IL FRAMEWORK EMBASP

DOTT. SSA JESSICA ZANGARI – 11/12/2020

CORSO DI INTELLIGENZA ARTIFICIALE – CDS INFORMATICA – UNIVERSITA' DELLA CALABRIA

EMBASP

CHE COS'E'?

- EmbASP è un framework che permette di integrare ASP (ma anche altri linguaggi logici) in software realizzati in linguaggi di programmazione imperativi «classici»
- In particolare, ad oggi EmbASP è stato implementato in Java, C# e Python
- EmbASP è un progetto open-source del nostro Dipartimento – Sito web ufficiale

A COSA SERVE?

- Ci consente di invocare un sistema ASP in modo sincrono o a-sincrono all'interno di un software più complesso, consentendo quindi di realizzare alcune «parti» tramite ASP
- Tramite EmbASP possiamo realizzare applicazioni «Desktop» o «Mobile», in particolare Android
- Suggeriamo di usare EmbASP per realizzare i vostri progetti didattici

COSA BISOGNA INSTALLARE?

- EmbASP necessita di ANTLR4 (la versione consigliata è 4.7): è sufficiente scaricare la libreria (versione Java / C# / Python) dal sito ufficiale qui o in alternativa usare Gradle o Maven
- Consigliamo di utilizzare un IDE (come Eclipse, etc.) per realizzare il progetto

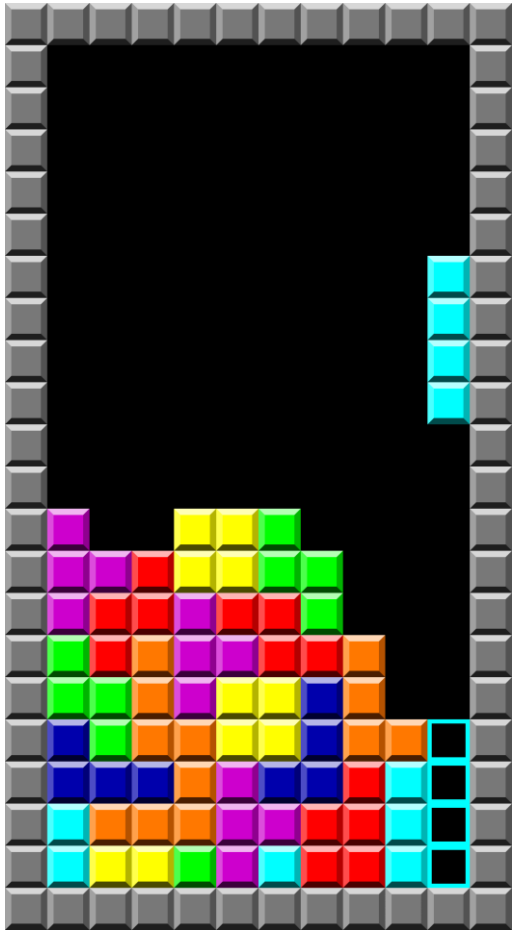
BUG REPORT

- Se notate comportamenti «strani» o inattesi, scrivete per email al team di sviluppo allegando possibilmente il codice completo su cui riscontrate il problema (fa comodo anche qualche screenshot): embasp@mat.unical.it

