במימון פרס
הסגל
האקדמי של
IBM חיפה

JTL

the Java Tools Language

יוסי גיל, טל כהן, איתי ממן

הפקולטה למדעי המחשב, טכניון

הַקּוֹרֵא שִׁירָה בַּתַּרְגּוּם, מָשׁוּל לִפְלוֹנִי זֶה הַנּוֹשֵׁק אֶהוּבָתוֹ מִבַּעַד לְמִטְפַּחַת.

ח.נ. ביאליק, 1917

נניח ...

נניח שרצונך למצוא את כל המתודות במחלקה מסויימת, אשר הן גם **public** וגם **final** וגם מחזירות **int**

```
public Method[] pufim_reflection(Class c) {
    Vector<Method> v = new Vector<Method>();
    for (Method m : c.getMethods()) {
      int mod = m.getModifiers();
      if (
           m.getReturnType() == Integer.Type
         &&
           Modifiers.isPublic(mod)
         &&
          Modifiers.isFinal(mod)
      )
       v.add(m);
    }
    return v.toArray(new Method[0]);
}
```

Java Reflection Library

מימוש ב JTL

```
public final int method
```

עוד נניח ...

נניח שרצונך למצוא את כל המחלקות EJB בתכנית Java אשר ממשות את המתודה finalize()

```
subtypes(/class[@name="javax.ejb.EnterpriseBean"])
    /method[
        @name = "finalize"
        and .//returns/@type = "void"
        and not(.//parameter)
]
```

מימוש בעזרת ספריית XIRC
(פורסמה WCRE'04)

מימוש ב JTL

```
class implements 'javax.ejb.EnterpriseBean' {
     public void finalize();
};
```

בוחן פתע

מה פשר השאילתה הבאה?

```
abstract class {
        primitive field;
        no abstract method;
}
```

OUTLINE

- מדוע בכלל לבצע שאילתות על תכניות?
  - שורה ארוכה של ישומים חשובים להנדסת תכנה ולשפות תכנות!
- האם JTL היא הוקוס-פוקוס ואחיזת עיניים?
  - לא. השפה מבוססת על Prolog ועל מודל רלציוני!
- שימוש בשפה כחלק מכלי תכנה אחרים.
- דוגמאות מתקדמות.
- הרחבות, ובעיות פתוחות.

"נניח שברצונך למצוא..." למה להניח?

- **Programming Languages Perspective**
  - **All but trivial applications:** reflection capabilities are required within the language.
- **Software Engineering Perspective:**
  - **Code is the bread and butter of the software engineer.**
    - **One needs to analyze it, organize it, reuse it.**

שאילתות רפלקטיביות כחלק משפת התכנות

- תכנות גנרי (Parametric Polymorphism/Templates):
  - Generic Graph Search Package: חבילה שיכולה לפעול על כל יצוג של גרפים, בתנאי שכל צומת תממש פונקציות אשר מחזירות קשתות אשר מממשות פונקציות אשר...
- MIXINS: הרחבה של OOP לשם תמיכה בעל-מחלקה אבסטרקטית.
  - ה MIXIN `M` מייצר "לכל" מחלקה `T`, תת-מחלקה `M<T>` כך ש `M<T>` יורש מ `T` ומוסיף לו מתודות.
  - כדי שזה יתבצע בשפות כמו Java נדרש לבטא דרישות על המחלקה T.

המוטיבציה האמיתית

- Aspect Oriented Programming - הפעלת טרנספורמציה אוניברסלית על הקוד.
  - למשל הוספת Logging לכל פעולה הקשורה לשינוי מסד הנתונים.
- AspectJ ושפות AOP אחרות הן הרחבה של Java באמצעות מספר מרכיבים:
  - שפת שאילתות לבחירת הקוד לשינוי Pointcuts Language
  - שפת טרנספורמציות לביצוע השינוי.
  - מודולי זמן ריצה וקומפילציה לתמיכה בשפה.

שאילתות על קוד בהנדסת תכנה

- **ארכיטקטורה או Framework של מערכת**: תנאים על הקוד שכל תת מערכת צריכה לקיים.
- **חריגות מכללים**: EJB, אכיפת סטנדרטים של תכנה.
- **שימוש חוזר:** חיפושים בספריות ומאגרים
- **חשיפת ארכיטקטורה**: Architecture Discovery
- **שפות כימוס:** Encapsulation Languages
- **חישוב מטריקות**
- **CASE:** למשל refactoring ב Eclipse.
- **עיטור תכניות:** Program Annotation
- **מעקב דרישות:** requirement tracing
- **ניהול קונפיגורציות**:
- **Micro patterns**

פתרונות קודמים

- שפות ad hoc
- ספריות ניתוח תכנה: למשל Java Reflection Library, BCEL, ועוד
- שפות יעודיות: JAMOOS, Refine

אולי הגיע הזמן לרדת מהעצים?

