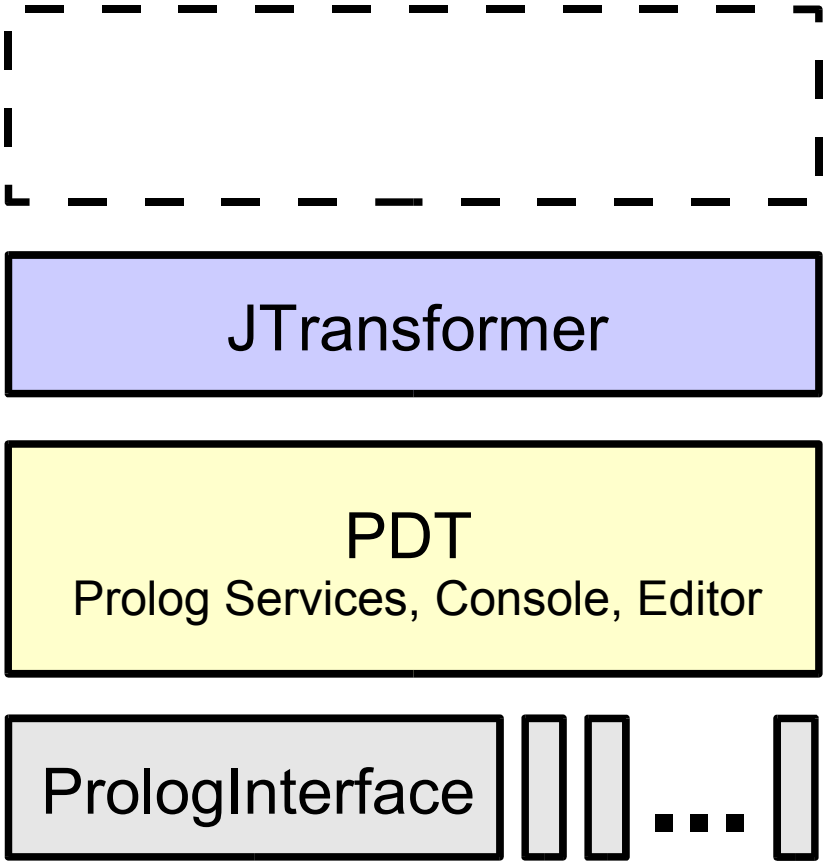
“The Plugins Formerly Known As JTransformer :-)”

23. Feb. '05
--lu

The Big Picture



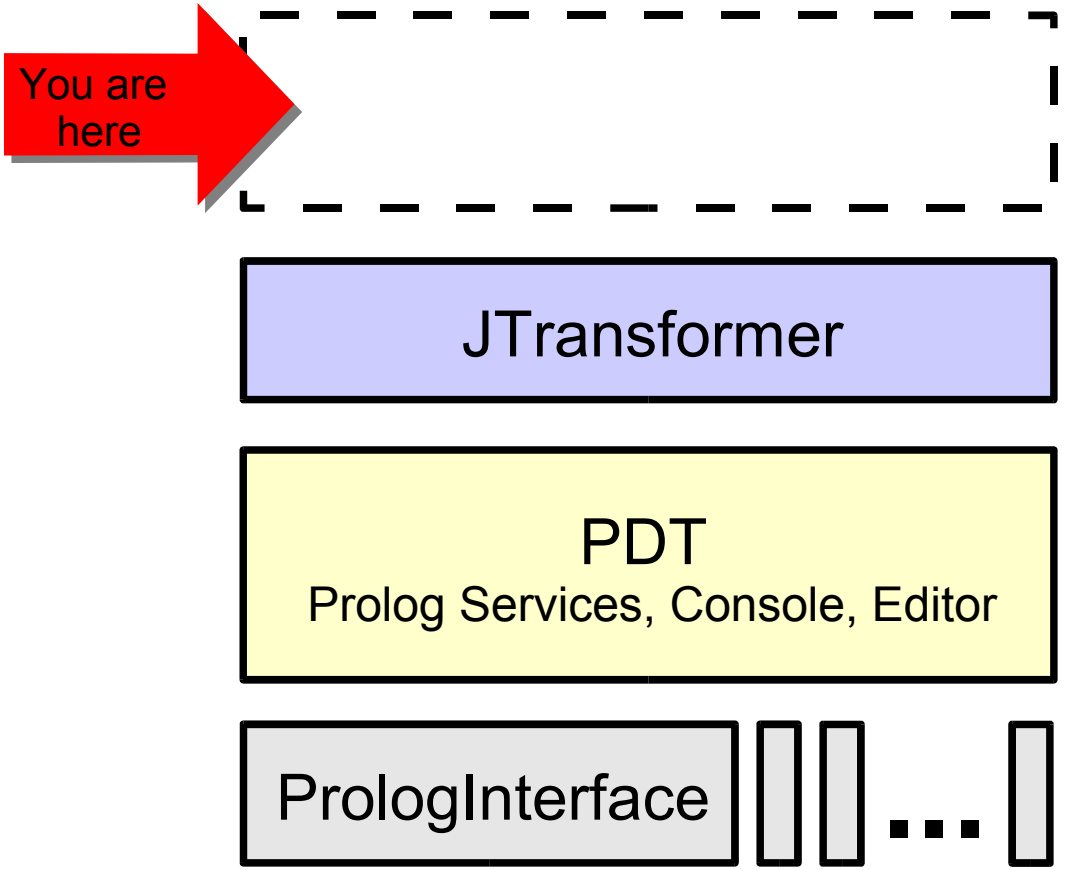
Plugins using JTransformer

Java -> Prolog Element Facts

Prolog Development Tools

Prolog queries, ...

