



תוכנה 1 בשפת Java  
Cool stuff in Java

**תומר הנדלמן**

בית הספר למדעי המחשב  
אוניברסיטת תל אביב

# בתוכנית להיום :

1. Reflections
2. Annotations
3. Java 6 features



# 1. Reflections

בית הספר למדעי המחשב  
אוניברסיטת תל אביב

# השתקפות (reflection)

- למרות שהמחלקה Class היא מחלקה "רגילה" היא משמשת בתור מטה-מחלקה – מחלקה המתארת מחלקות אחרות
- בג'אווה, המבנה של הקוד (מחלקות, שירותים, ושדות) זמין בזמן ריצה וניתן לחקור אותו
- יש מחלקה שהעצמים שלה מייצגים מחלקות, מחלקה שמייצגת חבילות, מחלקה לשירותים, מחלקה לשדות, ומחלקה לבנאים
- ניתן להפעיל בנאים ושירותים בעזרת העצמים המתאימים:
  - `java.lang.Class`
  - `java.lang.Package`
  - `java.lang.reflect.Constructor`
  - `java.lang.reflect.Method`
  - `java.lang.reflect.Field`

# המחלקה Class

```
Rectangle r = ...  
Class x = r.getClass();           // an Object method  
Class y = Rectangle.class;       // literal  
Class z = Class.forName("Rectangle"); // lookup
```

■ העצם שמייצג את המחלקה יכול להחזיר את כל הפרטים לגביה: את שמה, את מי היא מרחיבה ומממשת, את השדות שלה, את השירותים והבנאים שלה

■ למשל, בניית עצם תוך שימוש בבנאי ברירת המחדל:

```
Rectangle a = (Rectangle) z.newInstance();
```

# למה זה טוב?

- מספר דוגמאות בהן זה עוזר:
- Factory .
- בדיקות.
- כשצריך גישה למתודות פרטיות.
- פלאג – אין. (באופן כללי לצורך מתן ורסטיליות ללקוח)

# חסרונות

- אבטחה
- ביצועים
- אין שום אכיפה על גישה סטאטית – שגיאות בזמן ריצה.
- חשיפה של חלקים פנימיים של המחלקה – פוגע בהפשטה (אבסטרקציה) ועשוי להוביל לשגיאות.
- בשורה התחתונה – צריך באמת לדעת מה אתם עושים.



# כמה דוגמאות בקוד...



# Reflection

```
public interface Guest {  
    public void talk();  
}
```

# Reflection

```
public class CowGuest implements Guest {
    private String m_name = null;

    public CowGuest (String name) {
        m_name = name;
    }

    public void talk() {
        System.out.println("Moo moo Moo");
        System.out.println("I'moo " + m_name + " moo");
        System.out.println("Moo moo m m moo");
        System.out.println("Bye. Mooe");
    }

    // special method
    public void giveMilk() {
        System.out.println("Splutch!");
        System.out.println("A lot of yummi milk for everyone!");
    }
}
```

# Reflection

```
public class LectureGuest implements Guest {
    private String m_name = null;

    public LectureGuest (String name) {
        m_name = name;
    }

    public void talk() {
        System.out.println("I'm " + m_name + ". Let's start.");
        System.out.println("Bla bla Java bla");
        System.out.println("Relection bla boring bla annotation bla bla bla");
    }

    // special method
    public void prepareTest() {
        System.out.println("And now, for the test!");
        System.out.println("HAHA! This is going to be so HARD! Ha!");
        System.out.println("EVIL EVIL EVIL! Payback time!");
        System.out.println("I can never solve this. GREAT!");
    }
}
```

# Reflection

עכשיו...נניח שאנחנו רוצים ליצור  
plug-in עם אורח חדש...

# Reflection - problems

```
public class NotSecured {  
  
    public String notStatic()  
    {  
        return "I think seeing this string during reflection is risky";  
    }  
  
    public static String getPublicInfo()  
    {  
        return "I'm a public method, every one knows me";  
    }  
  
    private static String getBiggestSecret()  
    {  
        return "42";  
    }  
  
}
```

# Reflection - problems

```
Method [] methods =
    Class.forName("tau.reflection.NotSecured").getDeclaredMethods();
    for(Method m : methods)
    {
        System.out.println(m.toString());
        System.out.println("Retrieved value is " +
            m.invoke(null));
    }
```



## 2. Annotations

בית הספר למדעי המחשב  
אוניברסיטת תל אביב

# Annotation

אנוטציה היא מטה-מידע (מידע אודות מידע). האנוטציה מאפשרת התייחסות מיוחדת לאובייקט בזמן קומפילציה או בזמן ריצה .



# Annotation

■ סוגי אנוטציות:

■ סמנים (markers) – אנוטציות ריקות  
שממשות כדי לסמן משהו, לדוגמא:

■ `@Deprecated`

■ `public void getA() {};`

■ `@Override`

# Annotation

■ אנוטציות עם איברים בתוכן, לדוגמא :

```
public @interface MyAnnotation {  
    String doSomething();  
    int count; String date();  
}
```

**Usage:**

```
@MyAnnotation (doSomething="What to do", count=1,  
                date="09-09-2005")
```

```
public void mymethod() {  
    ....  
}
```

# Annotation

איך יוצרים אנוטציה :

1. תמיד מתחיל ב-@Interface

2. אין לשים פרמטרים במתודות

3. אין להשתמש throw במתודות

4. הערכים המוחזרים מהמתודות הם :

- ⑩ **primitives**
- ⑩ **String**
- ⑩ **Class**
- ⑩ **enum**
- ⑩ **array of the above types**

# Annotation

## אנוטציות לאנוטציות :

**RetentionPolicy.SOURCE** - יישמרו רק ברמת ה-  
Source – לא ייכנסו לקבצי ה-class . **Deprecated** ,  
דוקומנטציה וכו'

**RetentionPolicy.CLASS** – לקומפילר תהיה גישה  
אליהן אך לא ל-VM

**RetentionPolicy.RUNTIME** – יקראו רק ע"י ה-VM  
בזמן ריצה .

# Annotation

- ושוב נשאלת השאלה, למה זה טוב?
- יכול לשמש למשל ל:
  - בדיקות.
  - אזהרות בזמן קומפילציה.
  - תיעוד קוד (@Documented)
  - קישור בין אובייקט ג'אווה למידע בבסיס הנתונים.
  - ועוד דברים שאינם במסגרת הקורס...

