Progetto Risolutore Polinomi, studente Marco Greco Matricola 200901

Il mio primo progetto (risolutore di Polinomi) si basa sull'implementazione delle seguenti classi:

- ✓ Applicazione. [Progetto]
- ✓ Monomio.
- ✓ Polinomio. (Interfaccia)
- ✓ PolinomioAL.
- ✓ PolinomioAstratto.

Monomio,Polinomio,Polinomio,PolinomioAl e PolinomioAstratto visti a lezione. In questa relazione mi soffermo sulla classe Applicazione (il progetto in sé). A differenza del progetto Sudoku ho implementato una classe unica nella quale gestisco sia la logica dell'applicazione e sia la parte grafica.

Classe Applicazione

La classe Applicazione contiene un main con il fine di creare una nuova FinestraApplicazione e renderla visibile.

```
public class Applicazione {


    public static void main(String[]args) {
        JFrame f = new FinestraApplicazione();
        f.setVisible(true);
    }

}
```

Classe FinestraApplicazione

La classe FinestraApplicazione è quella che gestisce tutto il progetto, dalla logica alla grafica. All'inizio dichiaro un contatore (countSelezionati) con il quale tengo conto del numero di checkbox selezionati; una JtextArea (risultato) dove faccio comparire il risultato di un'operazione, una Stringa (input) che sarà poi il polinomio passato dall'utente attraverso un showInputDialog; i pannelli polipanel (pannello di destra dove saranno contenuti tutti i checkbox de polinomi) e risPanel (il pannello più grande dove contiene la JTextArea risultato); un JScrollPane sp che mi permette di scorrere il pannello dei polinomi se dopo una certa quantità di inserimenti i JCheckBox non rientrano all'interno della finestra; i JMenuItems inserisci, elimina, salvaconnome, carica, esci, somma, differenza, prodotto, derivata, valore, about, version che saranno aggiunti ai JMenu qui dichiarati menucomandi, menuFile e menuhelp.

Inoltre, ho inserito le espressioni regolari utili per identificare un polinomio e un ascoltatore.

```
int countSelezionati;
JTextArea risultato;
String input;
JPanel poliPanel,risPanel;
JScrollPane sp;
JMenuItem inserisci,elimina,salvaconnome,
          carica,esci,somma,differenza,
          prodotto,derivata,valore,about,version;
JMenu menucomandi,menuFile,menuhelp;
String NUM = "[1-9]+";
String VAR = "([xX]|[xX][//^]"+NUM+")";
String OPER = "[\\+\\-]";
String MONOMIO = "("+NUM+VAR+"?|"+VAR+")";
String POLINOMIO ="[\\+\\-]?"+ MONOMIO+"("+OPER+MONOMIO+")*";
String DBL = "([\\-]?[0-9]+|[0-9]+[.][0-9]+)";

AscoltatoreEventi ascoltatore = new AscoltatoreEventi();
```

Ho creato inoltre un ArrayList di polinomi (listaPolinomi) nel quale salvo tutti i polinomi; un ArrayList first utilizzato per ricordarmi il primo JCheckBox premuto in modo da gestire l'operazione differenza; un ArrayList di JCheckBox dove salvo i JCheckBox;

```
ArrayList<Polinomio> listaPolinomi=new ArrayList<>();
ArrayList<JCheckBox> listaCheck=new ArrayList<>();
ArrayList<JMenuItem> comandiPolinomi = new ArrayList<>();
ArrayList<Integer> first = new ArrayList<>(10);
```

Ed infine un ArrayList (comandiPolinomi) per salvarmi i relativi JMenuItem dei comandi (molto utile per questione di efficienza per la loro gestione nella classe AscoltatoreEventi evitando di riscrivere stesse righe di codice per ogni JMenuItem modificando solo ciò che serve in base all'operazione (ogni indice dell'ArrayList mi consente di identificare un comando).

```
for(JMenuItem jmi:comandiPolinomi) {
    if(e.getSource()==jmi) {
```

```
premuto=comandiPolinomi.indexOf(jmi);
```

```
switch(premuto) {
case 0: risultato.append("La somma è pari a:\n\n  "+ris.toString());break;
case 1: risultato.append("La differenza è pari a:\n\n  "+ris.toString());break;
case 2: risultato.append("Il prodotto è pari a:\n\n  "+ris.toString());break;
case 3: risultato.append("La derivata è pari a:\n\n  "+ris.toString());break;
}
```

Nel metodo actionPerformed della classe AscoltatoreEventi.

Costruttore

Nel costruttore inizialmente gestisco i parametri della finestra (titolo, grandezza, posizione, renderla non ridimensionabile, l'operazione di chiusura di default..).