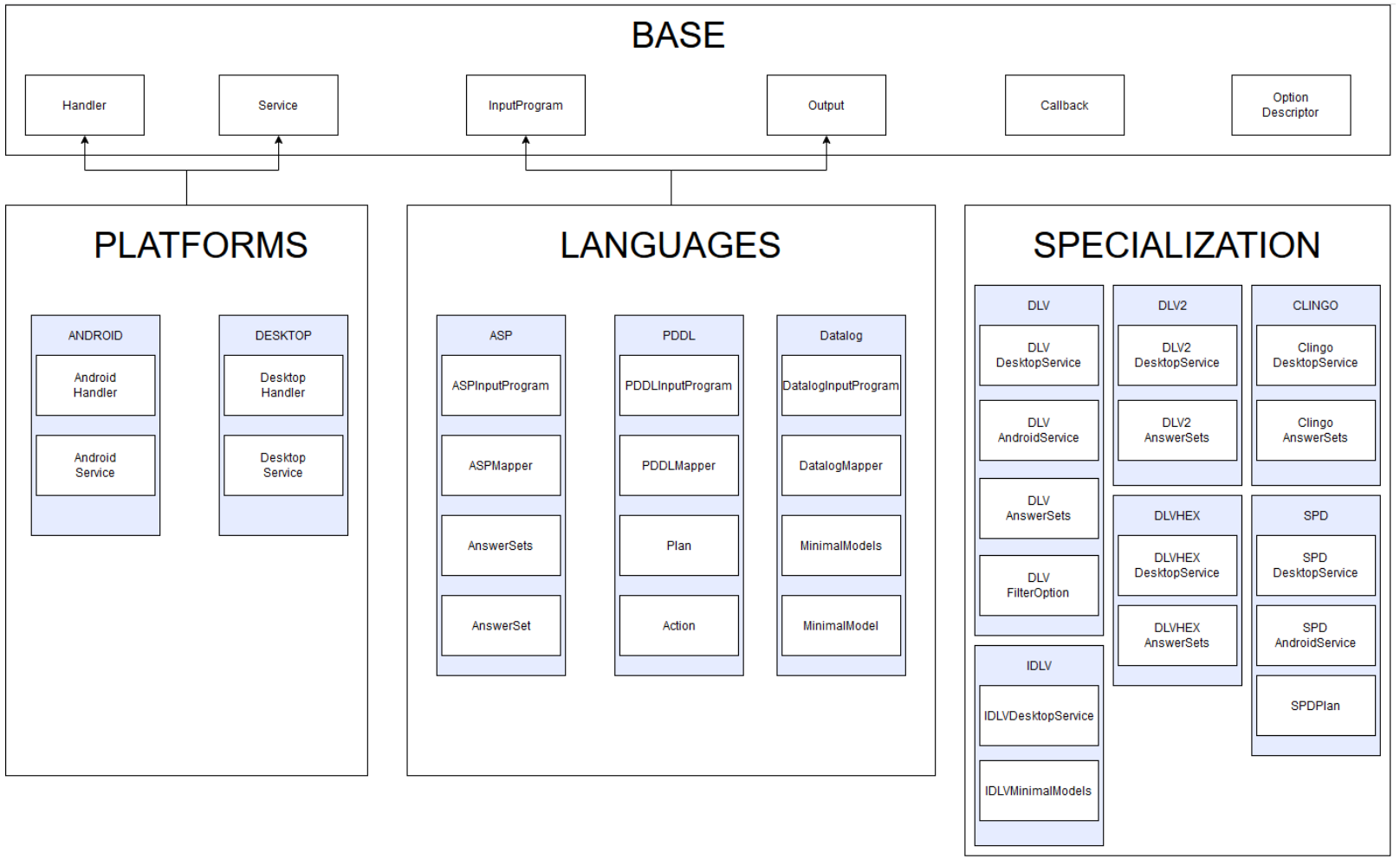
PERCHE' EMBASP?

Usare ASP all'interno di applicazioni:

1. Selezionare un sistema ASP
2. Creare un processo che esegue il sistema ASP selezionato
3. Convertire oggetti in fatti e fornirli al processo che esegue il sistema ASP
   - `Pezzo pezzoCorrente = new Pezzo("I")` → `pezzoCorrente("I")`
4. Fornire al processo che esegue il sistema ASP il programma logico
5. Raccogliere e analizzare il risultato dell'esecuzione del sistema ASP
6. Conversione del risultato in oggetti
   - `posizionePezzoCorrente(0,10)` → `griglia[0][10]= lightblue`
   - `posizionePezzoCorrente(1,10)` → `griglia[1][10]= lightblue`
   - `posizionePezzoCorrente(2,10)` → `griglia[2][10]= lightblue`
   - `posizionePezzoCorrente(3,10)` → `griglia[3][10]= lightblue`
7. Terminare il processo che ha eseguito il sistema ASP



