

## Interpreting Stack Trace of an Exception

- כשנתקלים בחריגה במהלך ריצת התוכנית, ניתן להשתמש במידע שניתן לנו כדי לזהות את סוג החריגה ואת המיקום בתוכנית שבו היא ארעה.

### Console:

```
Exception in thread "main" java.lang.NullPointerException at
com.example.myproject.Book.getTitle(Book.java:16) at
com.example.myproject.Author.getBookTitles(Author.java:25) at
com.example.myproject.Bootstrap.main(Bootstrap.java:14)
```

### Book.java:

```
public String getTitle() {
    System.out.println(title.toString()); <-- line 16
    return title;
}
```

2

## ממשק משתמש גרפי בעזרת SWT

תוכנה 1 בשפת Java

1

## SWT

- בנויה על העיקרון של publish/subscribe
- אלמנטים בסיסיים (Widgets) מייצרים אירועים (Events) שאליהם נרשמים מאזינים (Listener)
- דוגמא 1: משתמש לוחץ על כפתור, שמייצר אירוע לחיצה. מאזין שנרשם לאירוע הלחיצה של הכפתור יכול לשנות את כותרת החלון
- דוגמא 2: משתמש סוגר את החלון, שמייצר אירוע סגירת חלון. מאזין שנרשם לאירוע סגירת החלון פותח חלון ששואל את המשתמש אם הוא רוצה לשמור את השינויים לפני שיצא מהתוכנית.
- ה Widgets וה- Events מוגדרים ע"י כותבי הספרייה
- מאזינים נכתבים ע"י המשתמש
- מאפשר תגובות שונות לאירועים כתלות באפליקציה

4

## Interpreting Stack Trace of an Exception

- דוגמא נוספת:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
at java.util.Arrays.copyOfOf(Unknown Source)
at java.lang.AbstractStringBuilder.expandCapacity(Unknown Source)
at java.lang.AbstractStringBuilder.ensureCapacityInternal(Unknown Source)
at java.lang.AbstractStringBuilder.append(Unknown Source)
at java.lang.StringBuilder.append(Unknown Source)
at SmallTestMultiCollections.testOrder(SmallTestMultiCollections.java:56)
at SmallTestMultiCollections.main(SmallTestMultiCollections.java:34)
```

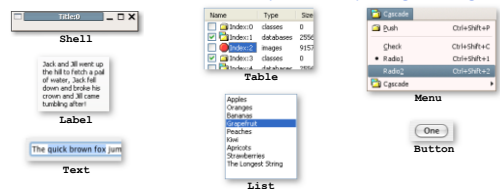
3

## עוד על Widgets

- ביצירת Widget נגדיר
  - את ה"הורה" שלו
  - את הסגנון שלו
- הורה הוא Widget היוצר מ-Composite, מה שאומר שניתן להוסיף אליו Widgets אחרים
- לדוגמא, כפתור שהורה שלו הוא טאב שהורה שלו הוא חלון – יופיע כפתור בתוך הטאב שבהלון
- ה-Widget מתווסף להורה בזמן הקריאה לבנאי
- עבור סגנונות קיימים קבועים במחלקה SWT
- נראה בהמשך

## SWT Widgets

- אבני הבניין של ממשקים גרפיים
- מוגדרים ב [org.eclipse.swt.widgets](http://www.eclipse.org/swt/widgets/)
- תת-טיפוסים של המחלקה האבסטרקטית Widget (קישור לתיעוד)
- האתר של SWT מכיל דוגמאות קצרות (snippets) לשימוש בכל Widget <http://www.eclipse.org/swt/widgets/>



5



## אז מה היה לנו כאן?

- `Display display = Display.getDefault();`  
- `Display` - מקשר בין SWT לתצוגת מערכת ההפעלה (למשל, המסך)
- `Shell shell = new Shell(display);`  
- `Shell` - חלון, שימו לב שיצירת חלון לא פותחת אותו עדיין.
- `shell.setLayout(new FillLayout(SWT.VERTICAL));`  
לכל `Composite` ניתן להוסיף `layout` שיגדיר כיצד `Widgets` מסודרים בתוכו
- `FillLayout` - ה- `Widgets` ממלאים את ה- `Composite`
- `SWT.VERTICAL` - ה- `Widgets` מסודרים בצורה אנכית לפי סדר הוספתם ל- `Composite`
- `RowLayout` - דומה ל- `FillLayout`, אבל ה- `Widgets` שומרים על גודל קבוע
- `GridLayout` - מסדר את ה- `Widgets` בגריד (לפי עמודות ושורות)
- ומה קורה אם אין `layout` בכלל??

8



## כפתור