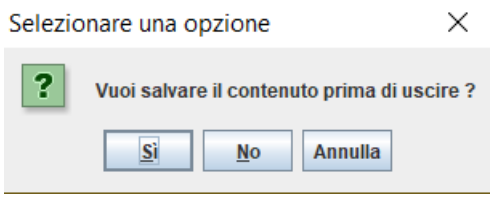
```
public FinestraApplicazione() {
    setTitle("RISOLUTORE POLINOMI");
    setSize(700,500);
    setLocation(570,300);
    setResizable(false);
    setDefaultCloseOperation( JFrame.DO_NOTHING_ON_CLOSE );


    addWindowListener( new WindowAdapter() {
        public void windowClosing(WindowEvent e){
         uscita();
        }
     } );
```

Alla finestra aggiungo un WindowListener passandogli come parametro un WindowAdapter che mi permette di implementare solo il metodo che mi serve, in questo caso windowClosing che una volta chiusa la finestra dell'applicazione invoca il metodo uscita() nel quale gestisco la richiesta del salvataggio dell'operato.

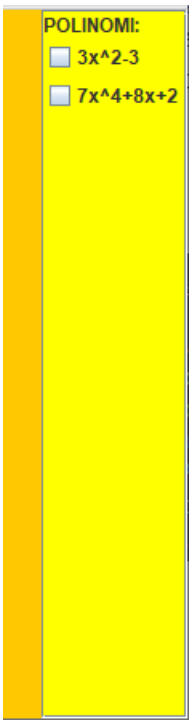
Il metodo uscita() mostra all'utente uno showConfirmDialog posizionato in prossimità del pannello risPanel. Se l'utente sceglie l'opzione NO allora chiude l'applicazione senza errori (0), se invece sceglie l'opzione SI chiamo il metodo doClick() sul JMenuItem salvaconnome in modo da far partire il salvataggio gestito nell'actionPerformed (lo vedremo in seguito). Nei casi restanti chiude lo showConfirmDialog e il programma continua a funzionare.

```
public void uscita() {
    int i=JOptionPane.showConfirmDialog(risPanel, "Vuoi salvare il contenuto prima di uscire ?");
    if(i==JOptionPane.NO_OPTION) System.exit(0);
    if(i==JOptionPane.YES_OPTION) salvaconnome.doClick();
}//uscita
```



- Pannello Polinomi (poliPanel)

Creo quindi un pannello dove saranno inseriti tutti i JCheckBox dei polinomi. Gli ho aggiunto in alto un label con scritto “POLINOMI: “ e settato al pannello un BoxLayout che garantisce l’inserimento dei componenti in sequenza verticalmente (BoxLayout.Y_AXIS). Uno JScrollPane permette lo scorrimento (non orizzontale perché il pannello si allarga in base alla lunghezza del polinomio) del pannello se dopo una certa quantità di inserimenti i JCheckBox non rientrano all’interno dello stesso.

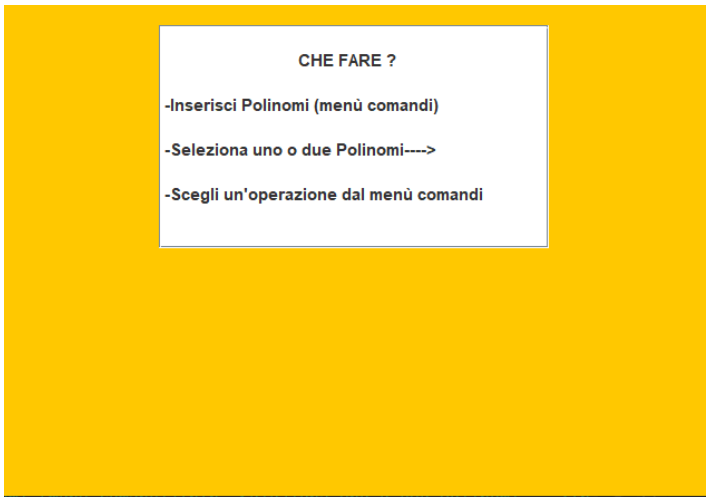



```
//Pannello dedicato ai Polinomi
poliPanel = new JPanel();
JLabel titoloPolinomio=new JLabel("POLINOMI:");
poliPanel.add(titoloPolinomio);
poliPanel.setLayout(new BoxLayout(poliPanel, BoxLayout.Y_AXIS));
poliPanel.setBackground(Color.YELLOW);
sp = new JScrollPane(poliPanel,ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
add(sp,BorderLayout.EAST);
```

- Pannello Risultato (risPanel)

Il pannello risPanel è volto a contenere una JTextArea 10x30 (risultato) che conterrà il risultato delle operazioni, settando un font più adatto e un testo iniziale che fornisce le indicazioni del programma.

Siccome al settaggio di un nuovo testo alla JTextArea essa stessa si ridimensionava aumentando la sua grandezza, ho ovviato a questo problema creando uno JScrollPane sp2 senza scorrimenti ed aggiungendolo alla JTextArea.