ARCHITETTURA DI EMBASP



INFORMAZIONI E CURIOSITÀ SU EMBASP

- Sito ufficiale: https://www.mat.unical.it/calimeri/projects/embasp/
- Descrizione dettagliata ed esempi d'uso per diversi linguaggi e piattaforme: https://embasp.readthedocs.io/en/latest/
- Documentazione ufficiale in formato Doxygen: https://embasp.readthedocs.io/en/latest/documentation/technical_doc.html

DEMO: IL GIOCO DEL SUDOKU E IL PROBLEMA DELLA 3-COLORABILITÀ REALIZZATI TRAMITE LA VERSIONE JAVA DI EMBASP

ESEGUIRE LA DEMO SUDOKU

1. Scaricare il file zip contenente la demo Sudoku dal sito del corso
   - **NOTA**: la demo fa uso della versione 7.2.0 di EmbASP
2. Estrarre il file zip
3. Nella cartella lib sono presenti gli eseguibili di DLV2 per Windows e Linux 64bit aggiornati al 2 Novembre. Se si fa uso di un differente sistema operativo, aggiungere nella cartella lib la versione di DLV2 appropriata scaricabile dal sito del corso
   - **NOTA**: su Linux è necessario aggiungere i permessi di esecuzione. Entrate nella cartella lib e da terminale eseguire il comando: `chmod +xX dlv2`
   - **NOTA**: su Mac OS è necessario aggiungere i permessi di esecuzione (come nel caso Linux) e inoltre consentire l'esecuzione dell'eseguibile nonostante provenga da uno sviluppatore non identificato (scegliere «consenti comunque» in Preferenze → Sicurezza e Privacy)
   - **NOTA**: su Windows è necessario che l'eseguibile di DLV2 abbia estensione .exe (nella demo è già così)
4. Importare il progetto con il proprio IDE
   - Ad esempio con Eclipse, occorre:
     1. Selezionare File → Import → General → Project from Folder or Archive
     2. Nella finestra che fare click su "Directory…" e selezionare la cartella **interna** «SudokuEmbASP» ottenuta dall'estrazione del file zip della demo
     3. Fare click su Finish
     - **NOTA**: questa procedura potrebbe variare in base alla versione di Eclipse utilizzata

Continua …

ESEGUIRE LA DEMO SUDOKU

5. Indicare ad EmbASP l'eseguibile di DLV2 decommentando una tra le istruzioni che creano l'oggetto Handler in base al proprio sistema operativo
   - Su Windows 64 bit è sufficiente decommentare:

     ```
     handler = new DesktopHandler(new DLV2DesktopService("lib/dlv2.exe"));
     ```

   - Su Linux 64 bit è sufficiente decommentare:

     ```
     handler = new DesktopHandler(new DLV2DesktopService("lib/dlv2"));
     ```

   - Su MacOS 64 bit è sufficiente decommentare:

     ```
     handler = new DesktopHandler(new DLV2DesktopService("lib/dlv2-mac"));
     ```

   - Su altri sistemi operativi è necessario aggiungere alla cartella lib un appropriato eseguibile di DLV2 scaricabile dal sito del corso e specificare opportunamente il nome dell'eseguibile nel costruttore di `DLV2DesktopService`

