

Co-evolving Annotations and Source Code through Smart Annotations

Andy Kellens

Carlos Noguera

Kris De Schutter

Coen De Roover

Theo D'Hondt



Software Languages Lab - Vrije Universiteit Brussel

Context

▶ Source-code annotations

```
public class AccessorExample {  
    public Integer my_field;  
  
    public Integer getMy_field() {  
        return my_field;  
    }  
}
```

Context

▶ Source-code annotations

Annotations

Meta-Data

```
@AccessedClass
public class AccessorExample {
    @AccessedField
    public Integer my_field;
    @Getter("my_field")
    public Integer getMy_field() {
        return my_field;
    }
}
```

Use of annotations (1)

▶ Aspect-oriented programming

Software Implementation

```
import java.io.*;
import java.net.*;

/**
 * Program to copy a file to another directory.
 * Author: Anand Srinivasan
 */
public class CopyFile {
    // constants values for the operation
    public static final int MAX_BUFFER_SIZE = 1024;
    public static final int MAX_RETRY_COUNT = 3;

    // program options
    private static boolean copy = true;
    private static boolean delete = false;
    private static boolean overwrite = true;

    public static void copyFile(String src, String dest) {
        // check if source file exists
        File srcFile = new File(src);
        if (!srcFile.exists()) {
            System.out.println("Source file does not exist: " + src);
            return;
        }

        // check if destination directory exists
        File destDir = new File(dest);
        if (!destDir.exists()) {
            System.out.println("Destination directory does not exist: " + dest);
            return;
        }

        // create destination file
        File destFile = new File(dest + "/" + srcFile.getName());
        if (destFile.exists()) {
            if (overwrite) {
                System.out.println("Destination file already exists. Overwriting...");
            } else {
                System.out.println("Destination file already exists. Not overwriting.");
                return;
            }
        }

        // copy file
        try {
            FileInputStream srcIn = new FileInputStream(srcFile);
            FileOutputStream destOut = new FileOutputStream(destFile);

            byte[] buffer = new byte[MAX_BUFFER_SIZE];
            int bytesRead;
            while ((bytesRead = srcIn.read(buffer)) != -1) {
                destOut.write(buffer, 0, bytesRead);
            }

            srcIn.close();
            destOut.close();

            System.out.println("File copied successfully: " + destFile.getAbsolutePath());
        } catch (IOException e) {
            System.out.println("Error copying file: " + e.getMessage());
        }
    }

    public static void deleteFile(String src) {
        File srcFile = new File(src);
        if (!srcFile.exists()) {
            System.out.println("Source file does not exist: " + src);
            return;
        }

        try {
            srcFile.delete();
            System.out.println("File deleted successfully: " + srcFile.getAbsolutePath());
        } catch (IOException e) {
            System.out.println("Error deleting file: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Usage: java CopyFile <src> <dest>");
            return;
        }

        String src = args[0];
        String dest = args[1];

        copyFile(src, dest);
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter = NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        // store argument GUI elements
        this.frame = frame;
        this.label = label;

        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        newYearMillis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 * componentWidth / stringWidth);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            @Override public void keyTyped(KeyEvent e) {
                // do nothing
            }
        });
    }
}
```

```
public class FileDownload {
    public static void download(String address, String localFileName) {
        try {
            URL url = new URL(address);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setDoOutput(true);
            OutputStream out = conn.getOutputStream();
            out.write("GET / HTTP/1.1\r\n\r\n");
            out.close();

            InputStream in = conn.getInputStream();
            File localFile = new File(localFileName);
            FileOutputStream fos = new FileOutputStream(localFile);

            byte[] buffer = new byte[1024];
            int numRead;
            while ((numRead = in.read(buffer)) != -1) {
                fos.write(buffer, 0, numRead);
            }

            in.close();
            fos.close();

            System.out.println("File downloaded successfully: " + localFile.getAbsolutePath());
        } catch (IOException e) {
            System.out.println("Error downloading file: " + e.getMessage());
        }
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter = NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        // store argument GUI elements
        this.frame = frame;
        this.label = label;

        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        newYearMillis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 * componentWidth / stringWidth);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            @Override public void keyTyped(KeyEvent e) {
                // do nothing
            }
        });
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter = NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        // store argument GUI elements
        this.frame = frame;
        this.label = label;

        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        newYearMillis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 * componentWidth / stringWidth);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            @Override public void keyTyped(KeyEvent e) {
                // do nothing
            }
        });
    }
}
```