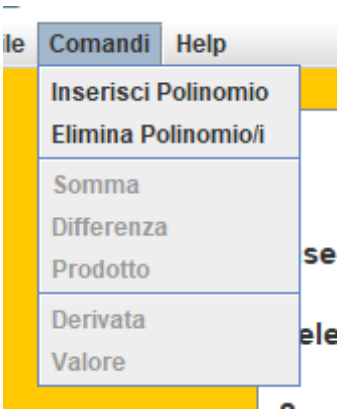
```
//Pannello dedicato al risultato
risPanel = new JPanel();
risPanel.setLayout(new FlowLayout(FlowLayout.CENTER,10,20));
risPanel.setBackground(Color.ORANGE);
add(risPanel,BorderLayout.CENTER);
risultato = new JTextArea(10,30);
//Inserisco una JScrollPane alla textarea in modo da non farla espadere all'aggiunta di
JScrollPane sp2=new JScrollPane(risultato,ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER,
        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
sp2.setPreferredSize(new Dimension(350,200));
risultato.setFont(new Font("Italic", Font.BOLD, 15));
risultato.append("\n                                    CHE FARE ?"
        + "\n\n -Inserisci Polinomi (menù comandi)\n\n -Seleziona uno o due Polinomi----
risultato.setEditable(false);
risPanel.add(sp2);
```

- Gestione Menù

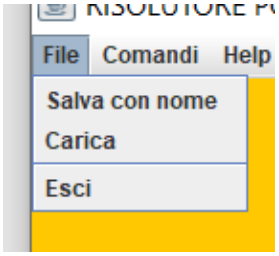
Ora mi occupo della creazione di un JMenuBar (m) che una volta aggiunto alla finestra lo riempio con i JMenu menuFile, menucomandi, menuhelp. Successivamente ad ogni JMenu aggiungo i JMenuItem dotandoli di un ascoltatore.

```
//Gestione Menù
JMenuBar m = new JMenuBar();
setJMenuBar(m);
```

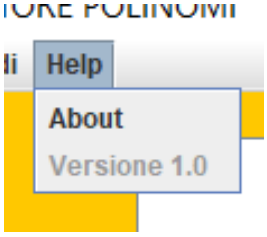
```
//IN COMANDI
menucomandi = new JMenu("Comandi"); m.add(menucomandi);
inserisci = new JMenuItem("Inserisci Polinomio");menucomandi.add(inserisci);
elimina= new JMenuItem("Elimina Polinomio/i");menucomandi.add(elimina);
menucomandi.addSeparator();
somma= new JMenuItem("Somma");menucomandi.add(somma);
comandiPolinomi.add(somma);
differenza = new JMenuItem("Differenza");menucomandi.add(differenza);
comandiPolinomi.add(differenza);
prodotto = new JMenuItem("Prodotto");menucomandi.add(prodotto);
comandiPolinomi.add(prodotto);
menucomandi.addSeparator();
derivata = new JMenuItem("Derivata");menucomandi.add(derivata);
comandiPolinomi.add(derivata);
valore = new JMenuItem("Valore");menucomandi.add(valore);
comandiPolinomi.add(valore);
inserisci.addActionListener(ascoltatore);
elimina.addActionListener(ascoltatore);
for(JMenuItem jmi:comandiPolinomi) {
    jmi.addActionListener(ascoltatore);
}
```



```
//IN FILE
menuFile = new JMenu("File"); m.add(menuFile);
salvaconnome = new JMenuItem("Salva con nome"); menuFile.add(salvaconnome);
carica = new JMenuItem("Carica"); menuFile.add(carica);
menuFile.addSeparator();
esci = new JMenuItem("Esci"); menuFile.add(esci);
salvaconnome.addActionListener(ascoltatore);
carica.addActionListener(ascoltatore);
esci.addActionListener(ascoltatore);
```




```
//ABOUT
menuhelp = new JMenu("Help"); m.add(menuhelp);
about=new JMenuItem("About");menuhelp.add(about);
about.addActionListener(ascoltatore);
version=new JMenuItem("Versione 1.0");menuhelp.add(version);//Scopo decorativo
version.setEnabled(false);
```



Il costruttore finisce con la chiamata del metodo nascondiComandi() che disabilità tutti i jMenuItem relativi ai comandi (In quanto non abbiamo inserito alcun polinomio all'inizio).

Metodo nascondiComandi

Il metodo nascondi comandi una volta invocato disabilita tutti i JMenuItem relativi alle operazioni che si possono fare sui polinomi:

```
//Nascondi tutti i comandi
private void nascondiComandi() {
    somma.setEnabled(false);
    differenza.setEnabled(false);
    prodotto.setEnabled(false);
    derivata.setEnabled(false);
    valore.setEnabled(false);
}
```

Metodo comandiPolinomio

Il metodo comandiPolinomio(boolean visibili) è finalizzato ad abilitare i JMenuItem relativi ad operazioni che si possono fare con due polinomi (somma,differenza,prodotto) e a disabilitare quelli associati ad operazioni eseguibili solo su un singolo polinomio (derivata,valore) se gli si passa true, viceversa altrimenti.

```
private void comandiPolinomio(boolean visibili) {
    if(visibili) {
        somma.setEnabled(true);
        differenza.setEnabled(true);
        prodotto.setEnabled(true);
        derivata.setEnabled(false);
        valore.setEnabled(false);
    }
    else {
        somma.setEnabled(false);
        differenza.setEnabled(false);
        prodotto.setEnabled(false);
        derivata.setEnabled(true);
        valore.setEnabled(true);
    }
}
```

Metodo creaPolinomio

Il metodo creaPolinomio riceve come parametro una stringa (quella passata dall'utente durante l'inserimento del polinomio nell'InputDialog).