ESEGUIRE LA DEMO 3COL

1. Scaricare il file zip contenente la demo 3Col dal sito del corso
   - **NOTA**: la demo fa uso della versione 7.2.0 di EmbASP
2. Estrarre il file zip
3. Nella cartella lib sono presenti gli eseguibili di DLV per Windows e Linux 64bit. Se si fa uso di un differente sistema operativo, aggiungere nella cartella lib la versione di DLV appropriata scaricabile dal sito del corso
   - **NOTA**: su Linux è necessario aggiungere i permessi di esecuzione. Entrate nella cartella lib e da terminale eseguire il comando:
     `chmod +xX dlv`
   - **NOTA**: su Mac OS è necessario aggiungere i permessi di esecuzione (come nel caso Linux) e inoltre consentire l'esecuzione dell'eseguibile nonostante provenga da uno sviluppatore non identificato (scegliere «consenti comunque» in Preferenze → Sicurezza e Privacy)
   - **NOTA**: su Windows è necessario che l'eseguibile di DLV abbia estensione .exe (nella demo è già così)
4. Importare il progetto con il proprio IDE
   - Ad esempio con Eclipse, occorre:
     1. Selezionare File → Import → General → Project from Folder or Archive
     2. Nella finestra che fare click su "Directory…" e selezionare la cartella **interna** «3ColEmbasp» ottenuta dall'estrazione del file zip della demo
     3. Fare click su Finish
     - **NOTA**: questa procedura potrebbe variare in base alla versione di Eclipse utilizzata

Continua …

ESEGUIRE LA DEMO 3COL

5. Indicare ad EmbASP l'eseguibile di DLV decommentando una tra le istruzioni che creano l'oggetto Handler in base al proprio sistema operativo
   - Su Windows 64 bit è sufficiente decommentare:

     ```
     handler = new DesktopHandler(new DLVDesktopService("lib/dlv.mingw.exe"));
     ```

   - Su Linux 64 bit è sufficiente decommentare:

     ```
     handler = new DesktopHandler(new DLVDesktopService("lib/dlv"));
     ```

   - Su altri sistemi operativi è necessario aggiungere alla cartella lib un appropriato eseguibile di DLV scaricabile dal sito del corso e specificare opportunamente il nome dell'eseguibile nel costruttore di `DLVDesktopService`
   - **NOTA**: la demo può essere facilmente riadattata per usare DLV2, usando un **DLV2DesktopService** al posto di **DLVDesktopService** e aggiungendo nella cartella lib un eseguibile di DLV2 appropriato in base al proprio sistema operativo

GUIDA PRATICA ALL'USO DI EMBASP PER IL PROGETTO

USARE EMBASP

- Le principali fasi da eseguire sono le seguenti:
  1. Scaricare EmbASP e predisporre il progetto
  2. Specificare l'eseguibile del sistema ASP che si intende utilizzare
  3. Specificare l'input per il sistema ASP (ovvero programma logico e fatti)
  4. Invocare il sistema ASP
  5. Analizzare gli eventuali answer set restituiti
- Nota: i punti 3 – 5 possono ripetersi più volte in base alle caratteristiche della propria applicazione
- Di seguito si mostra come usare la versione Java di EmbASP per realizzare un'applicazione Desktop tuttavia gli step da effettuare sono quelli su elencati anche nel caso in cui si volesse realizzare un'applicazione mobile o se si volesse realizzare il progetto in C# o Python. Le versioni Python e C# ricalcano quella Java, quindi l'uso è pressoché identico. Fare riferimento alle **NOTE** in verde per maggiori dettagli.

FASE 1

1. Scaricare il jar di EMBASP dal sito del corso:
   - Sul sito del corso trovate la versione 7.2.0
   - **NOTA**: qui trovate informazioni sulle release Python e C#
2. Creare un progetto JAVA con il proprio IDE
3. ***Suggerimento***: aggiungere nel progetto una cartella lib contenente:
   - Il jar di EmbASP
   - Il jar di ANTLR4: antlr-4.7-complete.jar
   - L'eseguibile del sistema ASP (ad esempio, DLV2 come nel caso della Demo Sudoku)
   - **NOTA**: su Linux è necessario aggiungere i permessi di esecuzione mentre su Windows è necessario che l'eseguibile del sistema ASP abbia estensione .exe
4. Assicurarsi che entrambe i jar siano riconosciuti come libreria dal proprio IDE
   - Ad esempio, su Eclipse a seconda della versione, potrebbe essere necessario aggiungere i jar al «build path»