The Big Picture



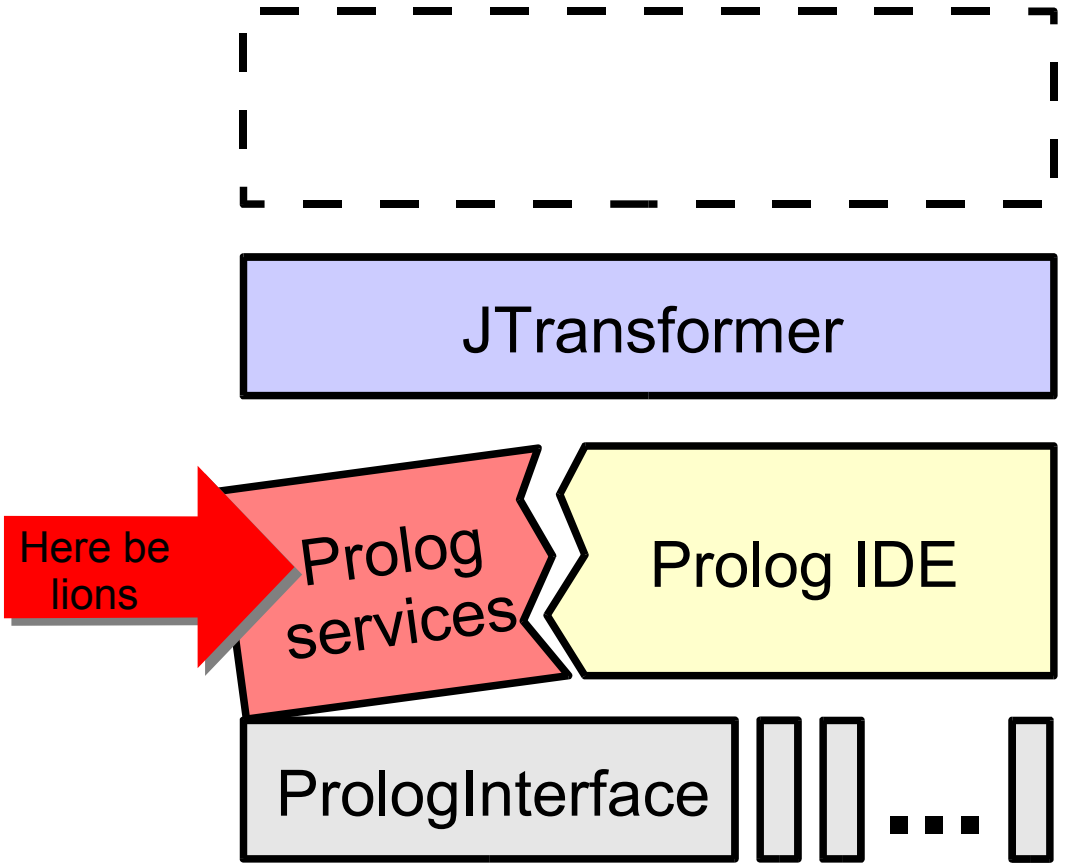
Plugins using JTransformer

Java -> Prolog Element Facts

Prolog Development Tools

Prolog queries, ...

The Big Picture



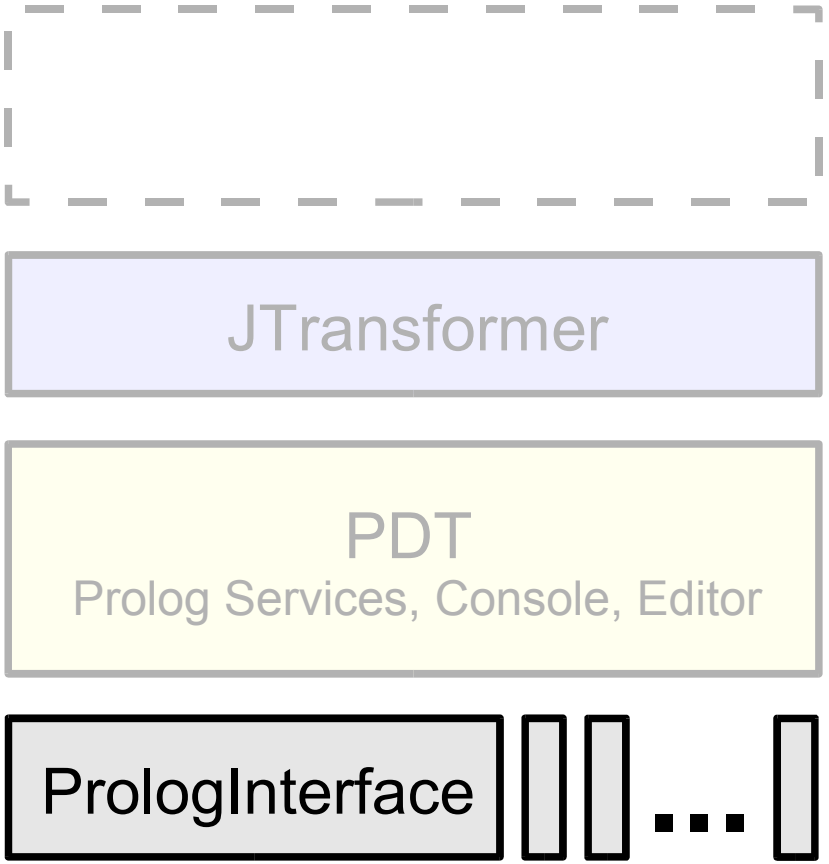
Plugins using JTransformer

Java -> Prolog Element Facts

Prolog Development Tools

Prolog queries, ...

Question No. I



How do I interact with Prolog?

The short answer...

```
PrologInterfaceFactory factory
   = PrologInterfaceFactory.newInstance();
PrologInterface pif = factory.create();
pif.start();

PrologSession session = pif.getSession();
List results
   = session.queryAll("current_thread(Name,Status)");

for (Iterator it=results.iterator();it.hasNext(); ){
   Map m=(Map)it.next();
   System.out.println("Thread: "+m.get("Name")
      + ", Status: " + m.get("Status"));
}

session.dispose();
```

The short answer...