Ho creato uno StringTokenizer che riceve la stringa (input) e ha come delimitatori gli operatori "+ e –" (i quali li prendo in considerazione) volto a ottenere ogni singolo monomio del polinomio passato in input. Infatti, dichiaro un oggetto Monomio (monomio), una stringa (segno) e creo l'oggetto Polinomio (poli) al quale aggiungo ogni singolo monomio.

Finché quindi ci sono tokens acquisisco il token con nextToken() e se è un operatore gestisco il segno, facendo poi un altro nextToken() il quale sarà sicuramente un monomio procedendo a crearlo invocando il metodo creaMonomio(token). Se l'operatore è il segno meno aggiungo il monomio al polinomio cambiato di segno, altrimenti l'aggiungo inalterato.

Se il primo token non si trattasse di un operatore, sarebbe sicuro (se l'utente non ci ha passato qualcosa di fantasioso) che fosse un monomio e pertanto procedo alla creazione dello stesso aggiungendolo al polinomio. (In questo else ricadiamo solo se l'utente ci ha passato un polinomio privo di un operatore iniziale entrandoci una sola volta, poiché qualora ci fossero altri tokens avremmo sicuramente operatori e quindi ricadremmo nell'if di prima).

La verifica che un polinomio passato sia corretto la eseguo in seguito nell'actionPerformed dell'ascoltatore.

```
private void creaPolinomio(String input) {
    StringTokenizer st= new StringTokenizer(input,"+-",true);
    String segno;
    Monomio monomio;
    String token;
    Polinomio poli=new PolinomioAL();
    while(st.hasMoreTokens()) {
        token =st.nextToken();
        if(token.equals("-")||token.equals("+")) {
            segno =token;
            token = st.nextToken();
            monomio = creaMonomio(token);
            if(segno.equals("-")) poli.add(monomio.mul(-1));
            else poli.add(monomio);
        }
        else {
            monomio = creaMonomio(token);
            poli.add(monomio);
        }

    }
```

```
    JCheckBox polinomioCheck = new JCheckBox(poli.toString());
    polinomioCheck.addActionListener(ascoltatore);
    polinomioCheck.setBackground(null);
    listaCheck.add(polinomioCheck);
    poliPanel.add(polinomioCheck);
    polinomioCheck.setSize(100,20);
    poliPanel.validate();
    validate();
    listaPolinomi.add(poli);

}//creaPolinomio
```

Concludo il metodo creando una JCheckBox (polinomioCheck) associandogli un ascoltatore e l'aggiungo alla lista dei JCheckBox (listaCheck). Rendo inoltre visibile lo stesso a video invocando add(polinomioCheck) sul pannello (poliPanel), e inserisco l'oggetto Polinomio (poli) nella lista dei polinomi. Chiamo il metodo validate() sia sul pannello che sulla finestra altrimenti il check non risulta visibile.

Metodo creaMonomio

Il metodo creaMonomio riceve una stringa (monomio) e ha la funzione di creare un oggetto monomio.

Creo anche qua uno StringTokenizer che riceve oltre che la stringa i delimitatori [x,X^] i quali li considero come tokens.

Inizializzo "di default" le variabili coefficiente (coef) ad 1 e grado (grad) a 0 e dichiaro il primo token (token).

```
private Monomio creaMonomio(String monomio) {
    StringTokenizer st = new StringTokenizer(monomio,"xX^",true);
    String token;
    int coef=1;
    int grad=0;
```

Un monomio può presentarsi in 5 forme diverse (considero come esempio un coef=3 e grad=2):

1. 3 (grado pari a 0)
2. X (coefficiente e grado pari a 1)
3. 3X (coefficiente diverso da 1 e grado pari a 1)
4. 3X^2 (coefficiente diverso da 1 con grado diverso da 1 e 0)
5. X^2 (coefficiente pari a 1 + grado diverso da 1 e 0)

Non è possibile aggiungere un Monomio pari a 0. (che senso avrebbe?)

Preso il primo token faccio un controllo.

- Se si tratta di una X ci riduciamo ai casi 2 e 5. Ora se vi è un altro token siamo sicuramente nel caso 5 (con st.nextToken() mi posiziono sull'apice"^" e setto il grado con l'intero del token successivo). Se così non fosse siamo nel caso 2 lasciando quindi il coefficiente "di default" (1) e settando il grado a 1.

```
    if(token.equalsIgnoreCase("x")) {
        if(st.hasMoreTokens()) {
            st.nextToken();
            grad=Integer.parseInt(st.nextToken());
        }
        else {grad=1;break;}
    }
```

- Se il primo token non fosse stata una X saremmo nei casi 1, 3 e 4:

- Se c'è un altro token vol dire che c'è sicuramente l'incognita X (caso 3 e 4) e ci resta da verificare quindi se sia di grado 1 (caso 3) o maggiore di uno (caso 4). Per fare ciò ci spostiamo sulla X (chiamando nextToken()) e se ha un ulteriore token siamo nel caso 4 (impostando a coef il valore del primo token [token] (quindi il coefficiente) e una volta spostati sull'apice, setto il grado con il valore del token successivo a "^".

```
if(st.hasMoreTokens()) {
    st.nextToken();
    if(st.hasMoreTokens()) {
        st.nextToken(); //^
        coef=Integer.parseInt(token);grad=Integer.parseInt(st.nextToken());break;
    }
    else { coef=Integer.parseInt(token);grad=1;}
}
```

- Se non ci fosse un altro token allora siamo nel primo caso e quindi procedo a settare il coefficiente (coef) con il valore del primo token (token) e lasciando il grado "di default" (0).

```
else coef=Integer.parseInt(token);
```

Per concludere il metodo creo un oggetto Monomio di coefficiente coef e grado grad ritornandolo.

```
    Monomio m = new Monomio(coef,grad);
    return m;
}//creaMonomio
```

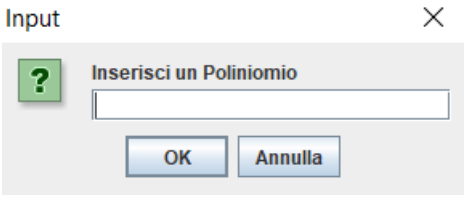
Classe AscoltatoreEventi

La classe AscoltatoreEventi implementa l'ActionListener e implemento l'unico metodo actionPerformed dell'interfaccia.

All'inizio dichiaro la variabile intera premuto dove mi salverò l'indice della posizione del JMenuItem relativo ad un'operazione scelta (pressione sul tasto) dall'utente all'interno dell'ArrayList comandiPolinomi (utile per quanto detto all'inizio della relazione, consentendomi in base all'indice di eseguire l'operazione opportuna).

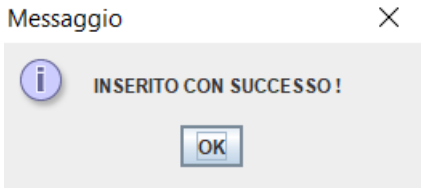
- Pressione inserisci

Alla pressione del JMenuItem inserisci faccio visionare uno showInputDialog chiedendo all'utente di inserire un polinomio all'interno di un ciclo infinito. Se preme su annulla fermo il ciclo e la finestra si chiude.

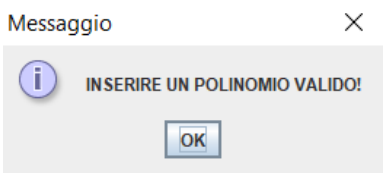



```
if(e.getSource()==inserisci) {
    for(;;) {
        input = JOptionPane.showInputDialog("Inserisci un Poliniomio");
        if(input==null) break; //pressione annulla
        if(input.matches(POLINOMIO)) {
            JOptionPane.showMessageDialog(null, "INSERITO CON SUCCESSO !");
            creaPolinomio(input);
        }
        else
            JOptionPane.showMessageDialog(null, "INSERIRE UN POLINOMIO VALIDO!");
    }
}//e==inserisci
```

Se la stringa passata matcha con l'espressione regolare POLINOMIO dichiarata all'inizio avviso l'utente che l'inseriento è avvenuto con successo e invoco il metodo creaPolinomio argomentato in precedenza.



Altrimenti avviso l'utente di inserire un polinomio valido e il ciclo continua.



- Gestione JMenuItem dei comandi e primo check spuntato

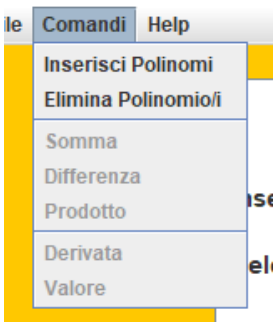
Quando premo su un JCheckBox se lo sto selezionando lo aggiungo all'ArrayList first (che mi aiuta a capire qual è il primo check selezionato il quale si troverà nella prima posizione della collezione) ed inoltre aumento il contatore countSelezionati.

Se lo sto deselezionando rimuovo tale CheckBox dalla collezione e diminuisco il contatore.

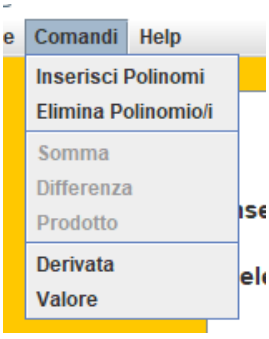
Gestito uno dei due casi verifico il numero dei JCheckBox selezionati stando attento ad abilitare i JMenuItem relativi alle operazioni che riguardano più di un polinomio e disabilitando valore e derivata effettuabili solo con un singolo polinomio se è almeno pari a 2, in caso contrario eseguo la cosa inversa.

Se il numero di JCheckBox selezionati torna a zero nascondo tutti i comandi.

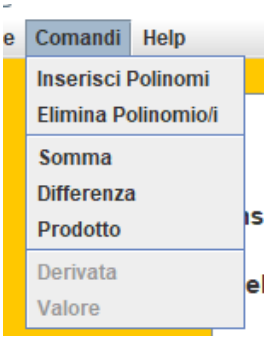
```
for(JCheckBox c:listaCheck) {
    if(e.getSource()==c) {
        if(c.isSelected()) { first.add(listaCheck.indexOf((c)));countSelezionati++;}
        else {first.remove(first.indexOf(listaCheck.indexOf((c)))); countSelezionati--;}
        if(countSelezionati>=2) comandiPolinomio(true);
        else if(countSelezionati==1) comandiPolinomio(false);
        else nascondiComandi();
    }
}
```



countSelezionati==0



countSelezionati==1

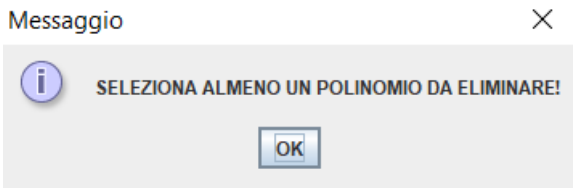


countSelezionati>=2

- Pressione elimina

Se l'utente esegue una pressione sul JMenuItem elimina, il programma reagisce cancellando tutti i JCheckBox selezionati ed internamente rimuove lo stesso dalla lista dei JCheckBox listaCheck e rimuove l'oggetto Polinomio associato al checkbox dalla collezione listaPolinomi.

Ho inizializzato a false una variabile booleana eliminabile, che se l'utente non avesse selezionato alcun JCheckBox non viene impostata a true e quindi il processo di eliminazione non avviene e mostra un MessageDialog avvisando l'utente di selezionare almeno un polinomio.




```
if(e.getSource()==elimina) {
    boolean eliminabile=false;
    //Inserisco i jcheckbox da eliminare in una collezione e i relativi polinomi per ovviare al Concurren
    ArrayList<JCheckBox> checkToDelete = new ArrayList<>();
    ArrayList<Polinomio> poliToDelete = new ArrayList<>();
    for(JCheckBox c:listaCheck)
        if(c.isSelected()) {
            eliminabile = true;
            c.setVisible(false);
            //listaPolinomi.remove(listaCheck.indexOf(c));
            //listaCheck.remove(c);
            poliToDelete.add(listaPolinomi.get(listaCheck.indexOf(c)));
            checkToDelete.add(c);
            countSelezionati--;
        }
    for(JCheckBox j:checkToDelete) listaCheck.remove(j);
    for(Polinomio p:poliToDelete) listaPolinomi.remove(p);
    if(!eliminabile) JOptionPane.showMessageDialog(null, "SELEZIONA ALMENO UN POLINOMIO DA ELIMINARE!");
    nascondiComandi();
}//e=elimina
```

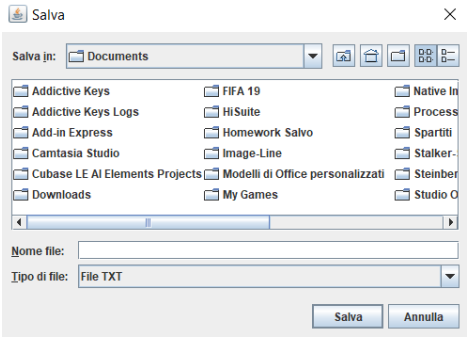
Ho usufruito di due ArrayList per ovviare all'eccezione ConcurrentModificationException sollevata se modifico la collezione durante uno scorrimento.

- Pressione salvaconnome e carica

Siamo giunti alla gestione del salvataggio/caricamento su/da file.

Alla pressione di uno dei due JMenuItem inizializzo a null una variabile stringa nome file, che sarà poi il percorso del file in questione, creo un JFileChooser (jfc) consentendo di lavorare solo con file di tipo txt (per semplicità, in generale con file di testo).

Una volta scelto il file, se stiamo salvando dichiaro un PrintWriter bufferizzato e scrivo su file l'elenco dei polinomi presi dalla collezione listaPolinomi; mostrando all'utente con uno showMessageDialog il percorso del file in cui è avvenuto il salvataggio.



L'ho racchiuso in un try-catch poiché può sorgere l'eccezione IOException che deriva da Exception in caso di file corrotto.

```
if(e.getSource()==salvaconnome) {

    PrintWriter pw;
    try {
        pw = new PrintWriter(new BufferedWriter(new FileWriter(nomefile)));
        for(Polinomio p:listaPolinomi) {
            pw.println(p.toString());
        }
        pw.close();
        JOptionPane.showMessageDialog(null,"Hai salvato il tuo operato sul file: " + nomefile);
    } catch (IOException e1) {
        JOptionPane.showMessageDialog(null,"Qualcosa è andato storto..");
        e1.printStackTrace();
    }
}
```

Se invece siamo in fase di caricamento inizialmente resetto il programma: svuotando la listaPolinomi,la listaCheck,azzerando il contatore dei checkBox selezionati e graficamente nascondo i JMenuItem relativi al JMenu Comandi e tolgo i JCheckBox dal pannello poliPanel.

Una volta fatto ciò creo un BufferedReader e con un ciclo infinito leggo ogni linea del file e contemporaneamente invoco creaPolinomio passandogli la linea svolgendo tutte le operazioni spiegate in precedenza nella sezione del metodo. Una volta finito il file (linea==null) fermo il ciclo e chiudo il BufferedReader. Anche qui il tutto è racchiuso da un try-catch per lo stesso motivo del salvataggio.

```
else {
    listaPolinomi.clear();
    for(JCheckBox jc:listaCheck) {
        jc.setVisible(false);
    }
    listaCheck.clear();
    countSelezionati=0;
    nascondiComandi();
    try {
        BufferedReader br = new BufferedReader(new FileReader(nomefile));
        String linea=null;
        for(;;) {
            linea=br.readLine();
            if(linea==null) break;
            creaPolinomio(linea);
        }
        br.close();
    } catch (IOException e1) {
        JOptionPane.showMessageDialog(null,"Qualcosa è andato storto.. Il file potrebbe essere co
        e1.printStackTrace();
    }
}
```

- Pressione di un Comando

Qui gestisco gli eventi generati dalla pressione dei vari JMenuItem del JMenu Comandi:

Come ho anticipato in precedenza verifico quale JMenuItem è stato a sollevare l'evento scorrendo la collezione comandiPolinomi, evitando di scrivere stesse righe di codice per ogni JMenuItem.

Ho deciso che è possibile effettuare operazioni massimo con due polinomi, pertanto se l'utente ha selezionato almeno 3 jcheckbox avviso l'utente che ciò non è possibile.

Se l'utente ha selezionato in modo corretto i JCheckBox inizializzo inizialmente una variabile booleana sommato che mi servirà poi durante l'operazione prodotto; e creo un nuovo Polinomio.

Con un ciclo for vedo quali JCheckBox sono selezionati.

Nota: gli indici delle posizioni di un polinomio e del suo JCheckbox nelle loro apposite collezioni sono gli stessi.

- Se l'utente ha selezionato di fare la **somma** basta aggiungere al polinomio ris il jCheckBox selezionato (saranno quindi aggiunti i due polinomi selezionati e la classe Polinomio si preoccupa di sommare i monomi simili).

```
if(e.getSource()==somma) {
    ris=ris.add(listaPolinomi.get(listaCheck.indexOf(jc)));
}
```

- Se l'operazione richiesta è la **differenza** devo stare attento quale JCheckBox è stato premuto per primo (e lo faccio controllando se il jCheckbox corrente è lo stesso di quello presente in prima posizione della collezione first) il quale lo aggiungo così come si presenta al polinomio ris e quando siamo di fronte al secondo ad essere stato premuto (se non è il primo elemento della collezione) devo aggiungerlo cambiato di segno, facendo sì che risulti essere il secondo operando.

```
else if(e.getSource()==differenza) {
    if(listaCheck.indexOf(jc)==first.get(0)){
        ris=ris.add(listaPolinomi.get(first.get(0))); //perchè ritorna un monomio
    }
    else{
        for(Monomio m:listaPolinomi.get(listaCheck.indexOf(jc))) {
            m=m.mul(-1); //perchè ritorna un monomio
            ris.add(m);
        }
    }
}
```

- Di fronte all'operazione **prodotto** la questione è leggermente diversa. Appena incontriamo un JChecBox selezionato dobbiamo aggiungerlo al polinomio (quindi controllo con un flag (sommato) se ciò non è già avvenuto) e incontrato il secondo JCheckBox (selezionato) moltiplico il polinomio ris con esso.

```
else if(e.getSource()==prodotto) {
    if(!sommato) {
        ris=ris.add(listaPolinomi.get(listaCheck.indexOf(jc)));
        sommato=true;
    }
    else ris=ris.mul(listaPolinomi.get(listaCheck.indexOf(jc)));
}
```

- Da adesso in poi i JCheckBox selezionati sarà al massimo uno solo. Alla pressione dell'operazione **derivata**, invoco il metodo derivata() della classe Polinomio sul polinomio corrispondente al JCheckBox selezionato e aggiungo il risultato a ris.

```
else if(e.getSource()==derivata) {
    ris=ris.add(listaPolinomi.get(listaCheck.indexOf(jc))).derivata();
}
```

- A questo punto rimane solo l'operazione "**valore**" che passato un double ci fornisce quanto vale il polinomio una volta sostituito all'incognita X. L'input lo ricevo mostrando un InputDialog col quale chiedo all'utente di inserire il valore e in caso di non matching con l'espressione regolare che identifica un double (DBL) avverto l'utente che il valore non è valido. Per mantenere continuità inserisco il tutto in un ciclo infinito e mi preoccupo nello stesso di far comparire nella JTextArea il risultato, solo se il matching è andato a buon fine.

