

מה עושים היום?

- קריאת Stack Trace של חריגה
- hashCode ו- equals
- ממשק משתמש גרפי

2

ממשק משתמש גרפי בעזרת SWT

תוכנה 1 בשפת Java

1

Interpreting Stack Trace of an Exception

- דוגמא נוספת:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
at java.util.Arrays.copyOf(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)
```

4

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;
}
```

3

מה יודפס?

```
public class Name {
    private String first, last;
    ...

    public static void main(String[] args) {
        Name name1 = new Name("Mickey", "Mouse");
        Name name2 = new Name("Mickey", "Mouse");
        System.out.println(name1.equals(name2));  false

        List<Name> names = new ArrayList<Name>();
        names.add(name1);
        System.out.println(names.contains(name2));  false
    }
}
```

6

תזכורת: המחלקה Object

```
package java.lang;

public class Object {
    public final native Class<?> getClass();

    public native int hashCode();

    public boolean equals(Object obj) {
        return (this == obj);
    }

    protected native Object clone() throws CloneNotSupportedException;

    public String toString() {
        return getClass().getName() + "@" +
            Integer.toHexString(hashCode());
    }
    ...
}
```

5

החזרה של equals

- רפלקסיבי
 - `x.equals(x)` יחזיר `true`
- סימטרי
 - `x.equals(y)` אמ"מ `y.equals(x)` יחזיר `true`
- טרנזיטיבי
 - אם `x.equals(y)` מחזיר `true` וגם `y.equals(z)` מחזיר `true` אז `x.equals(z)`
- עקבי
 - סדרת קריאות ל `x.equals(y)` תחזיר `true` (או `false`) באופן עקבי אם מידע שדרוש לצורך ההשוואה לא השתנה
- השוואה ל `null`
 - `x.equals(null)` תמיד תחזיר `false`

8

הבעיה

- רצינו השוואה לפי תוכן אבל לא דרסנו את `equals`
- מימוש ברירת המחדל הוא השוואה של מצביעים

```
public class Object {
    ...
    public boolean equals(Object obj) {
        return (this == obj);
    }
    ...
}
```

7

טעות נפוצה

- להגדיר את הפונקציה `equals` כך:

```
public boolean equals(Name other) {
    return first.equals(other.first) &&
        last.equals(other.last);
}
```

- זו אינה דריסה (`overriding`) אלא העמסה (`overloading`)
- שימוש ב `@Override` יפתור את הבעיה

10

מתכון ל `equals`

```
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Name other = (Name) obj;
    return first.equals(other.first) &&
        last.equals(other.last);
}
```

1. ודאו כי הארגומנט אינו מצביע לאובייקט הנוכחי
2. ודאו כי הארגומנט אינו `null`
3. ודאו כי הארגומנט הוא הטיפוס המתאים להשוואה
4. המירו את הארגומנט לטיפוס הנוכחי
5. לכל שדה "משמעותי", בדיקו ששדה זה בארגומנט תואם לשדה באובייקט הנוכחי

9

כמעט

```
public class Name {
    ...
    @Override public equals(Object obj) {
        ...
    }
}

public static void main(String[] args) {
    Name name1 = new Name("Mickey", "Mouse");
    Name name2 = new Name("Mickey", "Mouse");
    System.out.println(name1.equals(name2));
}

Set<Name> names = new HashSet<Name>();
names.add(name1);
System.out.println(names.contains(name2));
}
```

יודפס true

יודפס false

12

אז הכל בסדר?

```
public class Name {
    ...
    @Override public equals(Object obj) {
        ...
    }
}

public static void main(String[] args) {
    Name name1 = new Name("Mickey", "Mouse");
    Name name2 = new Name("Mickey", "Mouse");
    System.out.println(name1.equals(name2));
}

List<Name> names = new ArrayList<Name>();
names.add(name1);
System.out.println(names.contains(name2));
}
```

יודפס true

יודפס true

11

החזרה של hashCode

- **עקביות**
 - מחזירה אותו ערך עבור כל הקריאות באותה ריצה, אלא אם השתנה מידע שבשימוש בהשוואת equals של המחלקה
- **שוויון**
 - אם שני אובייקטים שווים לפי הגדרת equals אזי hashCode תחזיר ערך זהה עבורם
- **חוסר שוויון**
 - אם שני אובייקטים אינם שווים לפי equals לא מובטח ש hashCode תחזיר ערכים שונים
 - החזרת ערכים שונים יכולה לשפר ביצועים של מבני נתונים המבוססים על hashing (לדוגמא, HashSet ו HashMap)

14

hashCode ו equals

חובה לדרוש את hashCode בכל מחלקה שדורסת את equals!