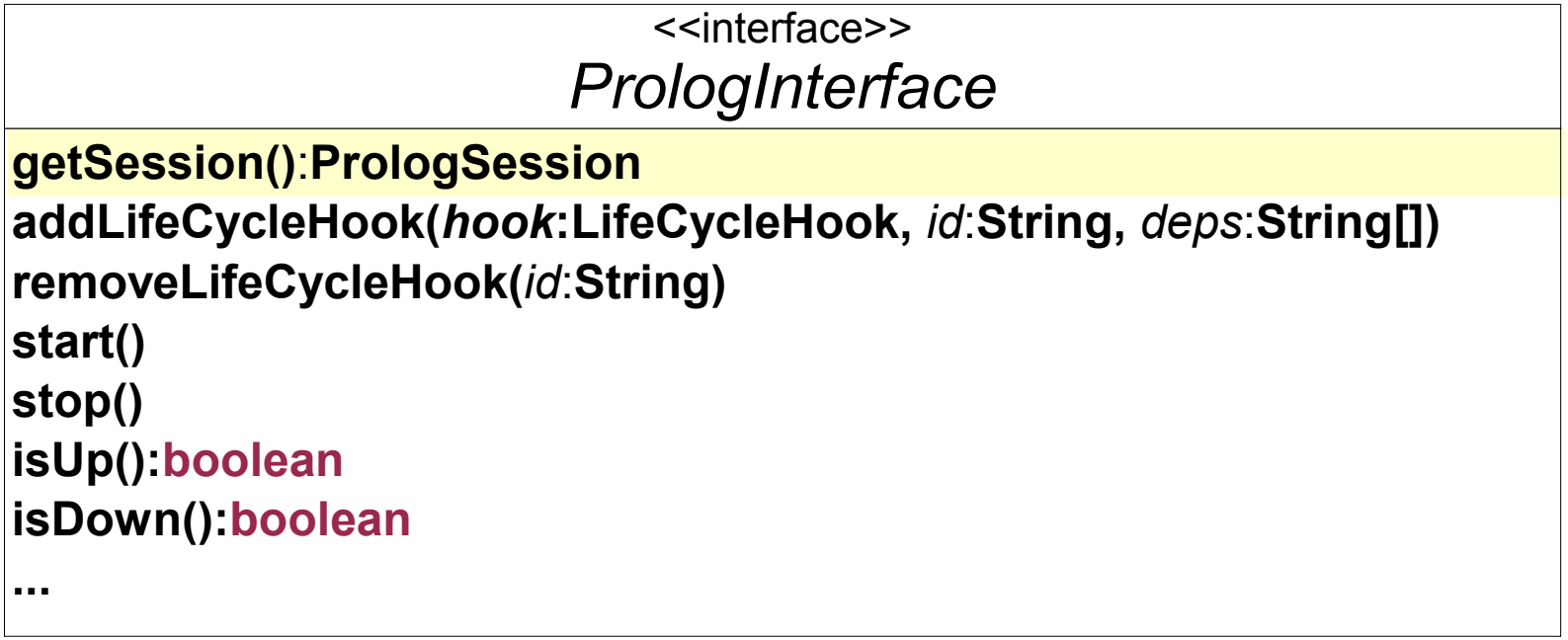
```
PrologInterfaceFactory factory
   = PrologInterfaceFactory.newInstance();
PrologInterface pif = factory.create();
pif.start();

PrologSession session = pif.getSession();
List results
   = session.queryAll("current_thread(Name,Status)");

for (Iterator it=results.iterator();it.hasNext(); ){
   Map m=(Map)it.next();
   System.out.println("Thread: "+m.get("Name")
      + ", Status: " + m.get("Status"));
}

session.dispose();
```

The short answer...

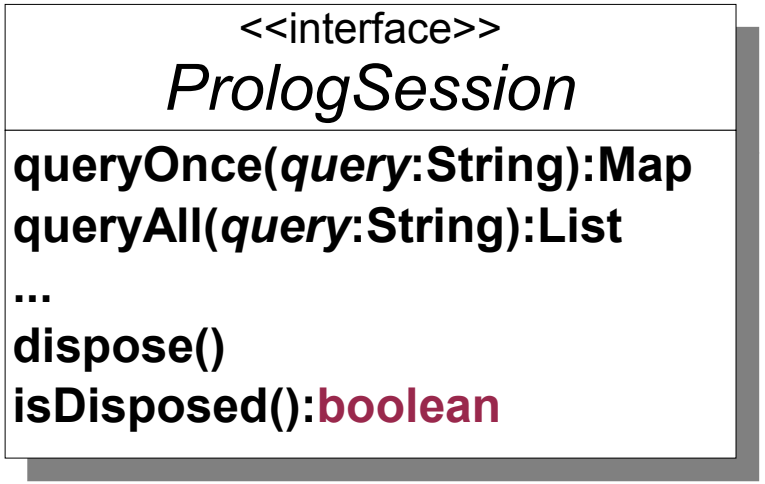
```
PrologInterfaceFactory factory
   = PrologInterfaceFactory.newInstance();
PrologInterface pif = factory.create();
pif.start();

PrologSession session = pif.getSession();
List results
   = session.queryAll("current_thread(Name,Status)");

for (Iterator it=results.iterator();it.hasNext(); ){
   Map m=(Map)it.next();
   System.out.println("Thread: "+m.get("Name")
      + ", Status: " + m.get("Status"));
}

session.dispose();
```

The long answer:

- ... through the ***PrologInterface:***
  - an interface to a Prolog runtime
    - queries
    - life cycle control
    - ...



Sessions



- attach to one Prolog Thread
- rather inexpensive
  - ...but not for free!
- **Not** thread-safe
- **Not** long-lived

May cause Impotence,

makes your skin age faster, etc....!

```
public class BadBadClass
   private PrologSession session;

   public BadBadClass(PrologSession s){
      this.session=s;
   }

   public foo(){
      session.queryOnce("blabla");
      //(...)
   }
}
```

Acceptable, but...

```
public class SoSoClass

   public foo(PrologSession session){
      session.queryOnce(“blabla”);
      //(...)
   }
}
```

What about session.dispose()?

Nice move!

```
public class NiceClass
   private PrologInterface pif;
   public NiceClass(PrologInterface p){
      this.pif=p;
   }

   public bar(){
      PrologSession session =
         pif.getSession();
      try{
         session.queryOnce(“blabla”);
         //(...)
      } finally{
         session.dispose();
      }
   }
}
```

Queries & Results

??

```
Map results = session.queryOnce(some query);
```

- Keys
  - names of variables bound in the query term
  - always **`java.lang.String`**
- Values
  - values bound to the respective variable names
  - **`java.lang.String`** or **`java.util.List`**

Let's see some examples...

Examples

```
A=some_atom,B=A.
```

- result map contains
  - “A” ⇨ “some_atom”
  - “B” ⇨ “some_atom”
- Values are of type **`java.lang.String`**

Examples *(cont.)*

```
A=[1,2,3], member(B,A).
```

- result map contains
  - "A" ⇨ a **`List`** containing "1", "2" and "3"
  - "B" ⇨ "1"
- **`queryOnce`**`()` only returns the **first** solution
- **`queryAll`**`()` returns a **`List`** of **all** solutions
  - Key type: **`java.lang.String`**
  - Value type: **`java.util.Map`**

Examples *(cont.)*

```
true; true.
```

- **`queryOnce`**`()` returns an empty **`Map`**.
- **`queryAll`**`()` returns a **`List`** containing two emtpy **`Map`**s.

Examples *(cont.)*

```
false.
```

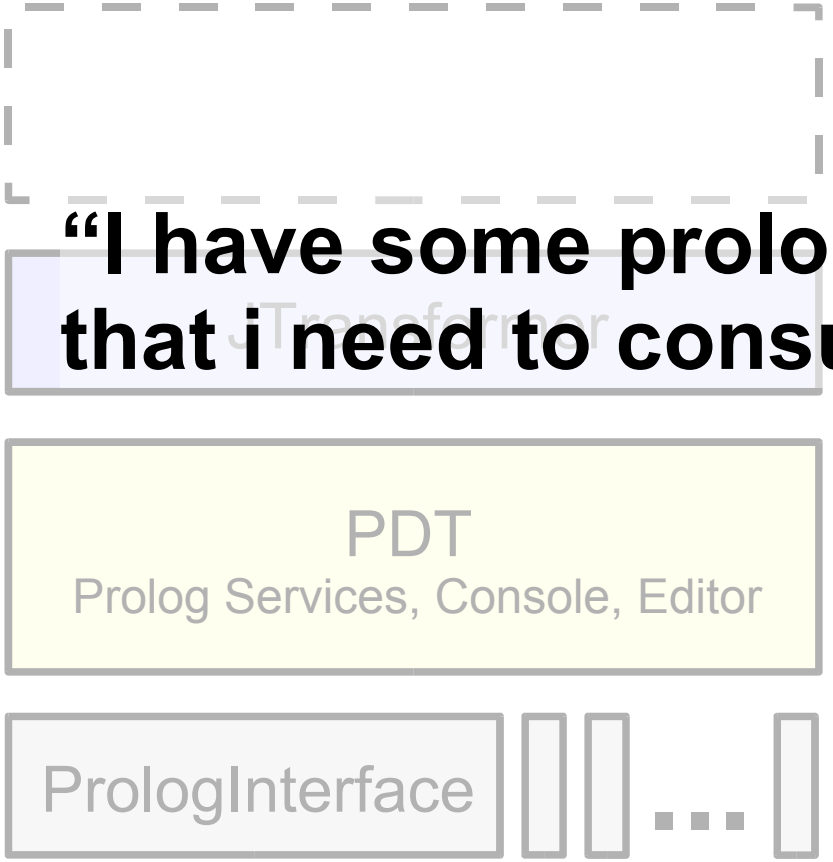
- **`queryOnce`**`()` returns **`null`**.
- **`queryAll`**`()` returns an empty **`List`**.

Quickies...

- nested lists?
  - will yield nested **`List`**s, yep.
- int, float, boolean, ...?
  - nope.
- “manual” backtracking *à la* ***JPL***?
  - slow
  - error-prone
  - use cases*?

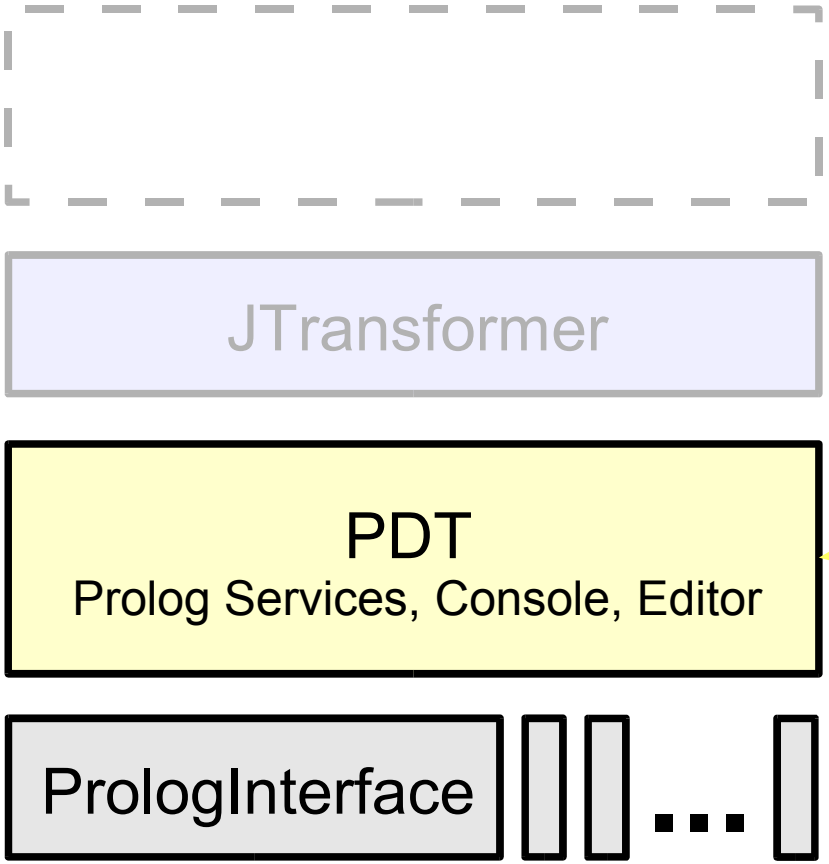
*degenerl_AT_cs.uni-bonn.de

Question No. II



“I have some prolog clauses, that i need to consult on startup...”

Question No. II



How do I know when the PIF is up?

Let's rewind the tape...

You've been here before...

- blabla bla bla ***PrologInterface:***
  - bla blablab bla blabla
    - blabla
    - life cycle control
    - ...

<<blablabla>>

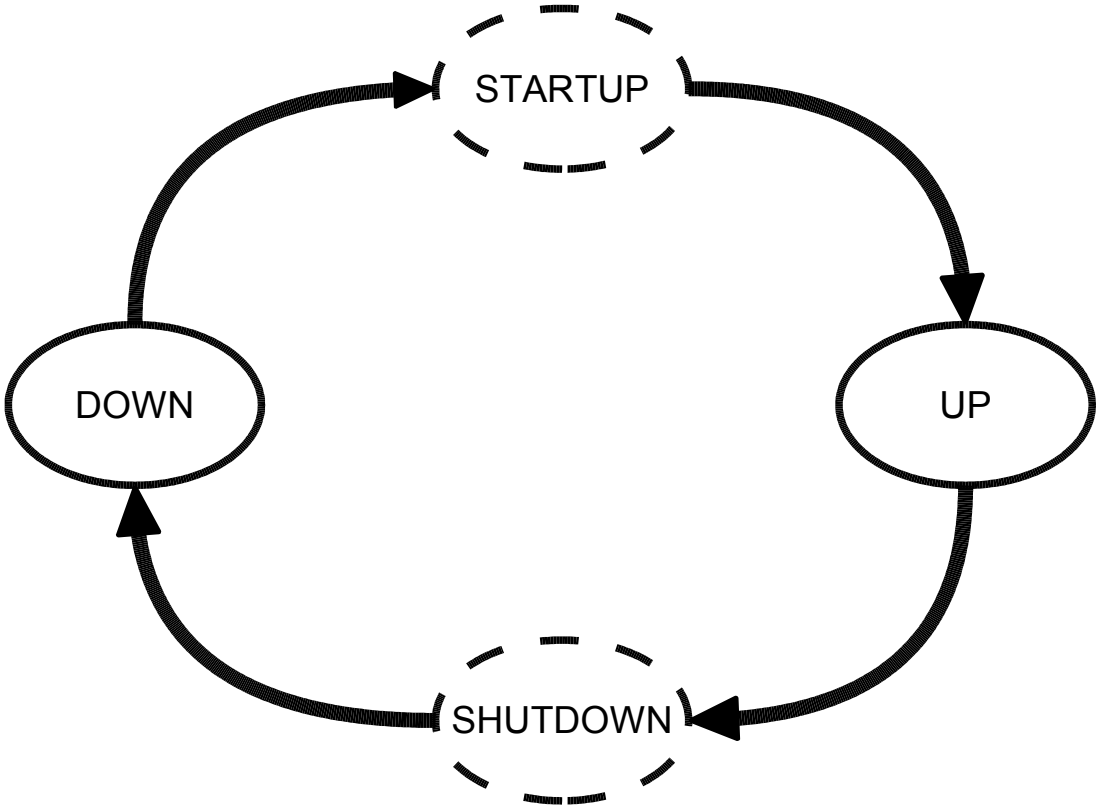
PrologInterface

blaBlabla():BlablaBlablabla

addLifeCycleHook(*hook*:LifeCycleHook, *id*:String, *deps*:String[])
removeLifeCycleHook(*id*:String)
start()
stop()
isUp():boolean
isDown():boolean

...

PrologInterface Life Cycle



<<interface>>

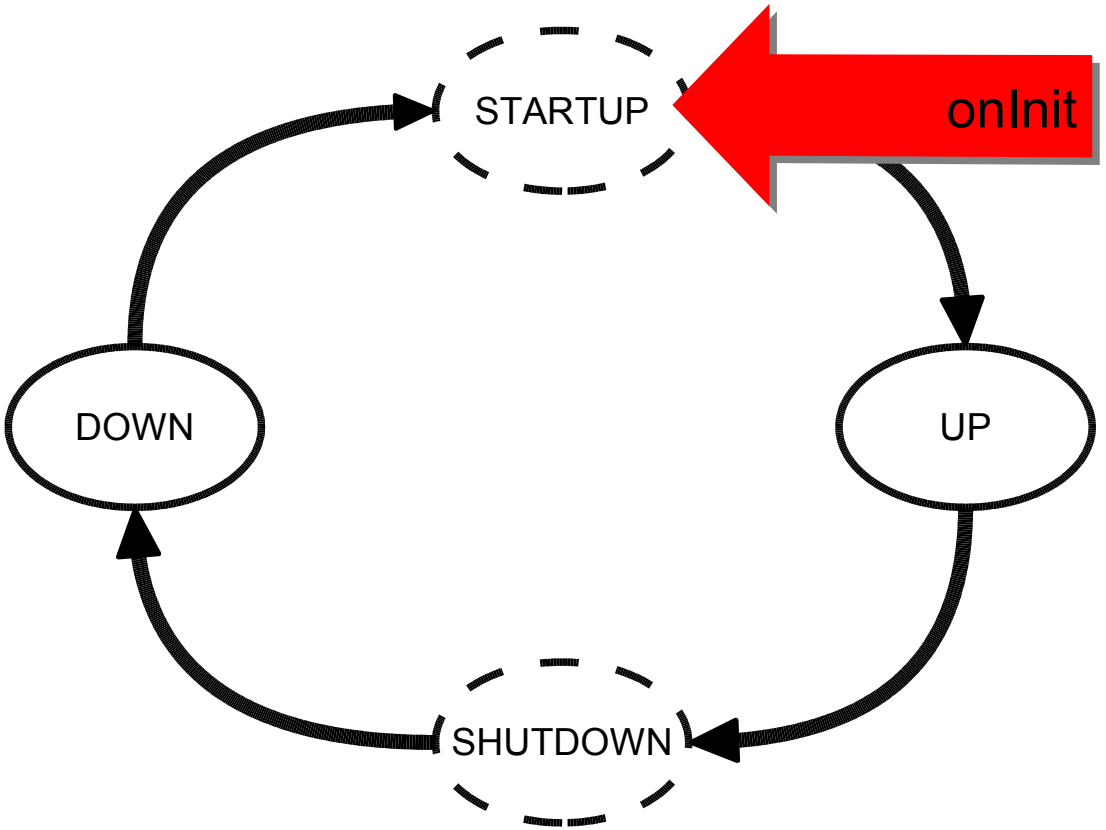
LifeCycleHook

onInit(*pif*:**PrologInterface**, session:**PrologSession**)

afterInit(*pif*:**PrologInterface**)

beforeShutdown(*pif*:**PrologInterface**, session:**PrologSession**)