```
public class ShellWithButton1 {
    public static void main(String[] args) {
        Display display = Display.getDefault();
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout(SWT.VERTICAL));
        shell.setText("example1");

        Button ok = new Button(shell, SWT.PUSH);
        ok.setText("Push Me!");

        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

7

## לולאת האירועים (Event loop)

```
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
```

- קוד סטנדרטי שמופיע כמעט בכל תכנית `SWT`
- כל עוד החלון הראשי של התכנית לא נסגר - טפל באירוע הבא בתור ובדוק האם קיים אירוע נוסף
- אם לא קיים - היכנס למצב שינה עד לקבלת אירוע נוסף
- בסוף משחררים את כל המשאבים שהוקצו ע"י קריאה ל- `dispose`
- כאשר משחררים אלמנט גרפי, כל הצאצאים שלו גם משתחררים
- כאשר משחררים את `Display`, כל המשאבים הגרפיים משתחררים

10



## אז מה היה לנו כאן?

```
shell.setText("example1");

Button ok = new Button(shell, SWT.PUSH);
ok.setText("Push Me!");

shell.pack();

shell.open();
```

- `setText` - משתנה בהתאם לטיפוס ה- `Widget`. בחלון - קובע את הכותרת, בכפתור לחיצה (`PUSH`) - קובע מה כתוב על הכפתור.
- `pack` - גורם לאובייקט גרפי לחשב ולהתאים את גודלו, למשל בהתאם ל- `layout`, ל- `Widgets` שבתוכו וכו'

• פתיחת החלון

9

## הוספת טיפול בארועים

- הכפתור לא מגיב ללחיצות. יש להוסיף טיפול באירוע "לחיצה"
- עלינו לממש מאזין המקבל שמטפל באירוע ולהרשם על הווידג'ט המתאים.
- כיצד נדע אילו אירועים מייצר `Widget` מסוים? איזה ממשק עלינו לממש?
- נסתכל בתיעוד
- התיעוד של `Button`:

Events:  
Selection

### Method Summary

```
void addSelectionListener(SelectionListener listener)
Add the listener to the collection of listeners who will be notified when the
of the messages defined in the SelectionListener interface.
```

12

## הוספת טיפול בארועים

- הכפתור לא מגיב ללחיצות. יש להוסיף טיפול באירוע "לחיצה"
- על המחלקה המטפלת לממש את הממשק `SelectionListener`
- על הכפתור עצמו להגדיר מי העצם (או העצמים) שיטפלו באירוע
- כמה גישות אפשריות:
  - הגדרת מחלקה שירשת מכפתור
  - מחלקה שמכילה כפתור כאחד משדותיה
  - יצירת מחלקה עצמאית שתטפל באירועי הלחיצה
- המימושים שנראה היום משתייכים לגישה השלישית

11

## טיפול בארועים במחלקה נפרדת

```

Display display = Display.getDefault();
Shell shell = new Shell(display);
shell.setLayout(new FillLayout(SWT.VERTICAL));
shell.setText("example2");

Button ok = new Button(shell, SWT.PUSH);
ok.setText("Push Me!");
ok.addSelectionListener(new ButtonHandler());

shell.pack();
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
    
```

14

## טיפול בארועים במחלקה נפרדת

```

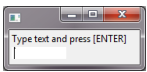
public class ButtonHandler
    implements SelectionListener {

    public void widgetSelected(SelectionEvent e) {
        if (e.getSource() instanceof Button) {
            Button b = (Button) e.getSource();
            b.setText("Thanks!");
        }
    }

    public void widgetDefaultSelected(SelectionEvent e){
        // TODO Auto-generated method stub
    }
}
    
```