- פתרונות שיטתיים קודמים – מבוססים על ה Abstract Syntax Tree
  - תיאוריה מתמטית מוכרת
  - בשימוש בכלים ניתוח תכניות קיימים.
  - מתאים לשפות תכנות קלאסיות.
- **Observation**
  - **מודל תכנות לא פשוט:** Visitors ,Multi Methods
  - **מערכת טיפוסים מורכבת ולא יציבה.**
  - **מתאים לניתוח ברמה דקדוקית, אך לא סמנטית של תכניות**
  - **אין תאוריה פשוטה וברורה לביצוע שאילתות**

המודל הרלציוני של JTL

- **הבחנה**: זמנו של Wirth חלף...
  - הלכו עברו להן הפרוצדרות שבתוך פרוצדרות שבתוך פונקציות,
  - המודולים המקוננים שבע פעמים מתו.
  - התכנות המובנה (Structured Programming) של If בתוך While שבתוך Repeat שבתוך Sequence שבתוך if כבר אינו חשוב כל כך...
- בשפות מודרניות, המבנה הוא שטוח, דו או תלת שכבתי, כאשר הסדר בכל שכבה אינו חשוב
  - פונקציות שבתוך מחלקות שבתוך Package.
- נייצג תכניות במסד נתונים רלציוני, עם קשרים ביניהם.
  - **חסרון**: יקשה עלינו לרדת לעומקה של פונקציה ב Java

יצוג תכניות במסד נתונים רלציוני

- טיפוסים יסודיים:
  - מחלקה: `class` `interface` `enum` `@annotation`
  - **Member :** constructor, initializer, static initializer, data member, function member
  - חבילה: `package`
  - String
  - ....
- קשרים בין הטיפוסים באמצעות רלציות דו מקומיות
  - למשל כדי למצוא את המחלקה של member מסויים או כל המחלקות ב package מסויים.

הבסיס

- מילים שמורות של Java שהן יחסים פרימיטיביים אונריים:
  - **`public`**, **`static`**, **`int`**, **`final`**,...
- פרמטר סתום implicit parameter לכל היחסים.
- הגדרת יחסים אונריים חדשים באמצעות אופרטורים לוגיים

הספריה הסטנדרטית של JTL

```
integral := byte | short | int | long;
enumerable := boolean | char;
discrete := integral | enumerable;
real := double | float;
primitive := discrete | real;
typed := _;
reference := typed !primitive !void;

default_access := ![private | protected | public];
visible := !private;

instance := !static member;
```

עוד מבחר מהספריה הסטנדרטית

```
invocable := (*);
method := typed invocable;
constructor := !typed invocable;
initializer := static !typed !invocable member;
code := invocable | initializer;
field := !code member;

extendable := !final type;
overridable := !final !static method;
concrete := !abstract;

java := !native code;
```

עוד פעמונים ושריקות...

- חיפוש ברשימת הארגומנטים לפונקציה
- ביטויים רגולריים על שמות מזהים

```
setter := public void "set[A-Z]?” (*);
boolean_getter := public boolean "is[A-Z]” (*);
other_getter := public !boolean !void "get[A-Z]*” (*);
getter := boolean_getter | other_getter;
```

- ביטויים רגולריים על רשימת ה exceptions
- משתנים הלוכדים תוצאות ביניים
  - `arith_candidate := X method(X,X)`

רלציות רב מקומיות

`members[M]`	holds iff `M` is a member of class **`This`**
`overriding[M]`	holds iff `M` is an overriding member of class **`This`**
`extends[C]`	holds iff **`This`** extends `C`

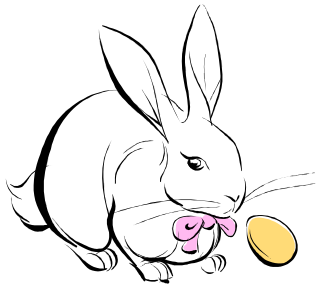
Members

```
inherits[M] := members[M] & !defines[M];
container[C] := C.members[This];
implementing[M] := !abstract overriding[M] M.abstract;
abstracting[M] := abstract overriding[M] !M.abstract;
precursor[M] := M.overriding[This];
```