PrologInterface Life Cycle



- synchronous
- *PIF* is not up yet
  - only use init session!
- don't block

<<interface>>

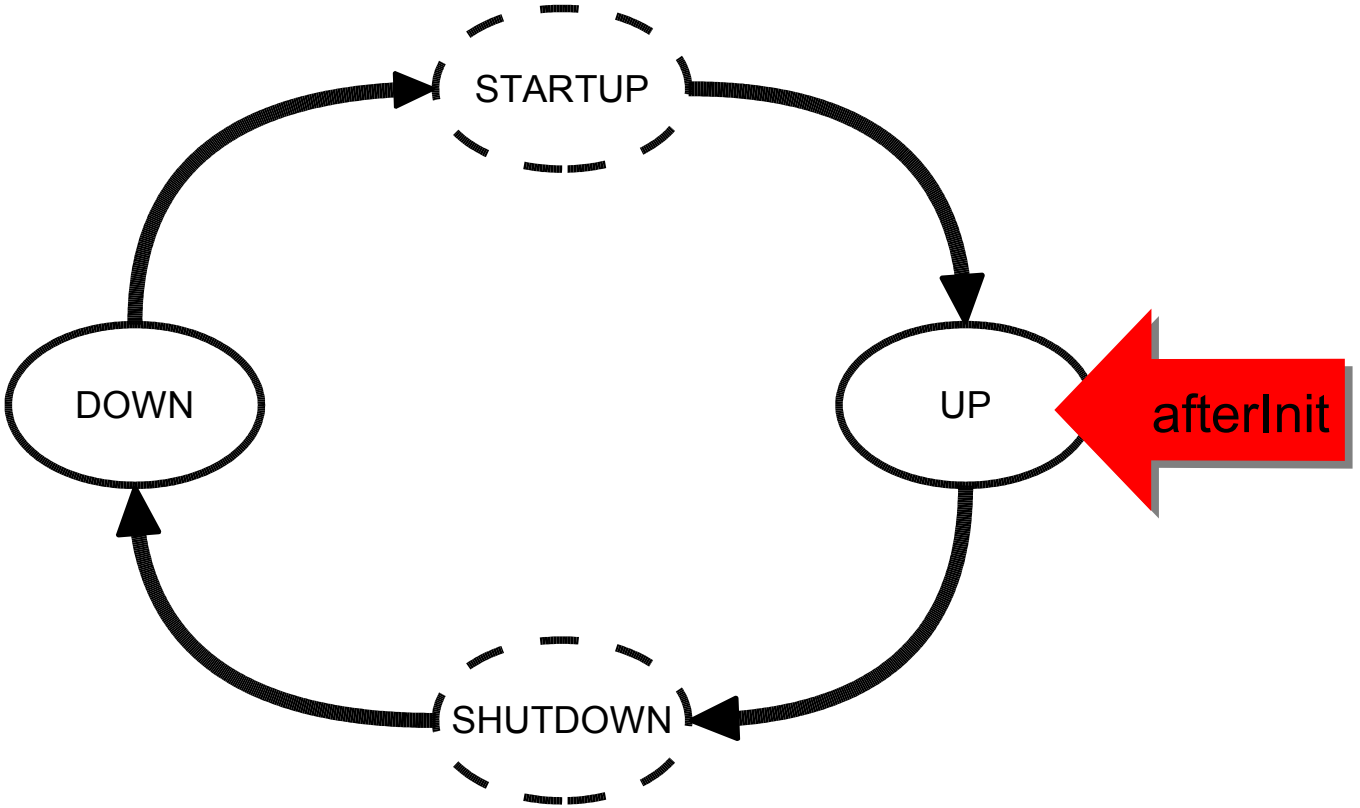
LifeCycleHook

onInit(*pif*:**PrologInterface**, session:**PrologSession**)

afterInit(*pif*:**PrologInterface**)

beforeShutdown(*pif*:**PrologInterface**, session:**PrologSession**)

PrologInterface Life Cycle



- asynchronous
- *PIF* is up

<<interface>>

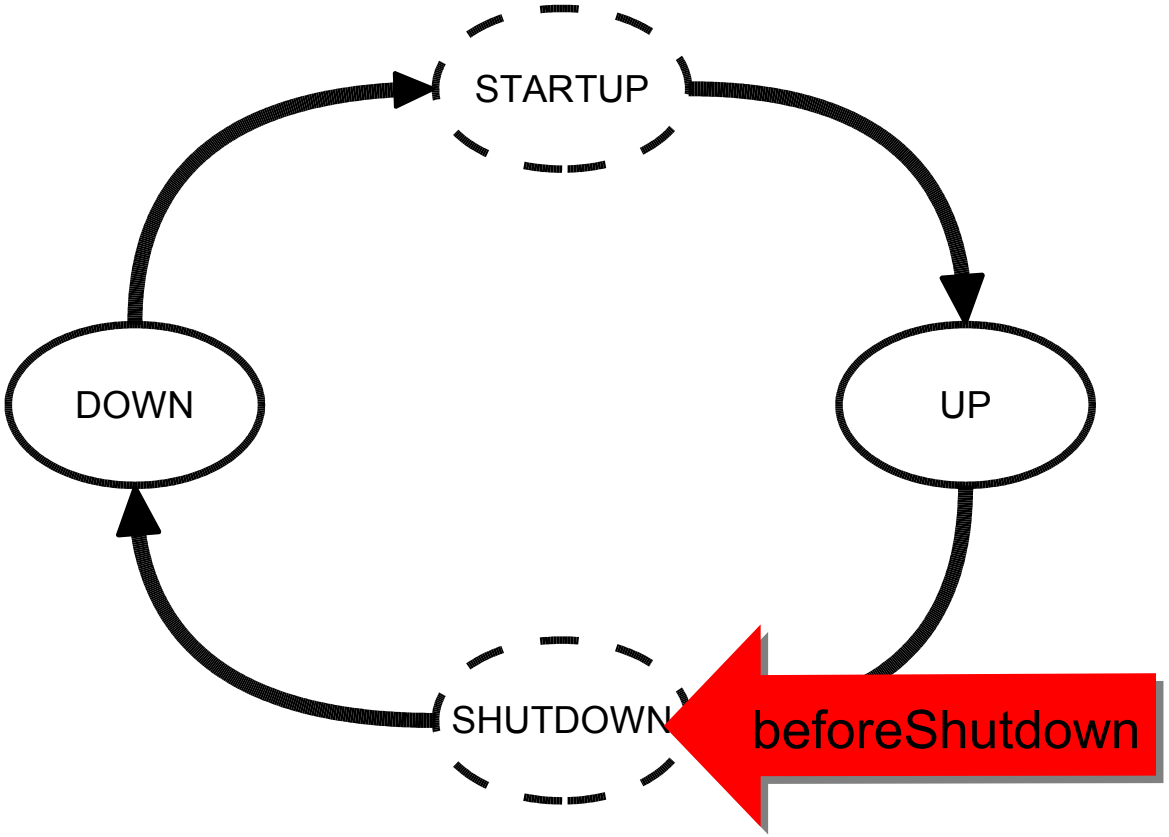
LifeCycleHook

onInit(*pif*:**PrologInterface**, session:**PrologSession**)

afterInit(*pif*:**PrologInterface**)

beforeShutdown(*pif*:**PrologInterface**, session:**PrologSession**)

