Esame di Programmazione III, Progr. in Rete e Lab., Ist. di Progr. in Rete e Lab., A.A. 2022/23 -- 22 giugno 2023

Esercizio 1 (10 punti)

Si simuli un'esecuzione del programma riportato sotto supponendo che **le liste di lock e di wait siano gestite con una politica First In First Out**. Per la simulazione si riportino tutti i cambiamenti di stato del programma, e le stampe che vengono fatte, assumendo che lo stato sia descritto come segue: ***{[lista di lock], [lista di wait], [inchiostroNero, inchiostroColor]}***. Si assuma che lo **stato iniziale** del programma sia: **{[s1, s2, s3, c1, c2, c3], [ ], [1, 1]}**, dove s1 è il primo elemento della lista e c3 l'ultimo.

```
public class Esercizio {
   public static void main(String[] args) {
      Stampante printer = new Stampante();
      Caricatore c1 = new Caricatore(printer, "nero");
      Caricatore c2 = new Caricatore(printer, "color");
      Caricatore c3 = new Caricatore(printer, "nero");
      Stampatore s1 = new Stampatore(printer, "nero");
      Stampatore s2 = new Stampatore(printer, "nero");
      Stampatore s3 = new Stampatore(printer, "color"); }
}
class Stampatore extends Thread {
   private Stampante stampante;   private String colore;
   public Stampatore(Stampante stamp, String colore) {
      stampante = stamp;  this.colore = colore;  start();
   }
   public void run() {
      if (colore.equals("nero"))  stampante.stampa("nero");
      else stampante.stampa("color"); }
}
class Caricatore extends Thread {
   private Stampante stampante; private String colore;
   public Caricatore(Stampante stamp, String colore) {
      stampante = stamp; this.colore = colore;  start(); }
   public void run() { stampante.caricaInchiostro(colore); }
}
class Stampante {
   private int inchiostroNero = 1, inchiostroColor = 1;
   public synchronized void stampa(String colore) {
       if (colore.equals("nero"))  {
          while (inchiostroNero<=0) {
                try {wait();} catch (InterruptedException e) {}
           }
          System.out.println(colore); inchiostroNero--;
       } else {
          while (inchiostroColor<=0) {
                try {wait();} catch (InterruptedException e) {}
          }
          System.out.println(colore); inchiostroColor--;
      }
  }
  public synchronized void caricaInchiostro(String colore) {
      if (colore.equals("nero"))  inchiostroNero++;   else inchiostroColor++;
      System.out.println("caricato " + colore);   notify();
   }}
```

Esercizio 2 (10 punti)

Si consideri il seguente codice.

1. Si dica se il codice compila correttamente o meno; nel caso non compilasse, spiegare il perché e correggere il codice in modo da risolvere i problemi.
2. Scrivere l'output a video del programma originale oppure, nel caso siano state apportate modifiche al codice, di quello modificato.

```
class A {
	public void m(A a) {
		System.out.println("m_AA");
	}
	public void m(B b) {
		System.out.println("m_AB");
	}
}
class B extends A {
	public void m(A a) {
		System.out.println("m_BA");
	}
	public void m(A a, B b) {
		super.m(a);
		super.m(b);
		m(a);
	}
}
public class AppEsame {
    public static void main(String[] args) {
		A a = new A();
		a.m(a);
		System.out.println();
		B b = new B();
		b.m(b);
		System.out.println();
		a.m(b);
		System.out.println();
		b.m(a);
		System.out.println();
		b.m(b, b);
		System.out.println();
		b.m(a, b);
	}
}
```

Esercizio 3 (10 punti)

Si sviluppino i seguenti punti:

1. Si descriva la Java Reflection, spiegando a cosa serve e su quali classi si basa, in dettaglio.
2. Si faccia un semplice esempio di applicazione della Java Reflection.

POSSIBILI SOLUZIONI

Esercizio 1

{[s1, s2, s3, c1, c2, c3], [], [1, 1]}
stampa “nero”
{[s2, s3, c1, c2, c3] [], [0,1]}
{[s3, c1, c2, c3], [s2], [0, 1]}
stampa “color”
{[c1, c2, c3], [s2], [0, 0]}
stampa “caricato nero”
{[c2, c3, s2],[], [1, 0]}
stampa “caricato color”
{[c3, s2], [], [1, 1]}
stampa “caricato nero”
{[s2], [], [2, 1]}
stampa “nero”
{[], [], [1, 1]}

Esercizio 2

Il codice compila correttamente.
Output a video dell’esecuzione del codice: