DEBUGGING

DEBUGGING

- Attività di ricerca e correzione dei difetti che sono causa di malfunzionamenti.
- É l'attività consequenziale alla scoperta di un malfunzionamento. Comprende due fasi:
  - **Ricerca del difetto**
  - **Correzione del difetto**
- Il debugging è ben lungi dall'essere stato formalizzato
  - Metodologie e tecniche di debugging rappresentano soprattutto un elemento dell'esperienza del programmatore/tester

RICERCA E LOCALIZZAZIONE DEI DIFETTI

- Ridurre la distanza tra difetto e malfunzionamento
  - Mantenendo un'immagine dello stato del processo in esecuzione in corrispondenza dell'esecuzione di specifiche istruzioni
    - Watch point e variabili di watch (Sonde)
      - Un watch, in generale, è una semplice istruzione che inoltra il valore di una variabile verso un canale di output
        - L'inserimento di un watch (sonda) è un'operazione invasiva nel codice: anche nel watch potrebbe annidarsi un difetto
        - In particolare l'inserimento di sonde potrebbe modificare sensibilmente il comportamento di un software concorrente
    - Asserzioni, espressioni booleane dipendenti da uno o più valori di variabili legate allo stato dell'esecuzione
      - Possiamo realizzare una asserzione con una interruzione programmata

ESEMPIO DI SONDA

- Con le due System.out.println verifichiamo il capitale prima e dopo l'acquisto
- Avremmo potuto scrivere su di un file anzichè sullo schermo per poter avere alla fine un *log* di informazioni relative all'esecuzione

```
public boolean acquista(String nomeSocieta, int quantita) {
   boolean ok=true;
   for (Societa s:l.getSocieta())
      if (s.getNome().contentEquals(nomeSocieta)) {
         if (ok) {
            System.out.println("Capitale attuale : "+ g.getCapitale());
            g.acquista(quantita, LocalDate.now(), s.getPrezzoAzione(), s);
            System.out.println("Capitale dopo l'acquisto : "+ g.getCapitale());
            return true;
         }
      }
   return false;
}
```

ESEMPIO DI ASSERZIONE

- Con questo controllo vigiliamo che la liquidità non diventi negative e in tal caso blocchiamo il programma in modo da sapere da che punto in poi le cose vanno male

```
public boolean acquista(String nomeSocieta, int quantita) {
   boolean ok=true;
   for (Societa s:l.getSocieta())
      if (s.getNome().contentEquals(nomeSocieta)) {
         if (ok) {
            g.acquista(quantita, LocalDate.now(), s.getPrezzoAzione(), s);
         if(g.getLiquidita()<0)

            throw new RuntimeException("Errore: non avevi soldi per questo acquisto");

          return true;
         }
      }
   return false;

}
```

AUTOMATIZZAZIONE DEL DEBUGGING

- Il debugging è un'attività estremamente intuitiva, che però deve essere operata nell'ambito dell'ambiente di sviluppo e di esecuzione del codice
- Strumenti a supporto del debugging sono quindi convenientemente integrati nelle piattaforme di sviluppo (IDE), in modo da poter accedere ai dati del programma, anche durante la sua esecuzione, senza essere invasivi rispetto al codice
  - In assenza di ambienti di sviluppo, l'inserimento di codice di debugging invasivo rimane l'unica alternativa

FUNZIONALITÀ DI DEBUGGING

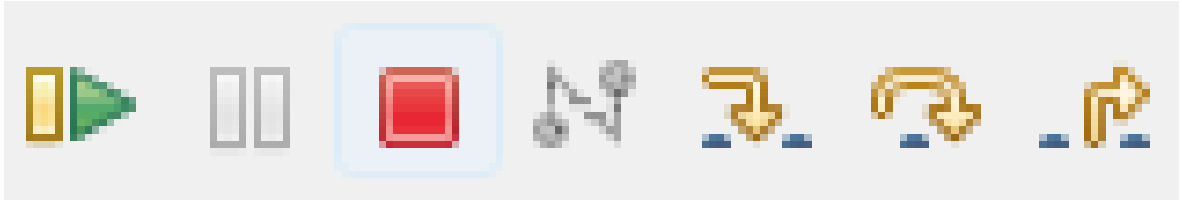
- Inserimento break point
  - Tramite Eclipse è possibile accedere a delle "Breakpoint properties"
    - Breakpoint condizionali:
      - Breakpoint che si attivano solo quando si passa in una certa riga e si verifica una certa condizione
    - Breakpoint dipendenti dallo Hit Count:
      - Breakpoint che si attivano solo dopo un certo numero di passaggi su quella riga di codice

- Esecuzione passo passo del codice
  - Entrando o meno all'interno dei metodi chiamati
  - Uscendo da un metodo verso il chiamante

ESECUZIONE STEP BY STEP

- Quando il programma si ferma su di un breakpoint è possible:
  - Farlo continuare (pulsante play)
  - Proseguire all'interno della funzione chiamata nella riga attuale
  - Proseguire con la riga successiva
  - Proseguire fino alla fine dell'esecuzione della funzione corrente

- Durante l'esecuzione step by step possiamo vedere facilmente i valori di tutte le variabili e chiedere di calcolare eventuali espressioni