13

## מחלקה פנימית



```

public class ShellWithLabelAndTextField1 {
    private Label label;
    private Text text;

    public static void main(String[] args) {...}
    public void createShell() {...}

    public class InnerHandler implements KeyListener {
        public void keyPressed(KeyEvent e) {
            if (e.character == SWT.CR) {
                label.setText(text.getText());
                text.setText("");
            }
        }

        public void keyReleased(KeyEvent e) {
            // TODO Auto-generated method stub
        }
    }
}
    
```

המחלקה הפנימית נגשת לשדות הפרטים של המחלקה העוטפת

16

## טיפול בארועים במחלקה נפרדת

- לעיתים הטיפול באירוע דורש הכרות אינטימית עם המקור (כדי להימנע מחשיפת המבנה הפנימי של המקור)
- שימוש במחלקה פנימית יוצר את האינטימיות הדרושה
- בדוגמה הבאה נרצה לעדכן תווית על סמך קלט מהמשתמש
- דרושה הכרות לא רק עם יוצר האירוע (Text) אלא גם עם חלקים אחרים במבנה

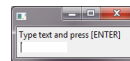
15

## שימוש במחלקות אנונימיות

- בדרך כלל נזדקק רק למאזין יחיד לכל אירוע
- נשתמש במחלקה לוקאלית אנונימית
- **תזכורת:** `new className([argument-list]) {classBody}`
- יצירת מופע חדש של מחלקה ללא שם, שטרם הוגדרה, שיושרת באופן אוטומטי מ `className`
- יצירת מופע חדש של מחלקה ללא שם, שטרם הוגדרה, שממשת באופן אוטומטי את `interfaceName`

18

## מחלקה פנימית



```

public class ShellWithLabelAndTextField1 {
    private Label label;
    private Text text;

    public static void main(String[] args) {
        ShellWithLabelAndTextField1 shell = new ShellWithLabelAndTextField1();
        shell.createShell();
    }

    public void createShell() {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new RowLayout(SWT.VERTICAL));

        label = new Label(shell, SWT.CENTER);
        label.setText("Type text and press [ENTER]");

        text = new Text(shell, SWT.LEFT);
        text.addKeyListener(new InnerHandler());
        // pack(), open(), while ... Dispose()
    }
}
    
```

17

## שימוש ב Adapter

```
public class ShellWithLabelAndTextField2 {
    private Label label;
    private Text text;

    public static void main(String[] args) {...}

    public void createShell() {
        text.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                if (e.character == SWT.CR) {
                    label.setText(text.getText());
                    text.setText("");
                }
            }
        });

        // pack(), open(), while ... Dispose()
    }
}
```

```
public class KeyAdapter implements KeyListener {
    @Override
    public void keyPressed(KeyEvent arg0) {
        //do nothing
    }
    @Override
    public void keyReleased(KeyEvent arg0) {
        // do nothing
    }
}
```

20

## מחלקה אנונימית

```
public class ShellWithLabelAndTextField2 {
    private Label label;
    private Text text;

    public static void main(String[] args) {...}

    public void createShell() {
        text.addKeyListener(new KeyListener() {
            public void keyPressed(KeyEvent e) {
                if (e.character == SWT.CR) {
                    label.setText(text.getText());
                    text.setText("");
                }
            }
            public void keyReleased(KeyEvent e) {
                // TODO Auto-generated method stub
            }
        });
        // pack(), open(), while ... Dispose()
    }
}
```

סוגר סוגריים של המתודה addKeyListener()

19

## דוגמא נוספת – רשימת משימות

```
public class TaskList1 {
    private static final Display display =
        Display.getDefault();
    private static Shell shell;

    public static void main(String[] args) {
        TaskList1 taskList = new TaskList1();
        taskList.createShell();
        taskList.runApplication();
    }

    private void createShell() {
        shell = new Shell(display);
        shell.setText("My Tasks");
    }

    private void runApplication() {
        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

- נכתוב אפליקציה פשוטה המאפשרת להזין משימות בשדה טקסט, ללחוץ על כפתור ולהוסיף את הטקסט שהוזן לרשימה.
- נתחיל מבנית שלד האפליקציה



22

## המחלקה SWT

- מוגדרת ב `org.eclipse.swt.SWT`
- אוסף של קבועים:
  - אירועים – `MouseDown, FocusIn, Close, Activate` ...
  - צבעים – `COLOR_BLUE, COLOR_BLACK` ...
  - תווים – `ESC, DEL, CR` ...
  - אירוע מקשים – `END, ARROW_DOWN` ...
  - עיצובים
- ניתן להוסיף מס' קבועים ע"י שימוש באופרטור (bitwise OR)
  - `SWT.V_SCROLL | SWT.H_SCROLL | SWT.BORDER`

21

## הוספת פונקציונליות

