Formatted Code Snippets – As used in Survey

See also Sn_m:c in CodeSnippets.xlsx, where n=snippet number, m=method chains (1=yes, 2=no), c=comments (1=good, 2=bad, 3=none).

- Indentation: 4 blanks
- Font: Courier New, 10 pt
- Max 83 char per line
- Tabs are 6 blanks in this Word document => Make sure to avoid tabs in code
- Empty lines:
    - Before: always before comment, before return unless it is only a {
    - After: not after the comment; typically not after the if; if comment and following statement are 5 lines, then empty line if possible.
    - Between: between complex statements;
- Method chains in red
- Additional code for resolution in blue
- Holes for the cloze questions are green

Inhalt

Snippet 1

hirondelle.web4j.Controller.logAndEmailSeriousProblem

http://www.web4j.com/web4j/javadoc/src-html/hirondelle/web4j/Controller.html#line.381

S1_1:1 method chains, good comments

```
/**
 * Informs the webmaster of an unexpected problem (Exception “ex”)
 * with the deployed application (indicated by “aRequest”).
 */
public void logAndEmailSeriousProblem(Throwable ex, HttpServletRequest aRequest)
{
    /* Create trouble ticket with context reference. */
    TroubleTicket troubleTicket = new TroubleTicket(ex, aRequest);

    /* Log message to file. */
    fLogger.severe("TOP LEVEL CATCHING Throwable.");
    fLogger.severe(troubleTicket.toString());

    /* Log message to output. */
    log("SERIOUS PROBLEM OCCURRED.");
    log(troubleTicket.toString());

    /* Remember most recent ticket and inform webmaster. */
    aRequest.getSession().getServletContext().
        setAttribute(MOST_RECENT_TROUBLE_TICKET, troubleTicket);
    troubleTicket.mailToWebmaster();
}
```

S1_1:2 method chains, bad comments

```
/**
 * Informs the webmaster of an unexpected problem (Exception “ex”)
 * with the deployed application (indicated by “aRequest”).
 */
public void logAndEmailSeriousProblem(Throwable ex, HttpServletRequest aRequest)
{
    /* Define local variable. */
    TroubleTicket troubleTicket = new TroubleTicket(ex, aRequest);

    /* Log message. */
    fLogger.severe("TOP LEVEL CATCHING Throwable.");
    fLogger.severe(troubleTicket.toString());
    /* Log message again. */
    log("SERIOUS PROBLEM OCCURRED.");
    log(troubleTicket.toString());

    /* Update context and mail trouble ticket. */
    aRequest.getSession().getServletContext().
        setAttribute(MOST_RECENT_TROUBLE_TICKET, troubleTicket);
    troubleTicket.mailToWebmaster();
}
```

S1_1:3 method chains, no comments

```
/**
 * Informs the webmaster of an unexpected problem (Exception “ex”)
 * with the deployed application (indicated by “aRequest”).
 */
public void logAndEmailSeriousProblem(Throwable ex, HttpServletRequest aRequest)
{
    TroubleTicket troubleTicket = new TroubleTicket(ex, aRequest);

    fLogger.severe("TOP LEVEL CATCHING Throwable.");
    fLogger.severe(troubleTicket.toString());

    log("SERIOUS PROBLEM OCCURRED.");
    log(troubleTicket.toString());

    aRequest.getSession().getServletContext().
        setAttribute(MOST_RECENT_TROUBLE_TICKET, troubleTicket);
    troubleTicket.mailToWebmaster();
}
```

S1_2:1 resolved method chains, good comments

```
/**
 * Informs the webmaster of an unexpected problem (Exception “ex”)
 * with the deployed application (indicated by “aRequest”).
 */
public void logAndEmailSeriousProblem(Throwable ex, HttpServletRequest aRequest)
{
    /* Create trouble ticket with context reference. */
    TroubleTicket troubleTicket = new TroubleTicket(ex, aRequest);

    /* Log message to file. */
    fLogger.severe("TOP LEVEL CATCHING Throwable.");
    fLogger.severe(troubleTicket.toString());

    /* Log message to output. */
    log("SERIOUS PROBLEM OCCURRED.");
    log(troubleTicket.toString());

    /* Remember most recent ticket and inform webmaster. */
    HttpSession session = aRequest.getSession();
    ServletContext context = session.getServletContext();
    context.setAttribute(MOST_RECENT_TROUBLE_TICKET, troubleTicket);
    troubleTicket.mailToWebmaster();
}
```

S1_2:2 resolved method chains, bad comments

```
/**
 * Informs the webmaster of an unexpected problem (Exception “ex”)
 * with the deployed application (indicated by “aRequest”).
 */
public void logAndEmailSeriousProblem(Throwable ex, HttpServletRequest aRequest)
{
    /* Define local variable. */
    TroubleTicket troubleTicket = new TroubleTicket(ex, aRequest);

    /* Log message. */
    fLogger.severe("TOP LEVEL CATCHING Throwable.");
    fLogger.severe(troubleTicket.toString());

    /* Log message again. */
    log("SERIOUS PROBLEM OCCURRED.");
    log(troubleTicket.toString());

    /* Update context and mail trouble ticket. */
    HttpSession session = aRequest.getSession();
    ServletContext context = session.getServletContext();
    context.setAttribute(MOST_RECENT_TROUBLE_TICKET, troubleTicket);
    troubleTicket.mailToWebmaster();
}
```

S1_2:3 resolved method chains, no comments

```
/**
 * Informs the webmaster of an unexpected problem (Exception “ex”)
 * with the deployed application (indicated by “aRequest”).
 */
public void logAndEmailSeriousProblem(Throwable ex, HttpServletRequest aRequest)
{
    TroubleTicket troubleTicket = new TroubleTicket(ex, aRequest);

    fLogger.severe("TOP LEVEL CATCHING Throwable.");
    fLogger.severe(troubleTicket.toString());

    log("SERIOUS PROBLEM OCCURRED.");
    log(troubleTicket.toString());

    HttpSession session = aRequest.getSession();
    ServletContext context = session.getServletContext();
    context.setAttribute(MOST_RECENT_TROUBLE_TICKET, troubleTicket);
    troubleTicket.mailToWebmaster();
}
```

Snippet 2

org.unicase.dashboard.impl.NotificationOperationImpl.apply
http://unicase.googlecode.com/svn/trunk/core/org.unicase.dashboard/src/org/unicase/dashboard/impl/NotificationOperationImpl.java

S2_1:1 method chains, good comments

```
/**
 * Apply the transmitted notifications (“nStore”) to the project so that
 * acknowledged notifications are deleted and other ones added.
 */
public void apply(Project project, NotificationStore nStore)
{
    ProjectSpace projectSpace = project.eContainer();

    /* If project space has not been initialized, there is nothing to do. */
    if (projectSpace != null) {

        /* Get project properties and check if there are notifications. */
        PropertyManager manager = projectSpace.getPropertyManager();
        StoreProperty property =
            manager.getLocalProperty(NOTIFICATION_COMPOSITE);

        if (property != null) {
            Value value = property.getValue();

            /* If the project already has notifications                   */
            /* and if transmitted notifications are acknowledged,         */
            /* then remove the transmitted notifications from the project.  */
            /* Otherwise, add the transmitted notifications to the project. */
            if (value != null && value instanceof NotificationComposite) {

                NotificationComposite nComposite = value;

                if (nStore.isAcknowledged()) {
                    nComposite.getNotifications().removeAll(
                        nStore.getNotifications());
                } else {
                    nComposite.getNotifications().addAll(
                        nStore.getNotifications());
                }
            }
        } else {

            /* If the project did not have notifications yet           */
            /* and if transmitted notifications are not acknowledged, */
            /* then add the transmitted notifications to the project  */
            /* and store them in the NOTIFICATION_COMPOSITE property. */
            if (!nStore.isAcknowledged()) {

                NotificationComposite nComposite =
                    Factory.createNotificationComposite();

                nComposite.getNotifications().addAll(
                    nStore.getNotifications());

                manager.setLocalProperty(NOTIFICATION_COMPOSITE, nComposite);
            }
        }
    }
}
```

S2_1:2 method chains, bad comments

```
/**
 * Apply the transmitted notifications (“nStore”) to the project so that
 * acknowledged notifications are deleted and other ones added.
 */
public void apply(Project project, NotificationStore nStore)
{
    ProjectSpace projectSpace = project.eContainer();

    /* If projectSpace is not null */
    if (projectSpace != null) {

        /* Define local variables. */
        PropertyManager manager = projectSpace.getPropertyManager();
        StoreProperty property =
            manager.getLocalProperty(NOTIFICATION_COMPOSITE);

        if (property != null) {
            Value value = property.getValue();

            /* If value is not null and if nstore is acknowledged,   */
            /* then remove nstore-notifications from notifications.  */
            /* Otherwise, add nstore-notifications to notifications. */
            if (value != null && value instanceof NotificationComposite) {
                NotificationComposite nComposite = value;

                if (nStore.isAcknowledged()) {
                    nComposite.getNotifications().removeAll(
                        nStore.getNotifications());
                } else {
                    nComposite.getNotifications().addAll(
                        nStore.getNotifications());
                }
            }
        } else {

            /* If property is not null                  */
            /* and if nStore is not acknowledged,       */
            /* then add nStore-notifications,           */
            /* and set NOTIFICATION_COMPOSITE property. */
            if (!nStore.isAcknowledged()) {

                NotificationComposite nComposite =
                    Factory.createNotificationComposite();

                nComposite.getNotifications().addAll(
                    nStore.getNotifications());

                manager.setLocalProperty(NOTIFICATION_COMPOSITE, nComposite);
            }
        }
    }
}
```

S2_1:3 method chains, no comments

```
/**
```

```
 * Apply the transmitted notifications (“nStore”) to the project so that
 * acknowledged notifications are deleted and other ones added.
 */
public void apply(Project project, NotificationStore nStore)
{
    ProjectSpace projectSpace = project.eContainer();

    if (projectSpace != null) {

        PropertyManager manager = projectSpace.getPropertyManager();
        StoreProperty property =
            manager.getLocalProperty(NOTIFICATION_COMPOSITE);

        if (property != null) {
            Value value = property.getValue();

            if (value != null && value instanceof NotificationComposite) {
                NotificationComposite nComposite = value;

                if (nStore.isAcknowledged()) {
                    nComposite.getNotifications().removeAll(
                        nStore.getNotifications());
                } else {
                    nComposite.getNotifications().addAll(
                        nStore.getNotifications());
                }
            }
        } else {

            if (!nStore.isAcknowledged()) {
                NotificationComposite nComposite =
                    Factory.createNotificationComposite();

                nComposite.getNotifications().addAll(
                    nStore.getNotifications());

                manager.setLocalProperty(NOTIFICATION_COMPOSITE, nComposite);
            }
        }
    }
}
```

S2_2:1 resolved method chains, good comments

```
/**
 * Apply the transmitted notifications (“nStore”) to the project so that
 * acknowledged notifications are deleted and other ones added.
 */
public void apply(Project project, NotificationStore nStore)
{
    ProjectSpace projectSpace = project.eContainer();

    /* If project space has not been initialized, there is nothing to do. */
    if (projectSpace != null) {

        /* Get project properties and check if there are notifications. */
        PropertyManager manager = projectSpace.getPropertyManager();
        StoreProperty property =
            manager.getLocalProperty(NOTIFICATION_COMPOSITE);
        NotificationList notifications;

        if (property != null) {
            Value value = property.getValue();

            /* If the project already has notifications                      */
            /* and if transmitted notifications are acknowledged,            */
            /* then remove the transmitted notifications from the project.  */
            /* Otherwise, add the transmitted notifications to the project. */
            if (value != null && value instanceof NotificationComposite) {

                NotificationComposite nComposite = value;
                notifications = nComposite.getNotifications();

                if (nStore.isAcknowledged()) {
                    notifications.removeAll(nStore.getNotifications());
                } else {
                    notifications.addAll(nStore.getNotifications());
                }
            }
        } else {

            /* If the project did not have notifications yet           */
            /* and if transmitted notifications are not acknowledged, */
            /* then add the transmitted notifications to the project  */
            /* and store them in the NOTIFICATION_COMPOSITE property. */
            if (!nStore.isAcknowledged()) {

                NotificationComposite nComposite =
                    Factory.createNotificationComposite();
                notifications = nComposite.getNotifications();

                notifications.addAll(nStore.getNotifications());
                manager.setLocalProperty(NOTIFICATION_COMPOSITE, nComposite);
            }
        }
    }
}
```

S2_2:2 resolved method chains, bad comments

```
/**
 * Apply the transmitted notifications (“nStore”) to the project so that
 * acknowledged notifications are deleted and other ones added.
 */
public void apply(Project project, NotificationStore nStore)
{
    ProjectSpace projectSpace = project.eContainer();

    /* If projectSpace is not null */
    if (projectSpace != null) {

        /* Define local variables. */
        PropertyManager manager = projectSpace.getPropertyManager();
        StoreProperty property =
            manager.getLocalProperty(NOTIFICATION_COMPOSITE);
        NotificationList notifications;

        if (property != null) {
            Value value = property.getValue();

            /* If value is not null and if nstore is acknowledged,    */
            /* then remove nstore-notifications from notifications.  */
            /* Otherwise, add nstore-notifications to notifications. */
            if (value != null && value instanceof NotificationComposite) {

                NotificationComposite nComposite = value;
                notifications = nComposite.getNotifications();

                if (nStore.isAcknowledged()) {
                    notifications.removeAll(nStore.getNotifications());
                } else {
                    notifications.addAll(nStore.getNotifications());
                }
            }
        } else {

            /* If property is not null                  */
            /* and if nStore is not acknowledged,       */
            /* then add nStore-notifications,           */
            /* and set NOTIFICATION_COMPOSITE property. */
            if (!nStore.isAcknowledged()) {

                NotificationComposite nComposite =
                    Factory.createNotificationComposite();
                notifications = nComposite.getNotifications();

                notifications.addAll(nStore.getNotifications());
                manager.setLocalProperty(NOTIFICATION_COMPOSITE, nComposite);
            }
        }
    }
}
```

S2_2:3 resolved method chains, no comments

```
/**
 * Apply the transmitted notifications ("nStore") to the project so that
 * acknowledged notifications are deleted and other ones added.
 */
public void apply(Project project, NotificationStore nStore)
{
    ProjectSpace projectSpace = project.eContainer();

    if (projectSpace != null) {

        PropertyManager manager = projectSpace.getPropertyManager();
        StoreProperty property =
            manager.getLocalProperty(NOTIFICATION_COMPOSITE);
        NotificationList notifications;

        if (property != null) {
            Value value = property.getValue();

            if (value != null && value instanceof NotificationComposite) {
                NotificationComposite nComposite = value;
                notifications = nComposite.getNotifications();

                if (nStore.isAcknowledged()) {
                    notifications.removeAll(nStore.getNotifications());
                } else {
                    notifications.addAll(nStore.getNotifications());
                }
            }
        } else {
            if (!nStore.isAcknowledged()) {
                NotificationComposite nComposite =
                    Factory.createNotificationComposite();
                notifications = nComposite.getNotifications();

                notifications.addAll(nStore.getNotifications());
                manager.setLocalProperty(NOTIFICATION_COMPOSITE, nComposite);
            }
        }
    }
}
```

Snippet 3

org.unicase.docExport.docWriter.ITextWriter.writeUTable

http://unicase.googlecode.com/svn/trunk/core/org.unicase.docExport/src/org/unicase/docExport/docWriter/ITextWriter.java

S3_1:1 (Revision 1) method chains, good comments

```
/**
 * The images of a unified table are copied to a simplified “yes/no” table.
 */
protected Table copyUTable(UTable uTable)
{
    /* Create a new table with the same size and color as uTable. */
    Table table = new Table(uTable.getColumnsCount());
    table.setBorderColor(new Color (
        uTable.getBoxModel().getBorderColor().getRed(),
        uTable.getBoxModel().getBorderColor().getGreen(),
        uTable.getBoxModel().getBorderColor().getBlue()));

    /* Go through all entries of uTable and copy image information. */
    Cell cell;
    for (UTableCell uCell : uTable.getEntries()) {

        UParagraph paragraph = uCell.getContent();

        /* Check if there is image information to copy. */
        if (paragraph.getChildren().size() > 0 &&
            paragraph.getChildren().get(0) instanceof UImage) {

            UImage image = paragraph.getChildren().get(0);

            /* Copy the last segment of the image. False and true */
            /* are transformed to “no” and “yes”, respectively.   */
            if (image.getPath().lastSegment().startsWith("false")) {
                cell = new Cell("no");

            } else {
                if (image.getPath().lastSegment().startsWith("true")) {
                    cell = new Cell("yes");
                } else {
                    cell = new Cell(image.getPath().lastSegment());
                }
            }

            /* Copy background color of uCell. */
            if (uCell.getBoxModel().getBackgroundColor() != null) {
                cell.setBackgroundColor(uCell.getBoxModel().getBackgroundColor());
            }
        }
        table.addCell(cell);
    }
    return table;
}
```

S3_1:2 (Revision 1) method chains, bad comments

```
/**
 * The images of a unified table are copied to a simplified “yes/no” table.
 */
```

```
protected Table copyUTable(UTable uTable)
{
    /* Define local variables. */
    Table table = new Table(uTable.getColumnsCount());
    table.setBorderColor(new Color (
        uTable.getBoxModel().getBorderColor().getRed(),
        uTable.getBoxModel().getBorderColor().getGreen(),
        uTable.getBoxModel().getBorderColor().getBlue()));

    Cell cell;
    for (UTableCell uCell : uTable.getEntries()) {

        /* Define local variable. */
         UParagraph paragraph = uCell.getContent();

        /* If children size larger than 0 and first children is an image. */
        if (paragraph.getChildren().size() > 0 &&
            paragraph.getChildren().get(0) instanceof UImage) {

            UImage image = paragraph.getChildren().get(0);

            /* If segment starts with false, then create a cell with “no”. */
            /* Otherwise, if segment starts with true, then a create cell  */
            /* with “yes”. Otherwise create cell with segment.             */
            if (image.getPath().lastSegment().startsWith("false")) {
                cell = new Cell("no");

            } else {
                if (image.getPath().lastSegment().startsWith("true")) {
                    cell = new Cell("yes");
                } else {
                    cell = new Cell(image.getPath().lastSegment());
                }
            }

            /* Copy background color. */
            if (uCell.getBoxModel().getBackgroundColor() != null) {
                cell.setBackgroundColor(uCell.getBoxModel().getBackgroundColor());
            }
        }
        table.addCell(cell);
    }
    return table;
}
```

S3_1:3 (Revision 1) method chains, no comments

```
/**
 * The images of a unified table are copied to a simplified “yes/no” table.
 */
protected Table copyUTable(UTable uTable)
{
    Table table = new Table(uTable.getColumnsCount());
    table.setBorderColor(new Color (
        uTable.getBoxModel().getBorderColor().getRed(),
        uTable.getBoxModel().getBorderColor().getGreen(),
        uTable.getBoxModel().getBorderColor().getBlue()));

    Cell cell;
    for (UTableCell uCell : uTable.getEntries()) {

        UParagraph paragraph = uCell.getContent();
        if (paragraph.getChildren().size() > 0 &&
            paragraph.getChildren().get(0) instanceof UImage) {

            UImage image = paragraph.getChildren().get(0);

            if (image.getPath().lastSegment().startsWith("false")) {
                cell = new Cell("no");

            } else {
                if (image.getPath().lastSegment().startsWith("true")) {
                    cell = new Cell("yes");
                } else {
                    cell = new Cell(image.getPath().lastSegment());
                }
            }

            if (uCell.getBoxModel().getBackgroundColor() != null) {
                cell.setBackgroundColor(uCell.getBoxModel().getBackgroundColor());
            }
        }
        table.addCell(cell);
    }
    return table;
}
```

S3_2:1 (Revision 1) Fully resolved method chains, good comments

```
/**
 * The images of a unified table are copied to a simplified “yes/no” table.
 */
protected Table writeUTable(UTable uTable)
{
    /* Create a new table with the same size and color as uTable. */
    Table table = new Table(uTable.getColumnsCount());
    BoxModelOption option = uTable.getBoxModel();
    UColor color = option.getBorderColor();
    table.setBorderColor(new Color
        (color.getRed(),color.getGreen(),color.getBlue()));

    /* Go through all entries of uTable and copy image information. */
    Cell cell;
    for (UTableCell uCell : uTable.getEntries()) {

        UParagraph paragraph = uCell.getContent();
        UChildren children = paragraph.getChildren();

        /* Check if there is image information to copy. */
        if (children.size() > 0 && children.get(0) instanceof UImage) {

            UImage image = children.get(0);
            Path path = image.getPath();
            USegment segment = path.lastSegment();


            /* Copy the last segment of the image. False and true */
            /* are transformed to “no” and “yes”, respectively.   */
            if (segment.startsWith("false")) {
                cell = new Cell("no");
            } else {
                if (segment.startsWith("true")) {
                    cell = new Cell("yes");
                } else {
                    cell = new Cell(segment);
                }
            }

            /* Copy background color of uCell. */
            option = uCell.getBoxModel();
            color = option.getBackgroundColor();
            if (color != null) {
                cell.setBackgroundColor(color);
            }
        }
        table.addCell(cell);
    }
    return table;
}
```

S3_2:2 (Revision 1) Fully resolved method chains, bad comments

```
/**
 * The images of a unified table are copied to a simplified “yes/no” table.
 */
protected Table writeUTable(UTable uTable)
{
    /* Define local variables. */
    Table table = new Table(uTable.getColumnsCount());
    BoxModelOption option = uTable.getBoxModel();
    UColor color = option.getBorderColor();
    table.setBorderColor(new Color
        (color.getRed(),color.getGreen(),color.getBlue()));

    Cell cell;
    for (UTableCell uCell : uTable.getEntries()) {

        /* Define local variable. */
        UParagraph paragraph = uCell.getContent();
        UChildren children = paragraph.getChildren();

        /* If children size larger than 0 and first children is an image. */
       if (children.size() > 0 && children.get(0) instanceof UImage) {

            UImage image = children.get(0);
            Path path = image.getPath();
            USegment segment = path.lastSegment();


            /* If segment starts with false, then create a cell with “no”. */
            /* Otherwise, if segment starts with true, then a create cell  */
            /* with “yes”. Otherwise create cell with segment.             */
            if (segment.startsWith("false")) {
                cell = new Cell("no");
            } else {
                if (segment.startsWith("true")) {
                    cell = new Cell("yes");
                } else {
                    cell = new Cell(segment);
                }
            }

            /* Copy background color. */
            option = uCell.getBoxModel();
            color = option.getBackgroundColor();
            if (color != null) {
                cell.setBackgroundColor(color);
            }
        }
        table.addCell(cell);
    }
    return table;
}
```

S3_2:3 (Revision 1) Fully resolved method chains, no comments

```
/**
 * The images of a unified table are copied to a simplified “yes/no” table.
 */
protected Table writeUTable(UTable uTable)
{
    Table table = new Table(uTable.getColumnsCount());
    BoxModelOption option = uTable.getBoxModel();
    UColor color = option.getBorderColor();
    table.setBorderColor(new Color
        (color.getRed(),color.getGreen(),color.getBlue()));

    Cell cell;
    for (UTableCell uCell : uTable.getEntries()) {

        UParagraph paragraph = uCell.getContent();
        UChildren children = paragraph.getChildren();

        if (children.size() > 0 && children.get(0) instanceof UImage) {

            UImage image = children.get(0);
            Path path = image.getPath();
            USegment segment = path.lastSegment();


            if (segment.startsWith("false")) {
                cell = new Cell("no");
            } else {
                if (segment.startsWith("true")) {
                    cell = new Cell("yes");
                } else {
                    cell = new Cell(segment);
                }
            }

            option = uCell.getBoxModel();
            color = option.getBackgroundColor();
            if (color != null) {
                cell.setBackgroundColor(color);
            }
        }
        table.addCell(cell);
    }
    return table;
}
```

Snippet 4

raptor.chess.pgn.PgnUtils.getMove

http://code.google.com/p/raptor-chess-interface/source/browse/tags/v.92/src/raptor/chess/pgn/PgnUtils.java

S4_1:1 method chains, good comments

```
/**
 * Generates a string (“builder”) for a given chess move in PGN (Portable
 * Game Notation). This includes the move number and all NAG annotations.
 */
public static boolean getMove(StringBuilder builder, Move move) {

    boolean result = false;
    int moveNumber = move.getFullMoveCount();

    /* Add move number and move details. */
    builder.append(moveNumber).append(move.isWhitesMove() ? ". " : "... ").
        append(move.toString());

    /* Add all available analysis data (sublines). */
    for (SublineNode subline : move.getSublines()) {
        result = true;
        builder.append(" (");
        appendSubline(builder, subline);
        builder.append(")");
    }

    /* Add all text comments of the move. */
    for (Comment comment : move.getComments()) {
        builder.append(" {").append(comment.getText()).append("}");
    }

    /* Add all Numeric Annotations Glyphs (NAGs) of the move. */
    for (Nag nag : move.getNags()) {
        builder.append(" ").append(nag.getNagString());
    }

    /* Add time taken for the move. */
    builder.append(" {").append(move.getTimeTakenForMove().getText()).append("}");

    return result;
}
```

S4_1:2 method chains, bad comments

```
/**
 * Generates a string (“builder”) for a given chess move in PGN (Portable
 * Game Notation). This includes the move number and all NAG annotations.
 */
public static boolean getMove(StringBuilder builder, Move move) {

    boolean result = false;
    int moveNumber = move.getFullMoveCount();

    /* Add move data. */
    builder.append(moveNumber).append(move.isWhitesMove() ? ". " : "... ").
        append(move.toString());
    /* Add sublines. */
    for (SublineNode subline : move.getSublines()) {
        result = true;
        builder.append(" (");
        appendSubline(builder, subline);
        builder.append(")");
    }

    /* Add comments. */
    for (Comment comment : move.getComments()) {
        builder.append(" {").append(comment.getText()).append("}");
    }

    /* Add NAGs. */
    for (Nag nag : move.getNags()) {
        builder.append(" ").append(nag.getNagString());
    }

    /* Add time. */
    builder.append(" {").append(move.getTimeTakenForMove().getText()).append("}");

    return result;
}
```

S4_1:3 method chains, no comments

```
/**
 * Generates a string (“builder”) for a given chess move in PGN (Portable
 * Game Notation). This includes the move number and all NAG annotations.
 */
public static boolean getMove(StringBuilder builder, Move move) {

    boolean result = false;
    int moveNumber = move.getFullMoveCount();

    builder.append(moveNumber).append(move.isWhitesMove() ? ". " : "... ").
        append(move.toString());

    for (SublineNode subline : move.getSublines()) {
        result = true;
        builder.append(" (");
        appendSubline(builder, subline);
        builder.append(")");
    }

    for (Comment comment : move.getComments()) {
        builder.append(" {").append(comment.getText()).append("}");
    }

    for (Nag nag : move.getNags()) {
        builder.append(" ").append(nag.getNagString());
    }

    builder.append(" {").append(move.getTimeTakenForMove().getText()).append("}");

    return result;
}
```

S4_2:1 resolved method chains, good comments

```
/**
```

```
 * Generates a string (“builder”) for a given chess move in PGN (Portable
 * Game Notation). This includes the move number and all NAG annotations.
 */
public static boolean getMove(StringBuilder builder, Move move) {

    boolean result = false;
    int moveNumber = move.getFullMoveCount();

    /* Add move number and move details. */
    builder.append(moveNumber);
    builder.append(move.isWhitesMove() ? ". " : "... ");
    builder.append(move.toString());

    /* Add all available analysis data (sublines). */
    for (SublineNode subline : move.getSublines()) {
        result = true;
        builder.append(" (");
        appendSubline(builder, subline);
        builder.append(")");
    }

    /* Add all text comments of the move. */
    for (Comment comment : move.getComments()) {
        builder.append(" {");
        builder.append(comment.getText());
        builder.append("}");
    }

    /* Add all Numeric Annotations Glyphs (NAGs) of the move. */
    for (Nag nag : move.getNags()) {
        builder.append(" ");
        builder.append(nag.getNagString());
    }

    /* Add time taken for the move. */
    builder.append(" {");
    TimeTakenForMove timeTaken = move.getTimeTakenForMove();
    builder.append(timeTaken.getText());
    builder.append("}");

    return result;
}
```

S4_2:2 resolved method chains, bad comments

```
/**
 * Generates a string (“builder”) for a given chess move in PGN (Portable
 * Game Notation). This includes the move number and all NAG annotations.
 */
public static boolean getMove(StringBuilder builder, Move move) {

    boolean result = false;
    int moveNumber = move.getFullMoveCount();

    /* Add move data. */
    builder.append(moveNumber);
    builder.append(move.isWhitesMove() ? ". " : "... ");
    builder.append(move.toString());

    /* Add sublines. */
    for (SublineNode subline : move.getSublines()) {
        result = true;
        builder.append(" (");
        appendSubline(builder, subline);
        builder.append(")");
    }

    /* Add comments. */
    for (Comment comment : move.getComments()) {
        builder.append(" {");
        builder.append(comment.getText());
        builder.append("}");
    }

    /* Add NAGs. */
    for (Nag nag : move.getNags()) {
        builder.append(" ");
        builder.append(nag.getNagString());
    }

    /* Add time. */
    builder.append(" {");
    TimeTakenForMove timeTaken = move.getTimeTakenForMove();
    builder.append(timeTaken.getText());
    builder.append("}");

    return result;
}
```

S4_2:3 resolved method chains, no comments

```
/**
 * Generates a string (“builder”) for a given chess move in PGN (Portable
 * Game Notation). This includes the move number and all NAG annotations.
 */
public static boolean getMove(StringBuilder builder, Move move) {

    boolean result = false;
    int moveNumber = move.getFullMoveCount();

    builder.append(moveNumber);
    builder.append(move.isWhitesMove() ? ". " : "... ");
    builder.append(move.toString());

    for (SublineNode subline : move.getSublines()) {
        result = true;
        builder.append(" (");
        appendSubline(builder, subline);
        builder.append(")");
    }

    for (Comment comment : move.getComments()) {
        builder.append(" {");
        builder.append(comment.getText());
        builder.append("}");
    }

    for (Nag nag : move.getNags()) {
        builder.append(" ");
        builder.append(nag.getNagString());
    }

    builder.append(" {");
    TimeTakenForMove timeTaken = move.getTimeTakenForMove();
    builder.append(timeTaken.getText());
    builder.append("}");

    return result;
}
```

Snippet 5

org.eclipse.jface.dialogs.Dialog.shortenText
http://www.docjar.com/html/api/org/eclipse/jface/dialogs/Dialog.java.html

S5_1:1 method chains, good comments

```
/**
 * Shortens the given text textValue so that its width in pixels does
 * not exceed the width of the given control. To do that, shortenText
 * overrides as many as necessary characters in the center of the original
 * text with an ellipsis character (constant ELLIPSIS = "...").
 */
public static String shortenText(String textValue, Control control)
{
    GraphicsContext gc = new GraphicsContext(control);
    int maxExtent = gc.textExtent(textValue).x;
    int maxWidth = control.getBounds().width - 5;

    /* Set start and end points for the center of the text. */
    int length = textValue.length();
    int start = length/2;
    int end = length/2 + 1;

    /* Take away more and more characters in the center of textValue */
    /* and replace them by a single ellipsis character until it is   */
    /* short enough.                                                  */
    while (start >= 0 && end < length) {
        String s1 = textValue.substring(0, start);
        String s2 = textValue.substring(end, length);
        String s = s1 + ELLIPSIS + s2;

        int l = gc.textExtent(s).x;

        /* When the text fits, we stop and return the shortened string. */
        if (l < maxWidth) {
            gc.dispose();
            return s;
        }
        start--;
        end++;
    }
}
```

S5_1:2 method chains, bad comments

```
/**
 * Shortens the given text textValue so that its width in pixels does
 * not exceed the width of the given control. To do that, shortenText
 * overrides as many as necessary characters in the center of the original
 * text with an ellipsis character (constant ELLIPSIS = "...").
 */
public static String shortenText(String textValue, Control control)
{
    /* Define local variables. */
    GraphicsContext gc = new GraphicsContext(control);
    int maxExtent = gc.textExtent(textValue).x;
    int maxWidth = control.getBounds().width - 5;

    /* Define further local variables. */
    int length = textValue.length();
    int start = length/2;
    int end = length/2 + 1;

    /* While the condition holds true,                      */
    /* override characters between s1 and s2 with ELLIPSIS. */
    while (start >= 0 && end < length) {
        String s1 = textValue.substring(0, start);
        String s2 = textValue.substring(end, length);
        String s = s1 + ELLIPSIS + s2;

        int l = gc.textExtent(s).x;

        /* When l is smaller than maxWidth, return s. */
        if (l < maxWidth) {
            gc.dispose();
            return s;
        }
        start--;
        end++;
    }
}
```

S5_1:3 method chains, no comments

```
/**
 * Shortens the given text textValue so that its width in pixels does
 * not exceed the width of the given control. To do that, shortenText
 * overrides as many as necessary characters in the center of the original
 * text with an ellipsis character (constant ELLIPSIS = "...").
 */
public static String shortenText(String textValue, Control control)
{
    GraphicsContext gc = new GraphicsContext(control);
    int maxExtent = gc.textExtent(textValue).x;
    int maxWidth = control.getBounds().width - 5;

    int length = textValue.length();
    int start = length/2;
    int end = length/2 + 1;

    while (start >= 0 && end < length) {
        String s1 = textValue.substring(0, start);
        String s2 = textValue.substring(end, length);
        String s = s1 + ELLIPSIS + s2;

        int l = gc.textExtent(s).x;

        if (l < maxWidth) {
            gc.dispose();
            return s;
        }
        start--;
        end++;
    }
}
```

S5_2:1 resolved method chains, good comments

```
/**
 * Shortens the given text textValue so that its width in pixels does
 * not exceed the width of the given control. To do that, shortenText
 * overrides as many as necessary characters in the center of the original
 * text with an ellipsis character (constant ELLIPSIS = "...").
 */
public static String shortenText(String textValue, Control control)
{
    GraphicsContext gc = new GraphicsContext(control);
    Extent extent = gc.textExtent(textValue);
    int maxExtent = extent.x;
    Bounds bounds = control.getBounds();
    int maxWidth = bounds.width - 5;

    /* Set start and end points for the center of the text. */
    int length = textValue.length();
    int start = length/2;
    int end = length/2 + 1;

    /* Take away more and more characters in the center of textValue */
    /* and replace them by a single ellipsis character until it is   */
    /* short enough.                                                 */
    while (start >= 0 && end < length) {
        String s1 = textValue.substring(0, start);
        String s2 = textValue.substring(end, length);
        String s = s1 + ELLIPSIS + s2;

        extent = gc.textExtent(s);
        int l = extent.x;

        /* When the text fits, we stop and return the shortened string. */
        if (l < maxWidth) {
            gc.dispose();
            return s;
        }
        start--;
        end++;
    }
}
```

S5_2:2 resolved method chains, bad comments

```
/**
 * Shortens the given text textValue so that its width in pixels does
 * not exceed the width of the given control. To do that, shortenText
 * overrides as many as necessary characters in the center of the original
 * text with an ellipsis character (constant ELLIPSIS = "...").
 */
public static String shortenText(String textValue, Control control)
{
    /* Define local variables. */
    GraphicsContext gc = new GraphicsContext(control);
    Extent extent = gc.textExtent(textValue);
    int maxExtent = extent.x;
    Bounds bounds = control.getBounds();
    int maxWidth = bounds.width - 5;


    /* Define further local variables. */
    int length = textValue.length();
    int start = length/2;
    int end = length/2 + 1;

    /* While the condition holds true,                      */
    /* override characters between s1 and s2 with ELLIPSIS. */
    while (start >= 0 && end < length) {
        String s1 = textValue.substring(0, start);
        String s2 = textValue.substring(end, length);
        String s = s1 + ELLIPSIS + s2;

        extent = gc.textExtent(s);
        int l = extent.x;

        /* When l is smaller than maxWidth, return s. */
        if (l < maxWidth) {
            gc.dispose();
            return s;
        }
        start--;
        end++;
    }
}
```

S5_2:3 resolved method chains, no comments

```
/**
 * Shortens the given text textValue so that its width in pixels does
 * not exceed the width of the given control. To do that, shortenText
 * overrides as many as necessary characters in the center of the original
 * text with an ellipsis character (constant ELLIPSIS = "...").
 */
public static String shortenText(String textValue, Control control)
{
    GraphicsContext gc = new GraphicsContext(control);
    Extent extent = gc.textExtent(textValue);
    int maxExtent = extent.x;
    Bounds bounds = control.getBounds();
    int maxWidth = bounds.width - 5;

    int length = textValue.length();
    int start = length/2;
    int end = length/2 + 1;

    while (start >= 0 && end < length) {
        String s1 = textValue.substring(0, start);
        String s2 = textValue.substring(end, length);
        String s = s1 + ELLIPSIS + s2;

        extent = gc.textExtent(s);
        int l = extent.x;

        if (l < maxWidth) {
            gc.dispose();
            return s;
        }
        start--;
        end++;
    }
}
```