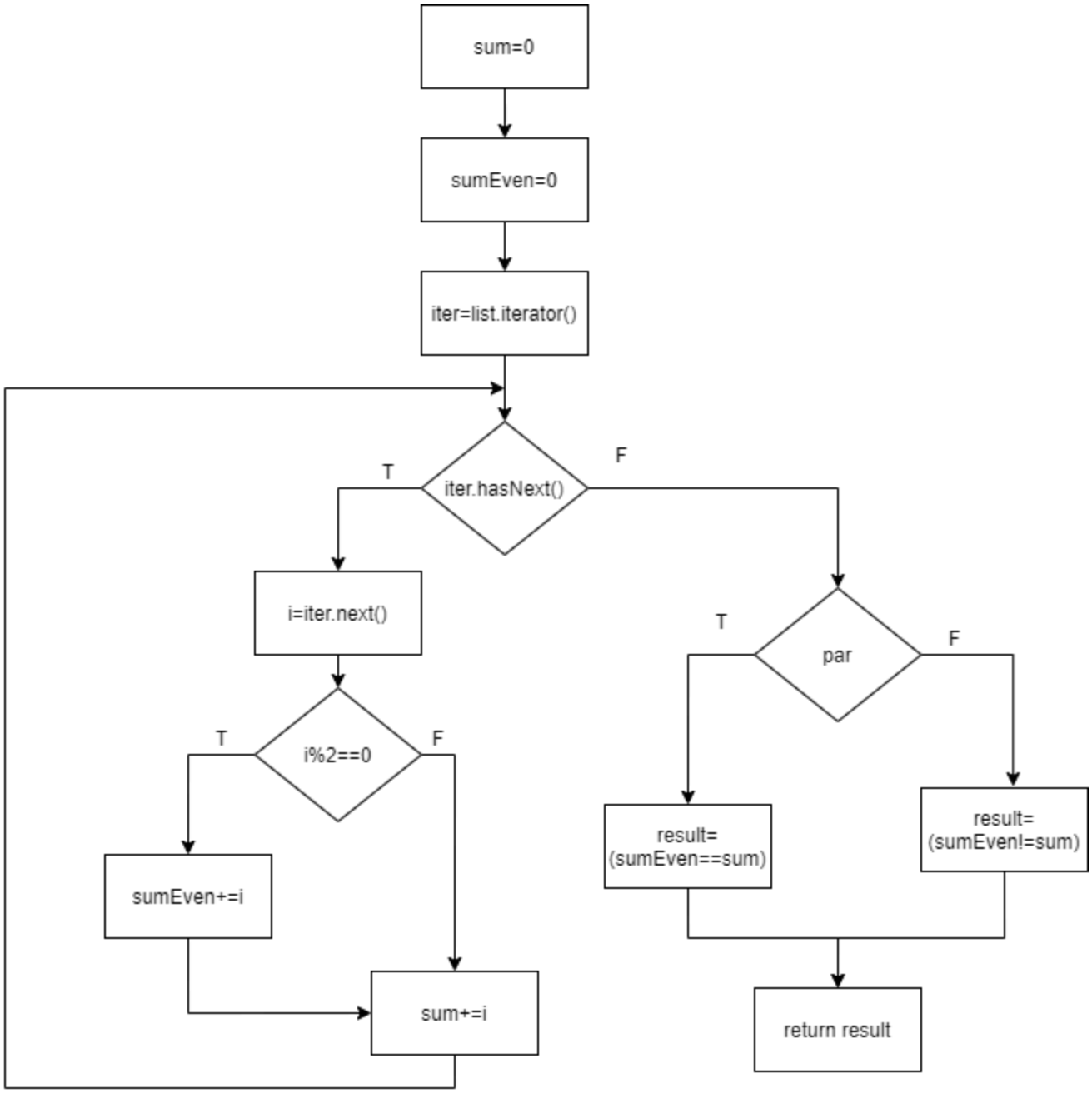


Es 4

L'output sarebbe

```
true A
true A
true B
true A
called 1 on instance of B
called 1 on instance of B
called 2 on instance of B
```

Es 5



Copertura tutte le istruzioni:

```
foo({2,1},true)
foo({2},false)
```

Copertura tutti i branch: Bastano i due test per coprire le istruzioni.