Aspect

Tight coupling with structure

Use of annotations (1)

▶ Aspect-oriented programming

Software Implementation

```
import java.io.*;
import java.util.*;

/**
 * Program to copy a file to another directory.
 * Author: Anand Kulkarni
 */
public class CopyFile {
    // constants values for the operation
    public static final int DEFAULT_BUFFER_SIZE = 1024;
    public static final int DEFAULT_TIMEOUT = 10;

    // program options
    private static boolean copy = true;
    private static boolean delete = false;
    private static boolean overwrite = true;
    private static boolean verbose = false;

    public static void copyFile(String srcFile, String destDir)
        throws IOException {
        // check if source file exists
        File srcFileObj = new File(srcFile);
        if (!srcFileObj.exists()) {
            System.out.println("Source file does not exist.");
            return;
        }

        // check if destination directory exists
        File destDirObj = new File(destDir);
        if (!destDirObj.exists()) {
            System.out.println("Destination directory does not exist.");
            return;
        }

        // check if destination directory is a directory
        if (!destDirObj.isDirectory()) {
            System.out.println("Destination is not a directory.");
            return;
        }

        // get the file name
        String fileName = srcFileObj.getName();

        // create the destination file
        File destFileObj = new File(destDirObj, fileName);

        // check if destination file exists
        if (destFileObj.exists()) {
            if (overwrite) {
                System.out.println("Destination file exists. Overwriting.");
            } else {
                System.out.println("Destination file exists. Not overwriting.");
                return;
            }
        }

        // copy the file
        try {
            FileInputStream in = new FileInputStream(srcFileObj);
            FileOutputStream out = new FileOutputStream(destFileObj);
            byte[] buffer = new byte[DEFAULT_BUFFER_SIZE];
            int bytesRead;
            while ((bytesRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, bytesRead);
            }
            in.close();
            out.close();
        } catch (IOException e) {
            System.out.println("Error copying file: " + e.getMessage());
        }

        // delete the source file
        if (delete) {
            srcFileObj.delete();
        }

        // print the result
        System.out.println("File copied successfully from " + srcFile + " to " + destDir + "/" + fileName);
    }

    public static void main(String[] args) {
        // parse command line arguments
        boolean copy = true;
        boolean delete = false;
        boolean overwrite = true;
        boolean verbose = false;

        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("-c")) copy = true;
            if (args[i].equals("-d")) delete = true;
            if (args[i].equals("-o")) overwrite = true;
            if (args[i].equals("-v")) verbose = true;
        }

        // get source file and destination directory
        String srcFile = args[args.length - 2];
        String destDir = args[args.length - 1];

        // copy the file
        copyFile(srcFile, destDir);
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter = NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        // store argument GUI elements
        this.frame = frame;
        this.label = label;
        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        newYearMillis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 * componentWidth / stringWidth);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            @Override public void keyTyped(KeyEvent e) {
                // do nothing
            }
        });
    }
}
```

```
public class FileDownload {
    public static void main(String[] args) {
        String localFileName = args[0];
        String url = args[1];
        FileOutputStream out = new FileOutputStream(localFileName);
        try {
            URL urlObj = new URL(url);
            HttpURLConnection conn = (HttpURLConnection) urlObj.openConnection();
            conn.setRequestMethod("GET");
            conn.setDoOutput(true);
            conn.setDoInput(true);
            conn.connect();
            int responseCode = conn.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) {
                InputStream in = conn.getInputStream();
                byte[] buffer = new byte[1024];
                int numRead = 0;
                long numWritten = 0;
                while ((numRead = in.read(buffer)) != -1) {
                    out.write(buffer, 0, numRead);
                    numWritten += numRead;
                }
                in.close();
                out.close();
                System.out.println("File downloaded successfully: " + localFileName);
            } else {
                System.out.println("Error downloading file: " + responseCode);
            }
        } catch (IOException e) {
            System.out.println("Error downloading file: " + e.getMessage());
        }
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter =
        NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        // store argument GUI elements
        this.frame = frame;
        this.label = label;
        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        newYearMillis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 * componentWidth / stringWidth);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            @Override public void keyTyped(KeyEvent e) {
                // do nothing
            }
        });
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter = NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        // store argument GUI elements
        this.frame = frame;
        this.label = label;
        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        newYearMillis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 * componentWidth / stringWidth);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            @Override public void keyTyped(KeyEvent e) {
                // do nothing
            }
        });
    }
}
```

Aspect

In terms of domain

Indirection layer

Use of annotations (2)

► Frameworks (e.g. Hibernate)

Software Implementation

```
import java.io.*;
import java.net.*;

/**
 * Simple program to copy a file to another directory.
 * Author: Steve Hatcher
 */
public class CopyFile {
    // constants values for the operation
    public static final int MAX_BUFFER_SIZE = 1024;
    public static final int MAX_RETRY_COUNT = 5;

    // program options
    private static boolean verbose = false;
    private static boolean overwrite = true;
    private static boolean recursive = true;
    private static boolean dryRun = false;

    public static void copyFile(String src, String dest) {
        // check if source file exists
        File srcFile = new File(src);
        if (!srcFile.exists()) {
            System.out.println("Source file does not exist: " + src);
            return;
        }

        // check if destination directory exists
        File destDir = new File(dest);
        if (!destDir.exists()) {
            System.out.println("Destination directory does not exist: " + dest);
            return;
        }

        // get file name
        String fileName = srcFile.getName();
        File destFile = new File(destDir, fileName);

        // check if file already exists
        if (destFile.exists()) {
            if (overwrite) {
                System.out.println("File already exists, overwriting: " + destFile);
            } else {
                System.out.println("File already exists, skipping: " + destFile);
                return;
            }
        }

        // copy file
        try {
            FileInputStream in = new FileInputStream(srcFile);
            FileOutputStream out = new FileOutputStream(destFile);
            byte[] buffer = new byte[MAX_BUFFER_SIZE];
            int bytesRead;
            while ((bytesRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, bytesRead);
            }
            in.close();
            out.close();
        } catch (IOException e) {
            System.out.println("Error copying file: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        // parse command line arguments
        if (args.length < 2) {
            System.out.println("Usage: java CopyFile <src> <dest>");
            return;
        }
        String src = args[0];
        String dest = args[1];

        // set options
        for (int i = 2; i < args.length; i++) {
            String arg = args[i];
            if (arg.equals("-v")) {
                verbose = true;
            } else if (arg.equals("-o")) {
                overwrite = true;
            } else if (arg.equals("-r")) {
                recursive = true;
            } else if (arg.equals("-d")) {
                dryRun = true;
            }
        }

        // copy file
        copyFile(src, dest);
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter = NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        // store argument GUI elements
        this.frame = frame;
        this.label = label;
        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        newYearMillis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 * componentWidth / stringWidth);
    }

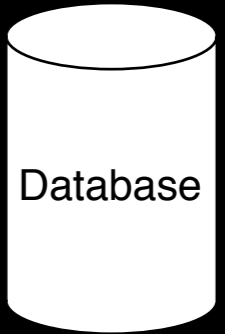
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            @Override public void keyTyped(KeyEvent e) {
                // do nothing
            }
        });
    }
}
```

```
public class FileDownload {
    public static void download(String address, String localFileName) {
        OutputStream out = null;
        URLConnection conn = null;
        InputStream in = null;
        try {
            URL url = new URL(address);
            out = new BufferedOutputStream(
                new FileOutputStream(
                    localFileName));
            conn = url.openConnection();
            in = conn.getInputStream();
            byte[] buffer = new byte[1024];
            int numRead;
            long numWritten = 0;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
            }
        } catch (IOException e) {
            System.out.println("Error downloading file: " + e.getMessage());
        }
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter = NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        // store argument GUI elements
        this.frame = frame;
        this.label = label;
        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        newYearMillis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 * componentWidth / stringWidth);
    }
}
```



Use of annotations (2)

► Frameworks (e.g. Hibernate)

Software Implementation

```
import java.io.*;
import java.net.*;

/**
 * Simple program to copy a file to another directory.
 *
 * @author Arjan Tijms
 */
public class Copy {

    @SuppressWarnings("try")
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Usage: java Copy <source> <target>");
            return;
        }

        String source = args[0];
        String target = args[1];

        File srcFile = new File(source);
        File destFile = new File(target);

        if (!srcFile.exists()) {
            System.out.println("Source file does not exist: " + source);
            return;
        }

        if (!destFile.exists()) {
            System.out.println("Target directory does not exist: " + target);
            return;
        }

        try {
            copyFile(srcFile, destFile);
        } catch (IOException e) {
            System.out.println("Error copying file: " + e.getMessage());
        }
    }

    private static void copyFile(File srcFile, File destFile)
        throws IOException {
        try (InputStream in = new FileInputStream(srcFile);
             OutputStream out = new FileOutputStream(destFile)) {
            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, bytesRead);
            }
        }
    }
}
```

```
public class FileDownload {
    public static void download(String address, String localFileName) {
        InputStream in = null;
        OutputStream out = null;
        try {
            URL url = new URL(address);
            out = new BufferedOutputStream(
                new FileOutputStream(localFileName));
            conn = url.openConnection();
            in = conn.getInputStream();
            byte[] buffer = new byte[1024];
            int numRead;
            long numWritten = 0;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
            }
        } catch (IOException e) {
            System.out.println("Error downloading file: " + e.getMessage());
        }
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter =
        NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        this.frame = frame;
        this.label = label;
        this.newYearMillis = System.currentTimeMillis();
        this.message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int
        fontStyle,
        String text) {
        int fontSize = componentWidth * 2 /
            text.length();
        Font font = new Font(fontName, fontStyle,
            fontSize);
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter =
        NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        this.frame = frame;
        this.label = label;
        this.newYearMillis = System.currentTimeMillis();
        this.message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 /
            text.length();
        Font font = new Font(fontName, fontStyle,
            fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 *
            componentWidth / stringWidth);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            @Override public void keyTyped(KeyEvent e) {
                // ...
            }
        });
    }
}
```

```
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter =
        NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        this.frame = frame;
        this.label = label;
        this.newYearMillis = System.currentTimeMillis();
        this.message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 /
            text.length();
        Font font = new Font(fontName, fontStyle,
            fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 *
            componentWidth / stringWidth);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            @Override public void keyTyped(KeyEvent e) {
                // ...
            }
        });
    }
}
```



@Table
@Column
@Entity
...

Brittleness of annotations

```
public class AccessorExample {  
  
    public Integer my_field;  
    public Integer another_field;  
  
    @Getter("my_field")  
    public Integer getMy_field() {  
        return my_field;  
    }  
  
}
```

Brittleness of annotations

```
public class AccessorExample {  
  
    public Integer my_field;  
    public Integer another_field;  
  
    @Getter("my_field")  
    public Integer getMy_field() {  
        return my_field;  
    }  
  
}
```



Brittleness of annotations

```
public class AccessorExample {  
  
    public Integer my_field;  
    public Integer another_field;  
  
    @Getter("my_field")  
    public Integer getMy_field() {  
        return my_field;  
    }  
  
    public Integer getAnother_field() {  
        return another_field;  
    }  
  
}
```



Brittleness of annotations

```
public class AccessorExample {
```

```
    public Integer my_field;  
    public Integer another_field;
```

```
    @Getter("my_field")  
    public Integer getMy_field() {  
        return my_field;  
    }
```



```
    public Integer getAnother_field() {  
        return another_field;  
    }
```

Missing



```
}
```

Brittleness of annotations

```
public class AccessorExample {
```

```
    public Integer my_field;  
    public Integer another_field;
```

```
    @Getter("my_field")  
    public Integer getMy_field() {  
        return my_field;  
    }
```



```
    public Integer getAnother_field() {  
        return another_field;  
    }
```

Missing



```
    @Getter("my_field")  
    public Integer getMy_field(Integer my_field)  
    {  
        return my_field;  
    }  
}
```

Brittleness of annotations

```
public class AccessorExample {
```

```
    public Integer my_field;  
    public Integer another_field;
```

```
    @Getter("my_field")  
    public Integer getMy_field() {  
        return my_field;  
    }
```



```
    public Integer getAnother_field() {  
        return another_field;  
    }
```

Missing



```
    @Getter("my_field")  
    public Integer getMy_field(Integer my_field)  
    {  
        return my_field;  
    }
```

Incorrect



Cause of brittleness

- ▶ **Annotations not linked to code**
- ▶ **Implicit assumptions/rules about:**
 - Where annotation should apply
 - Annotated location should follow
- ▶ **E.g.**
 - “All methods returning a field should be annotated with @Getter”
 - “If a method is annotated with @Getter, it should follow the Java naming convention”