FASE 2

1. Creare un oggetto `DesktopService` appropriato in base al sistema ASP che si intende usare, ad esempio:
   - **`DLV2`**`DesktopService` → se si vuole utilizzare **DLV2**
   - **`DLV`**`DesktopService` → se si vuole utilizzare **DLV (prima versione)**
   - Ad esempio, se stiamo implementando su Windows e usiamo DLV2:

```
DesktopService service = new DLV2DesktopService("lib/dlv2.exe");
```

2. Creare un oggetto `DesktopHandler` passando al costruttore l'oggetto `DesktopService` creato al punto 1

```
Handler handler = new DesktopHandler(service);
```

- **NOTA**: similmente, per realizzare un'applicazione Android occorre creare invece un oggetto `AndroidHandler` e un oggetto `DLV2AndroidService` o `DLVAndroidService`

FASE 2

- Se si volessero aggiungere opzioni per il sistema ASP, si possono definire uno o più oggetti di tipo `OptionDescriptor` e aggiungerli all'handler
- Ad esempio, per aggiungere l'opzione `-n 0` a DLV2 in modo da ottenere tutti gli answer set:

```
OptionDescriptor option = new OptionDescriptor("-n 0");
handler.addOption(option);
```

FASE 3

1. Creare uno o più oggetti di tipo ASPInputProgram. Ad esempio:

```
InputProgram program = new ASPInputProgram();
```

   - È possibile definire l'input in diversi modi:
     - tramite stringhe:

```
program.addProgram("a:-b. b.")
```

     - tramite file:

```
program.addFilesPath("programs/file1")
```

     - sotto forma di oggetti:

```
program.addObjectInput(new Cell(1, 2, 3))
```

2. Indicare all'handler gli oggetti di tipo ASPInputProgram

```
handler.addProgram(program);
```

FASE 3

- Per poter definire l'input sotto forma di oggetti, è necessario che tali classi siano dei **JavaBean** e che siano arricchite dalle annotazioni:
  - *@Id(string_name)* : il cui target deve essere la classe stessa e serve a definire il nome del predicato ASP a cui mappare gli oggetti della classe
  - *@Param(integer_position)* : il cui target è un campo di una classe annotate con *@Id* e serve a definire la posizione di tale campo negli atomi che rappresenteranno gli oggetti di tale classe
- In sintesi, tramite le annotazioni specifichiamo come EmbASP deve «tradurre» gli oggetti di una classe in fatti da passare come input al sistema ASP
- **IMPORTANTE**: Tali classi devono essere «registrate» ovvero «presentate» all'ASPMapper che è il componente di EmbASP che si occupa della traduzione, come in questo esempio:

```
ASPMapper.getInstance().registerClass(Cell.class);
```

- **IMPORTANTE**: Tali classi devono essere fornite di un costruttore senza parametri e dei metodi get/set per i campi annotati con *@Param*

FASE 3

```
@Id("cell")
public class Cell {
    @Param(0)
    private int row;
    @Param(1)
    private int column;
    @Param(2)
    private int value;
    …
};
```

`Cell c1=new Cell(1,3,6)` corrisponde al fatto **`cell(1,3,6)`** dove la riga è 1, la colonna è 3 e il valore è 6

`Cell c2=new Cell(6,5,7)` corrisponde al fatto **`cell(6,5,7)`** dove la riga è 6, la colonna è 5 e il valore è 7

NOTA: i campi non annotati non sono considerati per la trasformazione. Se togliessimo `@Param(2)` dal campo `value`, `Cell c1=new Cell(1,3,6)` corrisponderebbe al fatto **`cell(1,3)`** dove la riga è 1 e la colonna è 3