PrologInterface Life Cycle



- synchronous
- *PIF* is down
  - only use shutdown session!
- don't block

<<interface>>

LifeCycleHook

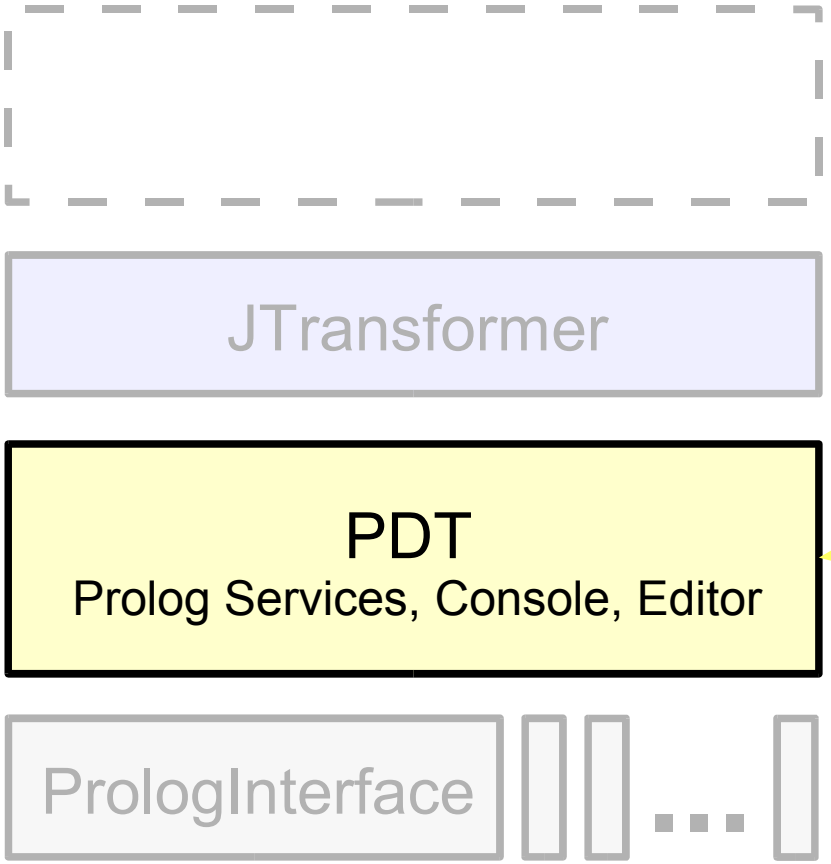
onInit(*pif*:**PrologInterface**, session:**PrologSession**)
afterInit(*pif*:**PrologInterface**)
beforeShutdown(*pif*:**PrologInterface**, session:**PrologSession**)

Notes on LifecycleHooks

- Hook IDs
  - prohibit multiple registration with the same PIF
- Hooks dependencies
  - nested execution
- “Global” hooks
  - extension point **`org.cs3.pdt.hooks`**

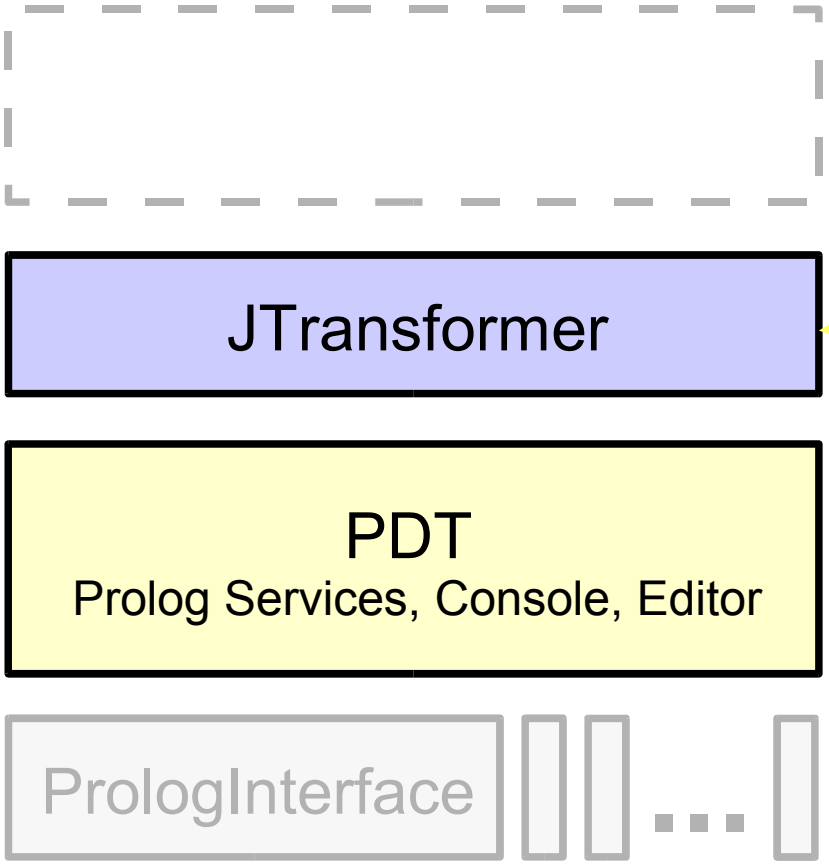
- Alternatives
  - extension point **`org.cs3.pdt.bootstrapContribution`**

Question No. II



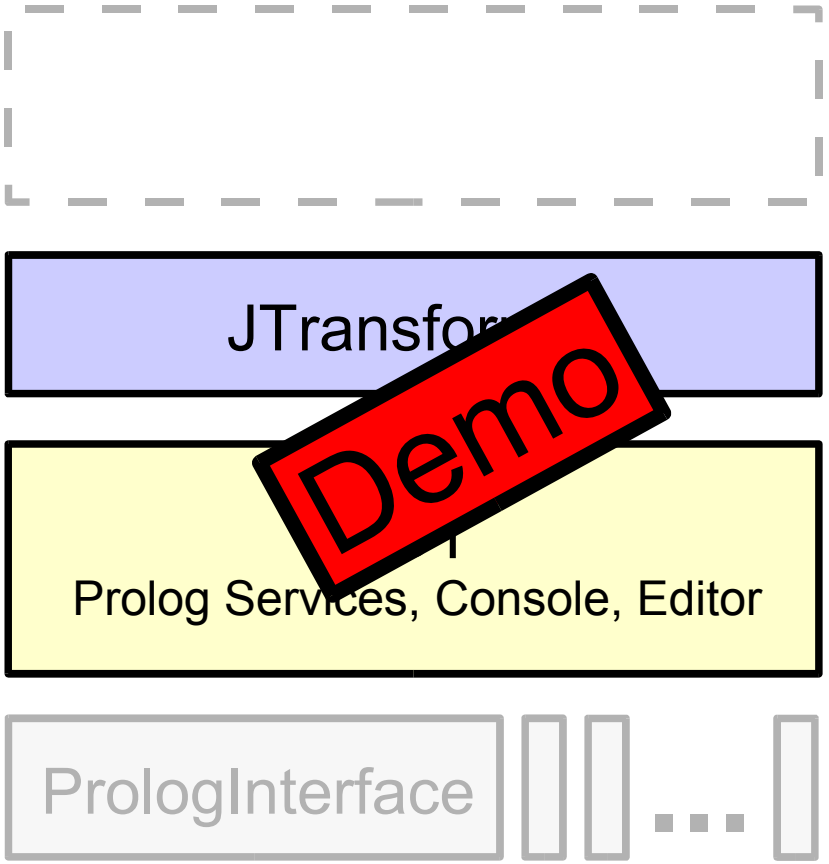
How do I use the Prolog IDE?