# Annotation - basic

בעולם ללא אנוטציות :

```
public class Generation3List extends Generation2List
{
    // Author: John Doe
    // Date: 3/17/2002
    // Current revision: 6
    // Last modified: 4/12/2004
    // By: Jane Doe
    // Reviewers: Alice, Bill, Cindy
    // class code goes here
}
```

# Annotation - basic

```
@interface ClassPreamble {  
    String author();  
    String date();  
    int currentRevision() default 1;  
    String lastModified() default "N/A";  
    String lastModifiedBy() default "N/A";  
    String[] reviewers(); // Note use of array  
}
```

# Annotation - basic

```
@ClassPreamble (  
    author = "John Doe",  
    date = "3/17/2002",  
    currentRevision = 6,  
    lastModified = "4/12/2004",  
    lastModifiedBy = "Jane Doe"  
    reviewers = {"Alice", "Bob", "Cindy"} // Note array  
    notation  
)  
public class Generation3List extends Generation2List {  
  
    // class code goes here  
  
}
```



# Annotation - tests

```
import java.lang.annotation.*;

/**
 * Indicates that the annotated method is a test
 * method.
 * This annotation should be used only on parameterless
 * static methods.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

# Annotation - tests

```
public class Tested {
    @Test public static void m1 () { }
    public static void m2 () { }
    @Test public static void m3 () {
        throw new RuntimeException("Boom");
    }
    public static void m4 () { }
    @Test public static void m5 () { }
    public static void m6 () { }
    @Test public static void m7 () {
        throw new RuntimeException("Crash");
    }
    public static void m8 () { }
}
```

# Annotation -test

```
public static void main(String[] args) throws Exception {
    int passed = 0, failed = 0;
    for (Method m : Class.forName(args[0]).getMethods())
    {
        if (m.isAnnotationPresent(Test.class)) {
            try {
                m.invoke(null);
                passed++;
            } catch (Throwable ex) {
                System.out.printf("Test %s failed: %s %n", m,
                                   ex.getCause());
                failed++;
            }
        }
    }
}
```

# Annotation - persistency

דוגמא: שימור העקביות במסד נתונים של עצמים

```
@Retention (RetentionPolicy.RUNTIME)  
@Target (ElementType.FIELD)  
@interface ID {
```

```
@Retention (RetentionPolicy.RUNTIME)  
@Target (ElementType.TYPE)  
@interface Table {  
String tableId();  
}
```

# Annotation - persistency

```
@Table(tableId = "studentsTable")
public class Student {
    @ID public int id;

    public Student(int id)
    {
        this.id = id;
    }
}
```

# Annotation - persistency

```
public void persist(Object obj)
{
    String tableName
        =
        obj.getClass().getAnnotation(tau.annotation.entities.Table.
            class).tableId();
    int id = 0;
    Field[] fields = obj.getClass().getFields();
    for(Field field : fields)
    {
        if(field
            .isAnnotationPresent(tau.annotation.entities.ID.class));
        {
            id = field.getInt(obj);
        }
    }
    System.out.println("now we can go to table:" + tableName
        + " in the dataBase and update:" + id);
}
```

### 3. Some new features in Java6

Example program:  
Universal Translator

# Some new features in Java 6

The main :

```
System.setProperty("java.net.useSystemProxies", "true");
    final TrayIcon trayIcon ;

    if (SystemTray.isSupported()) {

        SystemTray tray = SystemTray.getSystemTray();
        Image image =
Toolkit.getDefaultToolkit().getImage(new
URL("http://wiki.laptop.org/images/thumb/9/92/Translate_icon1.svg/40px-Translate_icon1.svg.png"));
        trayIcon = new TrayIcon(image, "Click to
translate");
```



# Some new features in Java 6

## Listening to mouse clicks (awt):

```
MouseListener mouseListener = new MouseListener() {

    public void mouseClicked(MouseEvent e)
    {
        try {

            settings.setTranslatedContent (Translate.translate(getClipboardContent
                s(), Language.ENGLISH, settings.getTargetLanguage()));
            trayIcon.displayMessage("Translated",
                settings.getTranslatedContent(), TrayIcon.MessageType.NONE);
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }
}
```

# Some new features in Java 6

## The state :

```
public class Settings
{
    private String translatedContent = "";
    private String targetLanguage = Language.HEBREW;
    private String sourceLanguage = Language.ENGLISH;

    ...
}
```

# Some new features in Java 6

## Dealing with clipboard :

```
ActionListener copyToClipboardAction = new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        setClipboardContent(settings.getTranslatedContent());
    }
};

public static void setClipboardContent(String text)
{
    Clipboard clipboard =
        Toolkit.getDefaultToolkit().getSystemClipboard();
    StringSelection ss = new StringSelection(text);
    clipboard.setContents(ss, null);
}
```



The end

**בית הספר למדעי המחשב  
אוניברסיטת תל אביב**