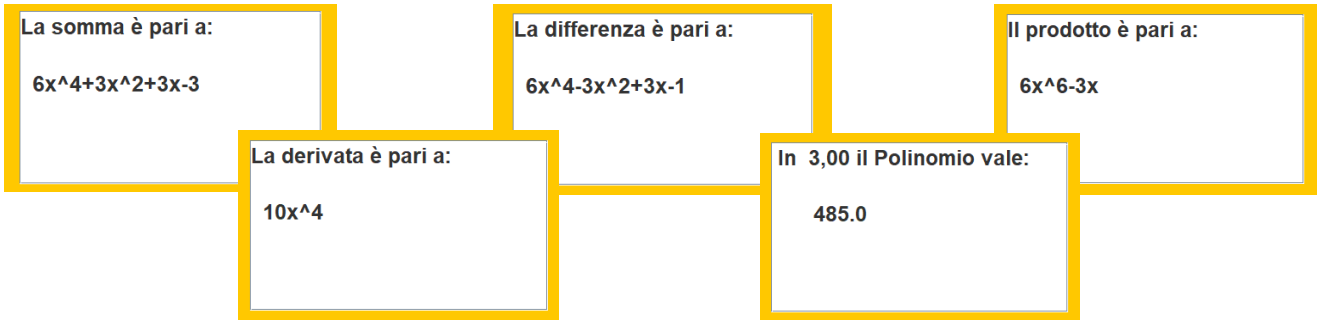
```
else if(e.getSource()==valore) {
    risultato.setText(null);
    risultato.setFont(new Font("Italic", Font.BOLD, 25));
    for(;;) {
        String x= JOptionPane.showInputDialog("Inserisci il valore:");
        if(x.matches(DBL)) {
            risultato.append(" In "+String.format("%5.2f", Double.parseDouble(x))+" il Polinomio vale:"
                    + "\n\n       "+(listaPolinomi.get(listaCheck.indexOf(jc)).valore(Double.parseDouble(x))));
            break;
        }
        else JOptionPane.showMessageDialog(risPanel, "Inserisci un Valore valido per favore..");
    }
```

Infine deseleziono i JCheckBox, decremento il conteggio di quelli selezionati e disabilito i comandi.

- Gestione Soluzioni

Mi salvo l'indice del comando all'interno della collezione comandiPolinomi nella variabile premuto in modo da scrivere la soluzione sulla JTextArea a seconda del comando premuto, eseguendo uno switch-case.

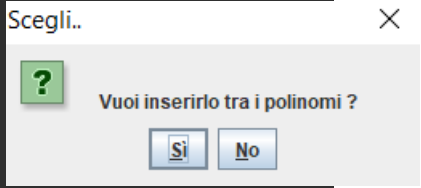
```
if(!(e.getSource()==valore)) {
    risultato.setText(null);
    risultato.setFont(new Font("Italic", Font.BOLD, 25));
    switch(premuto) {
    case 0: risultato.append("La somma è pari a:\n\n  "+ris.toString());break;
    case 1: risultato.append("La differenza è pari a:\n\n  "+ris.toString());break;
    case 2: risultato.append("Il prodotto è pari a:\n\n  "+ris.toString());break;
    case 3: risultato.append("La derivata è pari a:\n\n  "+ris.toString());break;
    }
```



Una volta mandato a video il risultato chiedo all'utente di salvare lo stesso nel pannello polinomi.

```
do {
    i=JOptionPane.showConfirmDialog(poliPanel,"\n Vuoi inserirlo tra i polinomi ?","Scegli..",JOptionPane.YES_NO_OPTION);
    if(i==JOptionPane.YES_OPTION) {
        JCheckBox nuovo = new JCheckBox(ris.toString());
        poliPanel.add(nuovo);
        nuovo.setSize(100,20);
        poliPanel.validate();
        validate();
        derivata.validate();
        listaPolinomi.add(ris);
        listaCheck.add(nuovo);
        nuovo.setBackground(null);
        nuovo.addActionListener(this);

    }
    else if(i==JOptionPane.CLOSED_OPTION) JOptionPane.showMessageDialog(null, "Dai dimmi cosa fare..");
}while(!(i==JOptionPane.YES_OPTION || i==JOptionPane.NO_OPTION));
```



- Pressione About e classe FinestraAbout

Alla pressione del JMenuItem about creo una finestra e la rendo visibile.

```
if(e.getSource()==about) {
    JFrame ab = new FinestraAbout();
    ab.setVisible(true);
}
```

Gestite le dimensioni e la sua posizione (i bounds) gli aggiungo un pannello Pabout volto a contenere una JTextArea nel quale illustro il funzionamento dell'applicazione:

```
class FinestraAbout extends JFrame{
    public FinestraAbout() {
        setTitle("Help");
        setSize(430,280);
        setLocation(710,300);
        setResizable(false);
        JPanel Pabout = new JPanel();
        add(Pabout);
        JTextArea txtA = new JTextArea();
        txtA.setBackground(Color.LIGHT_GRAY);
        txtA.setFont(new Font("Italic",Font.BOLD, 15));
        txtA.setPreferredSize(new Dimension(420,260));
        Pabout.add(txtA);
        txtA.setEditable(false);
        txtA.append("Questo programma svolge le operazioni base tra polinomi:\n"
                + "quali somma,differenza,moltiplicazione,derivata \ne ricerca del suo valore passatogli un numero (double). \n"
                + "\n\nProgramma sviluppato da Marco Greco, matricola 200901.\n\nUnical \nIngegneria Informatica ");

    }
}//FinestraAbout
```