Question No. III



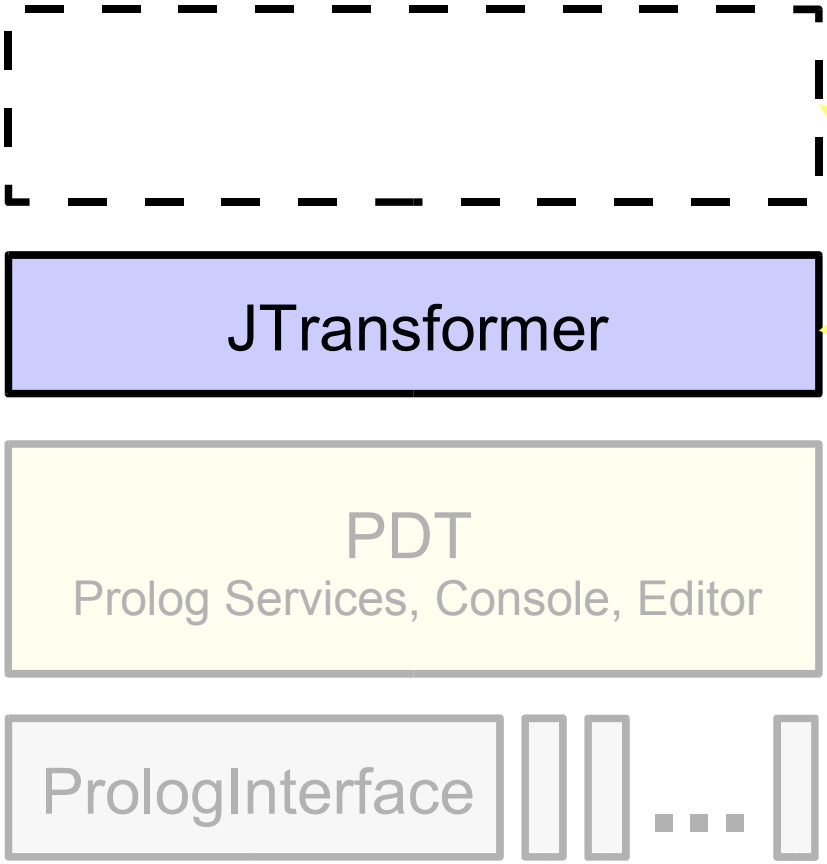
How do I get started with JTransformer?

Answer II & III



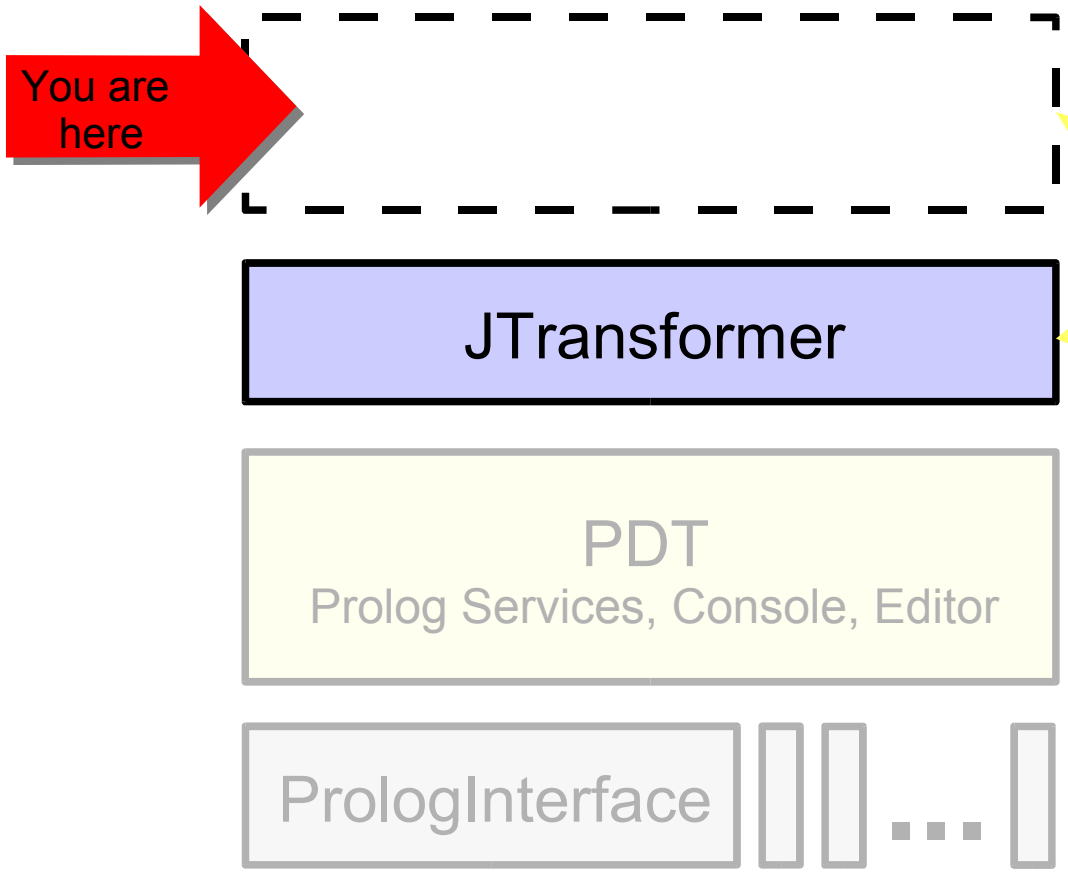
- PDT
  - Prolog Console
    - completion
    - history
  - Prolog Nature
  - Prolog Editor
    - comletion
    - find definition
  - AutoConsult
- JTransformer
  - JLMP Nature
  - PEF Navigator
  - simple queries

Question No. IV



How do plugins make use of JTransformer?

Question No. IV



How do plugins make use of JTransformer?