(A) CONOSCENZA TERMINOLOGICA

Dare una breve descrizione dei termini introdotti:

- Animazione
- Classe **Timer**
- **void start**()
- **void stop**()
- **void setDelay**(**int** delay)
- **int getDelay**()

(B) CONOSCENZA E COMPETENZA

Rispondere alle seguenti domande producendo anche qualche esempio

B1) Conoscenza

1. Cosa vuol dire *animazione*?
2. Come si crea *l'effetto di animazione*?
3. Come funziona l'oggetto **Timer**?

B1) Competenza

1. Come si realizza un'applicazione per *visualizzare animazioni*?
2. Cosa deve contenere la classe *oggettoTimer*?

(C) ESERCIZI DI COMPRENSIONE

1. Il **Canvas** di Java può essere usato per visualizzare immagini in movimento, una tecnica che prende il nome di …… . Per realizzare l'animazione ci si serve della classe ……, i cui oggetti hanno la caratteristica di emettere …….. in modo temporizzato.
2. Un'applicazione temporizzata prevede almeno l'implementazione di due classi:
   a. una, che contiene il costruttore di un oggetto di classe **Timer**, per la generazione di ……. di sincronismo, che implementa la classe di ascolto …….. , per gestire gli …….. prodotti dal timer e che, tramite un il metodo di ascolto ……. produca in corrispondenza al timer, effetti grafici o visivi sincronizzati;
   b. una seconda classe che crea un contenitore (un ….. , o un ……) con determinate caratteristiche e crea un oggetto *t* di classe **Timer** e lo registra.
3. Completare la seguente tabella con le caratteristiche della classe **Timer**

Metodi della classe Timer	Descrizione
Timer (**int** *delay*, **ActionListener** *listener*)	
void start ()	
void stop()	
void setDelay (**int** *delay*)	
int getDelay ()	

4. Scrivere a fianco di ciascuna riga di codice, un breve commento che ne descriva il significato.

```
class oggettoTimer extends Canvas implements ActionListener
{   Timer t;
      int delay;                                  // ……
      public oggettoTimer (int delay)             // ……
      {
         t= new Timer (delay, this);              // ……
            t.start();                            // ……
       }
       public void actionPerformed (ActionEvent evt)  // ……
       { gestione evento }                        // ……
}
```

5. Scrivere a fianco di ciascuna riga di codice, un breve commento che ne descriva il significato.

```
public class AnimazioneImmagini                   // ……
{
    public static void main(String args[] )       // ……
    {
        Frame f = new Frame ("Animazione");       // ……
        f.setBackground(Color.lightGray);         // ……
        oggettoTimer t=new oggettoTimer (100);    // ……
        f.add(t);                                 // ……
        f.addWindowListener(new Adattatore());    // ……
        f.setLocation(200, 200);                  // ……
        f.setSize(750, 400);                      // ……
        f.setVisible(true);                       // ……
    }
}
```

6. Completare la seguente tabella, indicando per ciascuna frase, se vera (V) o falsa (F).

	Vero	**Falso**
La classe *oggettoTimer* contiene il costruttore dell'oggetto **Timer**		
int *delay* deve essere un attributo della classe *oggettoTimer*		
Il metodo t.**start**() va posto nel metodo **actionPerformed**()		
Il metodo **actionPerformed**() contiene la gestione dell'evento.		
Il *delay* del **Timer** si imposta nell'applicazione di prova.		

(D) ESERCIZI DI APPLICAZIONE

1. Creare un'applicazione che mostri a ciclo continuo, i valori progressivi di un contatore inserito in un riquadro, in una stessa posizione, prefissata. Da input, si deve poter dare la posizione e la velocità di avanzamento del contatore.
2. Predisporre un adeguato numero di file immagine da usare come fotogrammi. Creare un'applicazione che le visualizzi in sequenza animata, come mostrato nello schema dell'Unità.
3. Modificare l'applicazione precedente, in modo che, se le immagini sono molte, i loro nomi siano inseriti in un array del tipo:
   String fileimmagine[] = **new String**[100];
   e che le rispettive immagini siano memorizzate nell'array parallelo
   Image immagini[] = **new Image**[100];
4. Creare un'applicazione che, mediante il **Timer**, realizzi un "banner", ossia una scritta scorrevole in senso orizzontale rispetto allo schermo. Prevedere opportuni parametri per ottenere lo scorrimento verso sinistra o verso destra.
5. Creare una classe Contatore, che servendosi di **Timer**, mostri in una casella il valore di un contatore che parta da 0 e si incrementi con una velocità iniziale. L'interfaccia deve disporre di due pulsanti *Aumenta* e *Diminuisci*: il primo aumenta, il secondo diminuisce la velocità del contatore. Disporre anche di un pulsante *Reset* che predisponga il contatore alla condizione iniziale, di un pulsante *Stop* che arresta il conteggio e di un pulsante *Esci* per terminare l'applicazione.

(E) ESERCITAZIONI PRATICHE

Esercitazione n. 1

Obiettivo: utilizzo della classe timer per realizzare applicazioni animate.
Titolo: realizzare un'applicazione che mostri una palla che scorre verso destra e quando incontra il bordo della finestra, inverte il movimento e cambia colore. Durante lo scorrimento verso destra, la palla è azzurra, quando scorre verso sinistra, diventa rossa. L'applicazione deve consentire la modifica del raggio della palla e la sua velocità.

Analisi.
La classe deve prevedere l'uso dei seguenti ascoltatori:

Classe di ascolto	**ActionListener**
Oggetto origine	**Timer**
Metodo di ascolto	**actionPerformed** ()
Gestore ascolto	**Non serve se ereditiamo direttamente da ActionListner**
Registrazione	**Non serve se ereditiamo direttamente da ActionListner**

Classe di ascolto	**WindowListener**
Oggetto origine	**Window**
Metodo di ascolto	**windowClosing** ()
Gestore di ascolto	**ChiudiFinestra**
Registrazione	**addWindowListener(new ChiudiFinestra());**

Lo schema della classe è il seguente:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.Timer;
class MuoviCerchio extends Panel implements ActionListener
{
    int raggio;          /* raggio del cerchio */
    int millisec;        /* tempo di refresh del disegno */
    Timer timer;                 /* per regolare l'animazione */
    boolean versoDestra;         /* da che parte va */
    int x_min;           /* ascissa minima corrente del cerchio */

     public MuoviCerchio (int raggio_cerchio, int ritardo)
     {  assegnazione valori attributi, inizializzazione e avvio del timer; }

     public void actionPerformed(ActionEvent e)
      {  Se va verso destra allora
            Se tocca bordo destro
               { cambia verso e decrementa la x; }
            Altrimenti incrementa la x;
        Altrimenti // va verso sinistra
            Se tocca bordo sinistro
               { cambia verso e incrementa la x;   }
            Altrimenti x_min++;
         repaint();

    public void paint(Graphics g)
      {
        Graphics2D g2 = (Graphics2D)g;
        Dimension d = getSize();
        Se si sposta verso destra
            Imposta blu;
        Altrimenti
            Imposta rosso;
        riempi la figura;
    }
    public static void main(String[] args)
      {
        crea la finestra e imposta le caratteristiche di (500, 300) e visibile;
        crea l'oggetto MuoviCerchio mc con i parametri appropriati;
        imposta colore di sfondo;
        registra mc;
        registra l'ascoltatore della finestra;
      }
  }
class ChiudiFinestra extends WindowAdapter
{
            public void windowClosing(WindowEvent e)
            {
                        System.exit(0);
            }
}
```