FASE 3: TRADUZIONE DA OGGETTI A FATTI (E VICEVERSA) NELLE VERSIONI C# E PYTHON DI EMBASP

- **NOTA**: nella versione C# di EmbASP in modo simile alla versione Java, al posto del concetto di annotation si usa quello C# Attribute
- Maggiori informazioni qui

```
[Id("cell")]
class Cell {
    [Param(0)]
    private int row;
    [Param(1)]
    private int column;
    [Param(2)]
    private int value;
    …
};
```

- **NOTA**: nella versione Python di EmbASP è necessario implementare la classe astratta Predicate, e aggiungere:
  - un campo *predicateName="string_name"* per definire il nome di predicato,
  - *[("class_field_name_1"), ("class_field_name_2"), …]* : una lista da passare a super nel costruttore che definisce gli argomenti e ne specifica l'ordine.
- Maggiori informazioni qui

```
class Cell(Predicate):
    predicate_name = "cell"
    def __init__(self, row=None,
        column=None, value=None):
    Predicate.__init__(self,[("row",int),
    ("column",int),("value",int)])
    self.row = row
    self.column = column
    self.value = value
    [...]
```

FASE 3

- Attenzione alla possibile incompatibilità tra stringhe e costanti simboliche. Ad esempio, supponiamo di avere la classe:

```
@Id("col")
public class Col {
    @Param(0)
    private int node;
    @Param(1)
    private String col;
    …
}
```

- Supponiamo di specificare il seguente programma logico tramite file:

```
col(X,red) | col(X,green) | col(X,blue) :- node(X).
 :- edge(X,Y), col(X,C), col(Y,C).
edge(1,2). edge(2,3). node(1). node(2). node(3).
```

- Immaginiamo infine di volere che il nodo 1 sia rosso e di specificarlo con un oggetto della classe `Col`:

  `Col c = new Col(1,"red")` che però corrisponde al fatto **`col(1,"red")`** che è diverso da **`col(1,red)`**

- Di conseguenza come answer set otterremmo anche quelli in cui il nodo 1 è colorato in verde o in blue

FASE 3

- Per evitare questa mancata corrispondenza tra costanti simboliche e stringhe esistono due alternative:
  1. Modifichiamo il programma logico usando soltanto stringhe:

```
col(X,"red") | col(X,"green") | col(X,"blue") :- node(X).
 :- edge(X,Y), col(X,C), col(Y,C).
edge(1,2). edge(2,3). node(1). node(2). node(3).
```

  2. Al posto della classe `String` Java, usiamo la classe `SymbolicConstant` che fa parte di EmbASP

```
@Id("col")
public class Col {
      @Param(0)
      private int node;
      @Param(1)
      private SymbolicConstant col;
      …
}
```

  In tal caso, `Col c = new Col(1,new SymbolicConstant("red"))` verrà tradotto come `col(1,red)`

- **NOTA**: il metodo **`getPrograms()`** di **`ASPInputProgram`** può essere utilizzato per visualizzare ed ispezionare cosa è stato aggiunto tramite stringhe e oggetti, e dunque per fare debugging qualora l'esito dell'esito dell'esecuzione del sistema ASP non sia quello previsto. Allo stesso modo il metodo **`getFilesPaths()`** permette di ispezionare i path dei file dati come input.

FASE 4

- EmbASP può invocare il sistema ASP in modo sincrono o asincrono
  - La modalità sincrona è bloccante: si rimane in attesa finchè il sistema non termina, ovvero finchè non restituisce gli eventuali answer set
  - La modalità asincrona **non** è bloccante: il sistema viene invocato in un thread secondario e quando termina viene invocata una funzione *callback*
  - La modalità sincrona è tipicamente più adatta per applicazioni Desktop mentre la modalità asincrona per applicazioni Android
- Per usare la modalità sincrona è sufficiente fare così:

```
Output output = handler.startSync();
```