- ▶ **Document assumptions/annotation constraints**
 - Logic program queries
- ▶ **Verify with respect to source code**
- ▶ **Tool support**
 - Eclipse plugin

Example of Smart Annotation

► Integration with Java annotations

```
@Target(ElementType.METHOD)

public @interface Getter {

}
```

Example of Smart Annotation

► Integration with Java annotations

```
@Target(ElementType.METHOD)
@TargetVariable("method")
public @interface Getter {
    @Necessary
    public static final String NAMING_CONVENTION
        = "?method methodDeclarationHasName: {get*}";

    @Necessary
    @Sufficient
    public static final String STRUCTURAL =
        "?method isMethodDeclaration, " +
        "?class definesMethod: ?method, " +
        "?class definesVariable: ?variable, " +
        "?method returns: ?variable";
}
```

Example of Smart Annotation

► Integration with Java annotations

Target of
annotation

Embedded
Annotation
rules

```
@Target(ElementType.METHOD)
@TargetVariable("method")
public @interface Getter {

    @Necessary
    public static final String NAMING_CONVENTION
        = "?method methodDeclarationHasName: {get*}"

    @Necessary
    @Sufficient
    public static final String STRUCTURAL =
        "?method isMethodDeclaration, " +
        "?class definesMethod: ?method, " +
        "?class definesVariable: ?variable, " +
        "?method returns: ?variable";
}
```

Expressing constraints

- ▶ **Static fields in annotation**
- ▶ **Using logic program queries**
 - Smalltalk Open Unification Language (SOUL)

Naming convention

```
?method methodDeclarationHasName: {get*}
```

Prototypical implementation

```
?method isMethodDeclaration,  
?class definesMethod: ?method,  
?class definesVariable: ?variable,  
?method returns: ?variable
```

Necessary vs. Sufficient

▶ @Necessary

- For all annotated locations the rule should hold

```
@Necessary
public static final String NAMING_CONVENTION
    = "?method methodDeclarationHasName: {get*}";
```

▶ @Sufficient

- All locations where the rule matches should also be annotated

```
@Necessary
@Sufficient
public static final String STRUCTURAL =
    "?method isMethodDeclaration, " +
    "?class definesMethod: ?method, " +
    "?class definesVariable: ?variable, " +
    "?method returns: ?variable";
```

Documenting exceptions

- ▶ **Overly specific/generic rules**
- ▶ **Exceptions in source code**
- ▶ **Document using meta-annotations**
 - @DoesApply
 - @DoesNotApply

```
@DoesNotApply(Getter.class)
public Integer getAnother_field() {
    return another_field;
}
```

Tool support (1)

The screenshot shows an IDE window titled "Java - Smart Annotations Getter Example/src/be/ac/vub/smart_annotations_example/AccessorExample". The main editor displays the following Java code:

```
package be.ac.vub.smart_annotations_example;

public class AccessorExample {

    public Integer my_field;
    public Integer another_field;

    @Getter("My_field")
    public Integer getMy_field() {
        return my_field;
    }

    @Getter("My_field")
    public Integer returnField2(Integer my_field) {
        return my_field;
    }
}
```

A warning message is displayed over the code: "Annotation:Getter Rule:STRUCTURAL Annotation missing".

The left sidebar shows a package hierarchy for "Smart Annotations Getter Example" with files: "AccessedClass.java", "AccessedField.java", "AccessorExample.java", "Getter.java", and "Setter.java".

The bottom panel shows the "Problems" view with the following table:

Description	Resource	Path	Location	
0 errors, 2 warnings, 0 others				
Warnings (2 items)				
Annotation:Getter Rule:NAMING_CONVENTION	AccessorExample.java	/Smart Annotations Gette	line 20	
Annotation:Getter Rule:STRUCTURAL	Annotation	AccessorExample.java	/Smart Annotations Gette	line 16

Tool support (2)

The screenshot shows an IDE window titled "AccessorExample.java". The code in the editor is as follows:

```
package be.ac.vub.smart_annotations_example;

public class AccessorExample {

    public Integer my_field;
    public Integer another_field;

    @Getter("My_field")
    public Integer getMy_field() {
        return my_field;
    }

    public Integer getAnother_field() {
```

A tooltip is displayed over the `getAnother_field()` method, showing two suggestions:

- Add the missing annotation
- Mark as exception

The tooltip also contains the following text:

Problem description: Annotation:Getter
Rule:STRUCTURAL Annotation missing

At the bottom of the tooltip, there is a note: "Press 'Tab' from proposal table or click for focus".