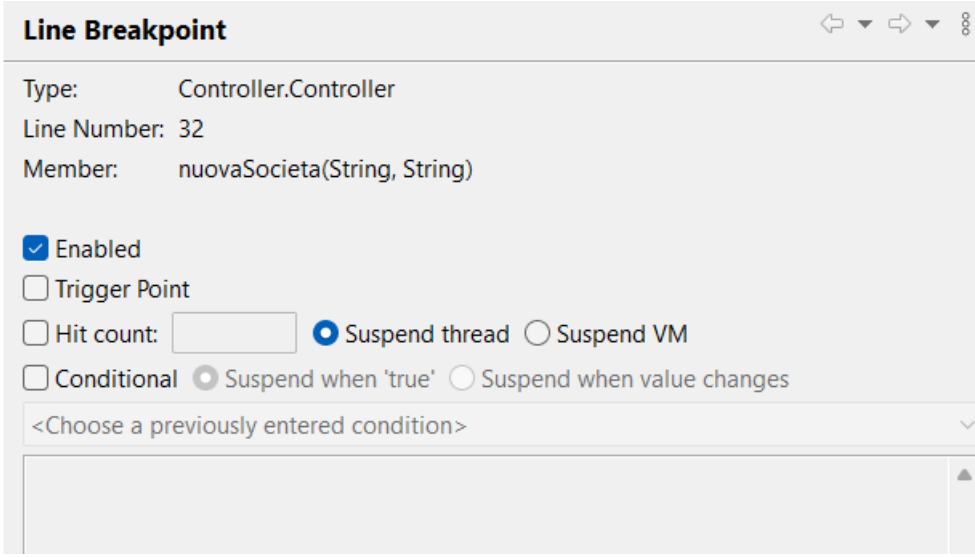


Variables | Breakpoints | Expressions

Name	Value
no method return value	
this	Controller (id=44)
nomeSocieta	"uno" (id=73)
prezzoAzione	"10" (id=79)

BREAKPOINT CONDIZIONALI

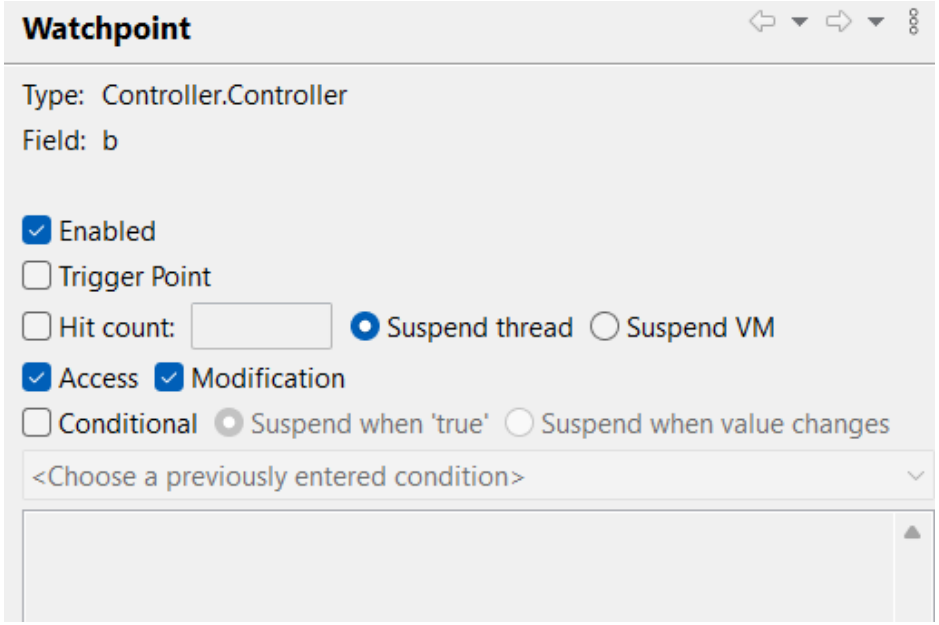
- Per abbreviare il debugging dovremmo poter sistemare il breakpoint il più vicino possible alla *causa* del difetto
- Nel dubbio, c'è il rischio di dover eseguire trope righe prima di arrivare a quella incriminate
  - Ad esempio se c'è un ciclo ripetuto 1000 volte, come possiamo fare a fermarci dopo 789 esecuzioni senza eseguire altrettante volte il ciclo?
- Breakpoint condizionali
  - Si attivano con il tasto destro su di un breakpoint e scegliendo Breakpoint properties



- Hit Count: il breakpoint ferma il programma solo quando si passa per la i-esima volta su di esso
- Conditional: ogni volta che passiamo sul breakpoint si valuta la condition: solo se è vero il programma si sospende

WATCHPOINT

- Un watchpoint è un breakpoint collegato non ad una riga ma ad un attributo di una classe
- Si inserisce come un breakpoint, puntando sulla riga di dichiarazione dell'attributo
- Fa sospendere l'esecuzione ogni volta che l'attributo è letto o scritto
- Tramite le Properties è possibile personalizzare quando effettuare la sospensione



ULTERIORI TIPI DI BREAKPOINT

- Similmente a breakpoint e watchpoint:
  - Breakpoint per le eccezioni;
  - Breakpoint per il caricamento di classi;
  - Breakpoint per un metodo;

- Ulteriori approfondimenti ed esempi:
  - http://www.vogella.com/articles/EclipseDebugging/article.html