FASE 4

- Per usare la modalità asincrona è necessario definire una classe che implementi l'interfaccia `Callback` (che fa parte di EmbASP) come segue:

```
public class MyCallback implements Callback {
    @Override
        public void callback(Output output) {
        // qui si può gestire l'output (come avviene nel modo sincrono)
    }
…
};
```

- A questo punto è sufficiente fare:

```
handler.startAsync(callbackObj);
```

- dove `callbackObj` deve essere un oggetto della classe `MyCallback`

FASE 4

- Nel caso in cui ci sia bisogno di invocare più volte il sistema ASP è bene separare ciò che non cambia tra le diverse esecuzioni da ciò che cambia di volta in volta
- Prima della prima esecuzione, creiamo quindi un oggetto `ASPInputProgram` contenente ciò che non cambia (ad esempio, il programma logico comprensivo di alcuni fatti che sono fissi):

```
InputProgram fixedProgram = new ASPInputProgram();
// … Istruzioni per popolare fixedProgram tramite file, stringhe, oggetti …
handler.addProgram(fixedProgram);
```

- Creiamo inoltre un altro oggetto `ASPInputProgram` contenente ciò che cambia e procediamo alla prima esecuzione:

```
InputProgram variableProgram = new ASPInputProgram();
// … Istruzioni per popolare variableProgram tramite file, stringhe, oggetti …
handler.addProgram(variableProgram);
// … prima esecuzione sincrona o asincrona …
```

- Ad ogni esecuzione successiva «ripuliamo» `variableProgram` e lo ripopoliamo:

```
variableProgram.clearAll();
// … Istruzioni per popolare variableProgram tramite file, stringhe, oggetti …
// … esecuzione successiva sincrona o asincrona …
```

FASE 5

- Per convertire l'oggetto `output` in un oggetto di tipo `AnswerSets`:

```
AnswerSets answersets = (AnswerSets) output;
```

- L'oggetto `answersets` contiene tutti gli answer set restituiti dal sistema ASP:

```
for(AnswerSet a: answersets.getAnswersets()){
    System.out.println(a.toString());
}
```

- Per considerare solo gli answer set ottimi:

```
for(AnswerSet a: answersets.getOptimalAnswerSets()){
    System.out.println(a.toString());
}
```

FASE 5

Al pari dei fatti in input, è possibile tradurre gli atomi che fanno parte degli answer set in oggetti di classi che:

- siano annotate opportunamente con *@Id* e *@Param*
- siano dei **JavaBean** e come tali siano dotate di costruttore senza parametri e dei metodi get/set per i campi annotati con *@Param*
- siano state registrate all'ASPMapper
- **NOTA**: anche per C# e Python vale quanto precedentemente riportato

```
for(AnswerSet a: answersets.getAnswersets()){
    try {
        for(Object answersets:a.getAtoms()){
            if(! (obj instanceof Cell))
                continue;
            Cell cell = (Cell) obj;
            sudokuMatrix[cell.getRow()][cell.getColumn()] = cell.getValue();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

IN CONCLUSIONE

- Attenzione a…
  - Usare l'eseguibile del sistema ASP corretto in base al proprio sistema operativo
  - Non dimenticare di registrare le classi usate per input/output all'ASPMapper
- Se si volesse realizzare il progetto in Python o C#?
  - Perché no! In passato EmbASP è stato anche usato per realizzare giochi su Unity!
  - Le implementazioni Python e C# di EmbASP ricalcano quella Java, quindi l'uso è pressoché identico
  - Diversi esempi d'uso, sono disponibili qui: https://embasp.readthedocs.io/en/latest/
- Se si volesse realizzare il progetto in C++ o in un altro linguaggio?
  - Saremmo lieti di ampliare insieme il set di linguaggi supportati!
- E' obbligatorio usare EmbASP per il progetto?
  - No, soltanto consigliato! È sempre possibile «reinventare la ruota», implementando da sé del codice per interfacciarsi con il sistema ASP!
  - Tra le alternative, ci sono JDLV e DLVWrapper