13

תמיכה באקליפס

- אקליפס תומך ביצירה אוטומטית (ומשולבת) של hashCode ו equals בתפריט Source ניתן למצוא Generate hashCode() and equals()

16

מימוש hashCode

```
@Override public int hashCode() {  
    return 31 * first.hashCode() + last.hashCode();  
}
```

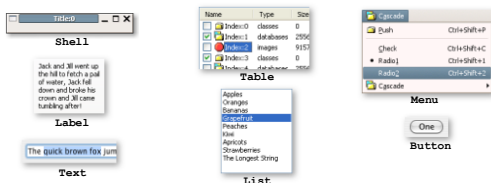
- השתדלו לייצר hash כך שלאובייקטים שונים יהיה ערך hash שונה
- המימוש החוקי הגרוע ביותר (לעולם לא לממש כך!)

```
@Override public int hashCode() {  
    return 42;  
}
```

15

SWT Widgets

- אבני הבניין של ממשקים גרפיים מוגדרים ב org.eclipse.swt.widgets
- תת-טיפוסים של המחלקה האבסטרקטית Widget



18

SWT

- בניה על העיקרון של publish/subscribe
- אלמנטים בסיסיים (Widgets) מייצרים אירועים (Events) שאליהם נרשמים מאזינים (Listener)
- ה Widgets וה- Events מוגדרים ע"י כותבי הספרייה מאזינים נכתבים ע"י המשתמש
- תגובות שומות לאירועים זהים כתלוי באפליקציה

17

הוספת טיפול בארועים

- הכפתור לא מגיב ללחיצות. יש להוסיף טיפול בארוע "לחיצה"
- על המחלקה המטפלת לממש את הממשק `SelectionListener`
- על הכפתור עצמו להגדיר מי העצם (או העצמים) שיטפלו בארוע
- כמה גישות אפשריות:
 - הגדרת מחלקה שירשת מכפתור
 - מחלקה שמכילה כפתור אחד משדותיה
 - יצירת מחלקה עצמאית שתטפל בארועי הלחיצה
- לכל אחת מהאפשרויות יתרונות וחסרונות שידונו בהמשך

20

כפתור



```
public class ShellWithButton {
    public static void main(String[] args) {
        Display display = Display.getDefault();
        Shell shell = new Shell (display);

        Button ok = new Button (shell, SWT.PUSH);
        ok.setText ("Push Me!");
        ok.setLocation (0,0);
        ok.setSize (100,30);

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

19

טיפול בארועים במחלקה נפרדת

```
public class ButtonHandler
    implements SelectionListener {

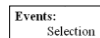
    public void widgetSelected (SelectionEvent e) {
        if (e.getSource () instanceof Button) {
            Button b = (Button) e.getSource ();
            b.setText ("Thanks!");
        }
    }

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

22

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 ווידג'ט? איזה ממשק עלינו לממש?
 - נסתכל בתיעד



Method Summary

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

21

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

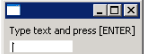
24

טיפול בארועים במחלקה נפרדת

```
public class ShellWithButton {
    public static void main(String[] args) {
        Display display = Display.getDefault();
        Shell shell = new Shell (display);
        Button ok = new Button (shell, SWT.PUSH);
        ok.addSelectionListener (new ButtonHandler ());
        ok.setText ("Push Me!");
        ok.setLocation (0,0);
        ok.setSize (100,30);
        shell.pack ();
        shell.open ();
        while (!shell.isDisposed ()) {
            if (!display.readAndDispatch ()) display.sleep ();
        }
        display.dispose ();
    }
}
```

23

מחלקה פנימית



```

public class ShellWithLabelAndTextField {
    private Label l;
    private Text t;

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

    public void createShell() {
        Display display = new Display();
        Shell shell = new Shell(display);

        GridLayout gl = new GridLayout();
        shell.setLayout(gl);

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


        t = new Text(shell, SWT.LEFT);
        t.addKeyListener(new InnerHandler());

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

```

26

מחלקה פנימית



```

public class ShellWithLabelAndTextField {
    private Label l;
    private Text t;

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

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

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

```

25

המחלקה הפנימית נוגשת לשדות הפרטיים של המחלקה העוטפת

מחלקה אנונימית

```

public class ShellWithLabelAndTextField {
    ...
    public void createShell() {
        ...
        t.addKeyListener(new KeyListener() {
            public void keyPressed(KeyEvent e) {
                if (e.character == SWT.CR) {
                    l.setText(t.getText());
                    t.setText("");
                }
            }

            public void keyReleased(KeyEvent e) {
                // TODO Auto-generated method stub
            }
        });
        // pack(), open(), while ... Dispose()
    }
}

```

28

סוגר שורשים של המתודה addKeyListener()

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, שיורשת באופן אוטומטי מ `className`
- יצירת מופע חדש של מחלקה ללא שם, שטרם הוגדרה, שמממשת באופן אוטומטי את `interfaceName`

27

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`, ...

30

שימוש ב Adapter

```

public class ShellWithLabelAndTextField {
    ...
    public void createShell() {
        ...
        t.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                if (e.character == SWT.CR) {
                    l.setText(t.getText());
                    t.setText("");
                }
            }
        });
        // pack(), open(), while ... Dispose()
    }
}

```

29

סוגר שורשים של המתודה addKeyListener()