```
//a text field to enter a task
final Text input = new Text(shell, SWT.LEFT);

//a button to add a task to the list
Button add = new Button(shell, SWT.PUSH);
add.setText("Add");

// the list
final List list = new List(shell, SWT.BORDER);

// the action to perform when pressing the button
add.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        String text = input.getText();
        //add the task to the list, if the input text is not empty
        if (text != null && text.length() > 0) {
            list.add(text);
            input.setText("");
        }
    }
});
```

- כעת, נוסיף את הפעולה של לחיצה על כפתור – תוך שימוש במחלקה אנונימית.
- הכפתור צריך להכיר את שדה הטקסט ואת הרשימה, לכן נשנה אותם ל-`final`
- עכשיו זה עובד!

24

## הוספת Widgets

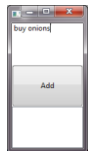
```
private void createShell() {
    shell = new Shell(display);
    shell.setText("My Tasks");
    shell.setLayout(new FillLayout(SWT.VERTICAL));

    //a text field to enter a task
    Text input = new Text(shell, SWT.LEFT);

    //a button to add a task to the list
    Button add = new Button(shell, SWT.PUSH);
    add.setText("Add");

    //the list
    List list = new List(shell, SWT.BORDER);
}
```

- נוסף Layout בסיסי ל-Shell, וכן שדה טקסט, כפתור ורשימה
- כרגע אין מאזינים לאף פעולה



23

## עיצוב החלון

```

shell = new Shell(display);
shell.setText("My Tasks");
shell.setLayout(new GridLayout(2, false));

// a text field to enter a task
final Text input = new Text(shell, SWT.LEFT);
input.setLayoutData(new GridData(
    GridData.FILL_HORIZONTAL));

// a button to add a task to the list
Button add = new Button(shell, SWT.FLAT);
add.setText("Add");

// the list
final List list = new List(shell, SWT.BORDER);
GridData listGridData = new GridData(GridData.FILL_BOTH);
listGridData.horizontalSpan = 2;
listGridData.widthHint = 300;
listGridData.heightHint = 300;
list.setLayoutData(listGridData);
    
```

שני טורים ברוחב לא אחיד

- משתמש ב-GridLayout
- מוסיף אלמנטים לגריד משמאל לימין ומלמעלה למטה לפי הסדר
- מאפשר הוספת GridData שמציין איך כל Widget יתנהג בגריד

שדה הטקסט נמתח לרוחב

הרשימה נמתחת לרוחב ולאורך. תוספת שני טורים, וגודלה המומלץ 300x300 פיקסלים

26

## נטפל בעיצוב החלון



- תחילה, נתכנן איך היינו רוצים שהחלון ייראה

- וכיצד הוא ישתנה בשינוי גודל החלון



25

## תוספת - מחיקת משימות מהרשימה

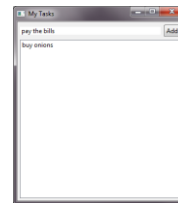
```

// the action to perform when DELETE is pressed on the list
list.addKeyListener(new KeyAdapter() {
    @Override
    public void keyPressed(KeyEvent e) {
        if (e.character == SWT.DEL) {
            int selectionIndex = list.getSelectionIndex();
            //if a list item is selected, delete it from the list
            if (selectionIndex >= 0)
                list.remove(selectionIndex);
        }
    }
});
    
```

- נוסף KeyListener לרשימה שיאזין ללחיצה על הפריט
- מ-Widget הרשימה ניתן לקבל את האינדקס של הפריט הנבחר ולמחוק את הפריט

28

## עיצוב החלון



- עכשיו האפליקציה שלנו נראית כך

- עוד תוספות?
  - סימון משימה כ"הושלמה"
  - הוספת תאריך יצירת המשימה
  - הוספת תאריך יעד לביצוע המשימה
  - עריכת משימה
  - מחיקת משימה
  - ...

27