# Ingegneria del Software 1: Debugging in Eclipse

Da completare entro Aprile 21, 2015

*Srđan Krstić and Marco Scavuzzo*

# Contents

# 1 Preliminaries

Il *debugging* (o semplicemente debug), in informatica, indica l'attivitá che consiste nell'individuazione della porzione di software affetta da errore (bug) rilevata nei software a seguito dell'utilizzo del programma.
Un *breakpoint* nel codice sorgente specifica dove l'esecuzione di un programma si deve fermare. Un volta raggiunto il breakpoint é possibile analizzare le variabili, cambiare il loro valore etc.
Un *watchpoint* é un break point su un campo. Il debugger si ferma quando il campo é letto o modificato

## 1.1 Fraction Class

```java
package org.ingsoft.debugging;

/**
 * A class representing a fraction of integer values. The class provides functionality
    for simplifying the fraction and performing the basic fraction computations.
 * @author Claudio
 */
public class Fraction {


      private int numerator;
      private int denominator;

      /**
       * Constructs a fraction with the specified numerator and denominator
       * @param numerator the numerator of the fraction
       * @param denominator the denominator of the fraction
       */
      public Fraction(int numerator, int denominator){
            this.numerator=numerator;
            this.denominator=denominator;
      }

      /**
       * Constructs a fraction with the specified numerator and a denominator of 1
       * @param numerator the numerator of the fraction
       */
      public Fraction(int numerator){
            this(numerator,1);
      }

      /**
       * @return the numerator
       */
      public int getNumerator() {
            return numerator;
      }
      /**
       * @param numerator the numerator to set
       */
      public void setNumerator(int numerator) {
            this.numerator = numerator;
```

```
	}
	/**
	 * @return the denominator
	 */
	public int getDenominator() {
		return denominator;
	}
	/**
	 * @param denominator the denominator to set
	 */
	public void setDenominator(int denominator) {
		this.denominator = denominator;
	}

	/**
	 * Add this fraction to the specified fraction and returns it as a new fraction
	    (not simplified).
	 * It does not modify this fraction.
	 * @param f the fraction to be added to this fraction
	 * @return a new fraction equivalent to this fraction plus the parameter
	 */
	public Fraction add(Fraction f)
	{
		int num= (this.numerator * f.denominator) + (this.denominator *
		    f.numerator);
		int den= this.denominator * f.denominator;
		Fraction sum=null;
		sum.setNumerator(num);
		sum.setDenominator(den);
		//Fraction sum=new Fraction(num, den);

		return sum;
	}

	@Override
	public String toString(){
		return this.numerator+"/"+this.denominator;
	}
}
```

## 1.2 Main

```
package org.ingsoft.debugging;

public class FractionMain {

	public static void main(String[] args) {
		Fraction f=new Fraction(3,4);
		Fraction g=new Fraction(5);
		Fraction[] myfractions=new Fraction[5];

		//add the fractions and store the result
```

```
            Fraction sum=f.add(g);

            myfractions[0]=f;
            myfractions[1]=g;
            myfractions[4]=sum;

            // Print the result
            System.out.println(myfractions[4].toString());
        }
}
```

# 2 Interfaccia di Debug (Debug perspective)

- *call stack* mostra le parti del codice che sono in esecuzione e come sono legate le une alle altre
- *Breakpoints*: mostra l'insieme dei break point del vostro programma. Consente di rimovere, attivare e disattivare un break point. È possibile anche disattivare tutti i breakpoint cliccando su "skip all breakpoints" (pallino azzurro barrato)
- *Variables* mostra attributi e variabili (cliccando sulla freccetta in alto a destra è possibile selezionare tra le altre
  - campi da mostrare (Java)
  - il tipo di ogni variabile (Layout)

# 3 Utilizzare il debugger

Prima di tutto assicuriamoci che il debugger consideri solo il "nostro" codice ed eliminiamo il debug di altro codice (e.g., librerie java). Cliccare su preferences > java debug > step filtering > use step filters > check everything > finish

## 3.1 Aggiungere rimuovere e disabilitare un break point su un istruzione

- Per *aggiungere* un break point è sufficiente fare doppio click sul bordo della riga dove vogliamo aggiungerlo. Il break point indica la riga sulla quale si desidera che la computazione venga fermata.
- Per *disabilitare* un break point cliccare con il tasto destro sul break point e premere "disable break point".
- Per *eliminare* un break point cliccare due volte su di esso.

## 3.2 Breakpoint properties

Dopo aver aggiunto un breakpoint puoi selezionare le proprietà del break point (per esempio è possibile specificare che un breakpoint deve divenire attivo dopo essere stato raggiunto 12 volte o quando una particolare condizione è verificata)

- *right click > breakpoint properties*

### 3.3 Aggiungere un break point su un metodo

È possibile aggiungere un break point su un metodo facendo doppio click sul margine sinistro del metodo. È possibile configurare se il debugger debba fermarsi quando all'uscita o all'entrata del metodo, agendo sulle proprietà.

### 3.4 Aggiungere un watchpoint

- posso settare un watch point cliccando due volte sul margine destro
- Posso configurare le proprietà per far in modo che l'esecuzione si fermi durante la lettura (Field access) o la modifica (Field Modification) di un campo, o entrambi.

### 3.5 Lanciare il debugger

- Per lanciare il debugger è sufficiente cliccare sul bug (insetto) presente a fianco del pulsante run o, alternativamente, andare su *run > debug*
- Una volta lanciato il debug, eclipse automaticamente ci porta nella debug perspective.
- la linea evidenziata in verde corrisponde alla linea di codice che sta per essere eseguita (nota che tale linea non è ancora stata eseguita).

### 3.6 Comandi di debugging

- *step into* consente di valutare le linee di codice che eseguono una data operazione. Per esempio se la linea di codice selezionata chiama un particolare metodo, cliccando su step into si va a valutare il codice contenuto in quel metodo
- *step over* esegue le line di codice ma non mostra il comportamento interno di quelle linee. (e.g., se viene chiamato un metodo salta il debugging delle linee di codice di quel metodo).
- *step return* permette di ritornare da un metodo nel quale e' stata eseguita una step into.
- *use step filters* se cliccato abilita l'utilizzo dei step filters (si consiglia di tenerlo abilitato per evitare di debuggare codice java nativo)
- *resume* viene utilizzato, quando sono inseriti piú break points, per andare al break point successivo

### 3.7 Cambiare il valore delle variabili

Per cambiare il valore a una variabile durante il debug è sufficiente cliccare all'interno del campo value (nota viene utilizzato il metodo toString() per mostrare il contenuto della variabile)

### 3.8 Espressioni

È possibile, durante il debug, valutare il valore di alcune espressioni (porzioni di codice). In particolare, due diverse azioni possono essere eseguite:

- selezionando un espressione per esempio this.numerator*f.denominator e cliccando su watch è possibile vedere il valore dell'espressione;
- si possono anche aggiungere nuove espressioni cliccando su add e aggiungendo una data espressione;
- si possono anche aggiungere altre espressioni (this.numerator*10).

Nota che e' anche possibile cambiare il valore delle variabili mentre il codice e' in esecuzione.