Below the tooltip, a table header is visible:

Description	Resource	Path	Location
-------------	----------	------	----------

At the bottom left of the IDE, there is a status bar showing "0 errors" and "Warnings (2 items)".

Example: Hibernate

- ▶ **Persistence framework**
- ▶ **60+ annotations**
 - @Table: annotated class maps onto a table
 - @Entity: annotated class is persistent
- ▶ **Multiple rules**
- ▶ **If violated, possible errors**

Example: Hibernate

- ▶ **Persistence framework**
- ▶ **60+ annotations**
 - @Table: annotated class maps onto a table
 - @Entity: annotated class is persistent
- ▶ **Multiple rules**
- ▶ **If violated, possible errors**

```
@Entity
@Table(name="Shops")
public class CandyStore {

    public CandyStore(){

    }

    public void method(){
        //Do something
    }
}
```

Example: Hibernate

- ▶ **Persistence framework**
- ▶ **60+ annotations**
 - @Table: annotated class maps onto a table
 - @Entity: annotated class is persistent
- ▶ **Multiple rules**
- ▶ **If violated, possible errors**

```
@Entity
@Table(name="Shops")
public class CandyStore {

    public CandyStore(){}

}

public void method(){}
    //Do something

}
```

If @Table, also @Entity

Class not final

Public constructor without arguments

No final methods

Example: Hibernate (2)

```
@TargetVariable("class")
public @interface Entity {

    @Necessary
    public static final String NO_ARG_CONSTRUCTOR =
        "?class definesConstructor: ?c," +
        "?c methodDeclarationHasParameters: ?params," +
        "[?params size == 0]," +
        "?c methodDeclarationHasModifiers: ?mods," +
        "or(?mods isPublic, ?mods isProtected)";

    @Necessary
    public static final String NO_FINAL_CLASS =
        "?class classDeclarationHasModifiers: ?mods," +
        "not(?mods isFinal)";

    @Necessary
    public static final String NO_FINAL_METHODS =
        "forall(?class definesMethod: ?meth," +
        "and(?meth methodDeclarationHasModifiers: ?mods,not(?mods isFinal))" +
        "));";

    @Sufficient
    public static final String TABLE_REQUIRES_ENTITY=
        "?class classDeclarationHasAnnotation: ? named: {Table}";
}
```

**Public constructor
without arguments**

Class not final

No final methods

**If @Table, also
@Entity**

Example: Hibernate (2)

```
@TargetVariable("class")
public @interface Entity {

    @Necessary
    public static final String NO_ARG_CONSTRUCTOR =
        "?class definesConstructor: ?c," +
        "?c methodDeclarationHasParameters: ?params," +
        "[?params size == 0]," +
        "?c methodDeclarationHasModifiers: ?mods," +
        "or(?mods isPublic, ?mods isProtected)";

    @Necessary
    public static final String NO_FINAL_CLASS =
        "?class classDeclarationHasModifiers: ?mods," +
        "not(?mods isFinal)";

    @Necessary
    public static final String NO_FINAL_METHODS =
        "forall(?class definesMethod: ?meth," +
        "and(?meth methodDeclarationHasModifiers: ?mods,not(?mods isFinal))" +
        "));";

    @Sufficient
    public static final String TABLE_REQUIRES_ENTITY=
        "?class classDeclarationHasAnnotation: ? named: {Table}";
}
```

**Public constructor
without arguments**

Class not final

No final methods

**If @Table, also
@Entity**

Conclusions & Future work

- ▶ **Co-evolve annotations and source code**
- ▶ **Make assumptions explicit and verifiable**
- ▶ **Necessary and sufficient conditions**
- ▶ **Support for Exceptions**

- ▶ **Tool support**
 - Debugging support
 - Externalizing the conditions
- ▶ **Object annotations**

Co-evolving Annotations and Source Code through Smart Annotations

Andy Kellens
Carlos Noguera
Kris De Schutter
Coen De Roover
Theo D'Hondt



<http://soft.vub.ac.be>



Software Languages Lab - Vrije Universiteit Brussel