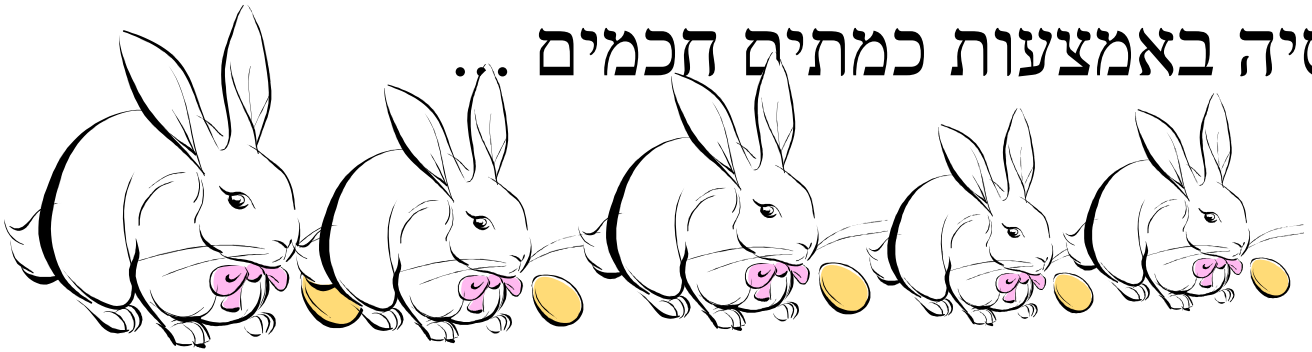
Types

```
extends+[C] := extends[C] | extends[X] X.extends+[C];
extends*[C] := equals[C] | extends+[C];
interfaceof[C] := C.class & C.implements[This];
interfaceof+[C] := C.implements+[This];
interfaceof*[C] := C.implements*[This];
```

מאיזה כובע אתה שולף כל כך הרבה ארנבות?



- הסמנטיקה הבסיסית היא של Prolog
  – ליתר דיוק של Datalog, כלומר חישוב מטה-מעלה.
- הגדרות רקורסיביות
- דקדוק מורחב מעט יותר: שלילה, OR, ורקורסיה בתוך כל הגדרה.
- הצבה למשתני עזר כמו ב Prolog
- ביטול רקורסיה באמצעות כמתים חכמים ...



Smart Quantifiers

- **הרעיון הבסיסי -** יצירת רלציות זמניות, והפעלת כמתים עליהם, כחלק משאילתה.

```
predicate[X]:{
```

list of quantifiers on all possible assignments to X

```
}
```

- **דוגמא**

The “query” defaults to members[#]!

```
abstract class members[#]: {
        all visible;   //universal quantification
        primitive field;//implicit existential quantification
        no abstract method;
        disjoint public, final;
}
```

כעת נוכל **באמת** להבין זאת

```
class implements 'javax.ejb.EnterpriseBean' {
     public void finalize();
};
```

דוגמא פשוטה לקינון ב Java

```
  public Method[] pufim_JTL(Class c) {
      return JTL.Methods.q(
            c.getMethods(),
            ”public final int (*)”
      );
}
```

קינון בשפת התכנות

Similar to smart quantifiers

```
Relation S =
  JTL.q(T,"A?.p[B?,C!,#!,D?!,E?F!,G,_]");
```

Start with relation T, and generate S from it, using predicate A.

- `?` Denotes an input column of A.
- `!` Denotes an output column of A
- `#` means an unnamed column

המרת טיפוסים

- **המימוש הנוכחי**: interpreted command line
- **בגירסא הבאה**: המרה אוטומטית לקלט ופלט מהייצוגים הבאים
  - Java Reflection Objects –
  - BCEL –
  - Strings –

בעיות פתוחות

- "צלילה" לתוך קוד:
  - כעת ניתן רק לשאול על תכונות שטוחות: רשימת משתנים בשימוש, רשימת פונקציות נקראות, וכו.
- חישוב בעולם פתוח:
  - מתי שאילתה היא בטוחה? (מספר התשובות לא ישתנה אם העולם יגדל).
- Java **Transformation** Language = JTL
- כריעות והכלה:
  - האם שאילתה אחת מוכלת בתוך שאילתה אחרת?
- מימוש יעיל.