שפת Java למתכנתי ++C - חלק שני

ניר אדר

מסמך זה הורד מהאתר http://underwar.livedns.co.il.
אין להפיץ מסמך זה במדיה כלשהי, ללא אישור מפורש מאת המחבר.
מחבר המסמך איננו אחראי לכל נזק, ישיר או עקיף, שיגרם עקב השימוש במידע המופיע במסמך, וכן לנכונות התוכן של הנושאים המופיעים במסמך. עם זאת, המחבר עשה את מירב המאמצים כדי לספק את המידע המדויק והמלא ביותר.

כל הזכויות שמורות ל**ניר אדר**

Nir Adar
Email: underwar@hotmail.com
Home Page: http://underwar.livedns.co.il

אנא שלחו תיקונים והערות אל המחבר.

1. EXCPTIONS

כל תוכניות מורכבת צריכה מנגנון בדיקת שגיאות מסוים – פונקציות שונות (כגון פונקציות המקצות זיכרון, או פונקציות המנסות לבצע פעולות שונות העלולות להיכשל) צריכות דרך לדווח לקורא להם האם הקריאה לפונקציה הצליחה או נכשלה. כמו כן צריך מנגנון שיתמוך בניתוח השגיאה ותגובה בהתאם.
בשפת C השתמשנו בערכים המוחזרים מהפונקציות כדרך לבדוק את תקינות פעולת הפונקציה.
מנגנון זה הינו מוגבל בתחומים שונים – הוא מסבך את הקוד, (קשה להבחין בין הקוד לבין בדיקת התקלות), במקרים מסוימים, בעיקר עבור הקצאות זיכרון דינאמיות מורכבות, מסובך לשחרר את הזיכרון שהוקצה. כמו כן, לא כל פונקציה יכולה לבחור ערך שייעוד לשגיאה. ייתכנו פונקציות שכל ערך מוחזר יכול להיות גם ערך לגיטימי.
לפיכך ++C ובעקבותיה גם Java תומכות במנגנון exceptions.

הרעיון הראשוני, בדומה ל-++C, הוא הפרדה בין קטע הקוד לקטע בדיקת השגיאות.
את קטע הקוד שאנו כותבים אנו שמים בתוך בלוק try.
במידה ומתרחשת שגיאה, השגיאה לא תוחזר מהפונקציה כערך מוחזר על ידי המילה השמורה return אלא על ידי מנגנון ה-exceptions בעזרת המילה throw.
Exception היא שגיאה אקטיבית – אם לא נטפל בה, התוכנית תפסיק את פעולתה.

דוגמא:

```
class MainClass
{
	static double Div(double fNumber1, double fNumber2) throws 
Exception
	{
		if (fNumber2 == 0) throw new Exception("Divide by zero 
error.");
		return fNumber1 / fNumber2;
	}

	static int Main()
	{

		try
```

```
		{
			System.out.println(Div(10, 2));
			System.out.println(Div(4, 0));
			System.out.println(Div(8, 2));
		}
		catch (Exception e)
		{
			System.out.println(e.getMessage());
		}

		return 0;
	}
}
```

הפונקציה Div מחלקת שני מספרים ומחזירה את התוצאה. במידה והתחרש חילוק ב-0 היא שולחת Exception.

ב-Main אנו קוראים שלוש פעמים לפונקציה Div.

בפעם השניה נשלח Exception, ואנו קופצים אל בלוק ה-catch שמטרתו לטפל בבעיה, וכלל לא מגיעים אל הקריאה השלישית.

ב-++C יכולנו לשלוח Exception מכל סוג. ב-Java אנו יכולים לשלוח רק אובייקטים מסוג Exception או שנגזרים ממחלקה זו (למעשה המחלקה ממנה נגזרת Exception היא המחלקה Throwable אולם הדיון בכך הוא מחוץ למסגרת מסמך זה). אנו יוצרים אובייקט חדש מסוג Exception ושולחים אותו.

שליחת exception נעשית בעזרת המילה השמורה throw.

כמו כן, חובה עלינו לציין את כל סוגי ה-Exceptions האפשריים שפונקציה יכולה לשלוח על ידי שימוש במילה השמורה throws בהגדרת הפונקציה, ולאחריה הסוגים השונים של ה-Exceptions הניתנים לזריקה, מופרדים על ידי פסיקים.

על מנת לבדוק ולתפוס שגיאות, אנו משתמשים במבנה הבא:

```
	try
	{

	}
	catch (Exception e)
	{

	}
```

כאשר קוראת בפונקציה כלשהי exception, המחסנית שלה משתחררת, והבקרה מועברת אל בלוק ה-catch שקרא לה. אם אין בלוק כזה, אנו יוצאים גם מהפונקציה הקוראת כלפי מעלה במחסנית, עד שאנו מוצאים קוד המטפל בשגיאה.

שיטות שימושיות של המחלקה Exception:

getMessage() – מחזירה מחרוזת המכילה את תיאור הסיבה שגרמה ל-Exception.

getStackTrace() – מחזיר מחרוזת המכילה תיאור של שרשרת הקריאות שהאובייקט Exception עבר עד שהגיע לבלוק ה-catch.

אם נרצה, נוכל ליצור מחלקות שיגזרו מ-Exception, ולתפוס אותם במקום אובייקט מסוג Exception. בדומה ל-C++, אנו מסוגלים לשים מספר בלוקים של catch אחד אחרי השני, לתפיסת שגיאות מסוגים שונים.

לעיתים נרצה לבצע פעולות כלשהן, גם אם קרא exception.

במקרה כזה, נוסיף לאחר ה-catch בלוק finally. בלוק זה יתבצע תמיד, בין אם קרא exception או לא, ואפילו אם הייתה פעולת return בתוך בלוק ה-try.

דוגמא:

```
try
{
	System.out.println(Div(10, 2));
	// System.out.println(Div(4, 0));
	System.out.println(Div(8, 2));
	return 0;
}
catch (Exception e)
{
	System.out.println(e.getMessage());
}
finally
{
	System.out.println("The End...");
}
```

2. קלט / פלט

<u>רקע</u>

לעתים קרובות תוכנית צריכה לקבל נתונים מגורם חיצוני או להוציא נתונים לגורם כזה.
במקרים רבים הגישה לנתונים הינה גישה לנתונים עוקבים (ולא גישה אקראית). הפתרון ש-Java נתנה לגישה כזו, בדומה ל-C++, היא גישה בעזרת הפשטה בשם זרם (stream). התוכנית פותחת זרם של נתונים המקושר לגורם החיצוני המתאים וקוראת או כותבת את הנתונים באופן סדרתי.

החבילה java.io מכילה אוסף גדול של מחלקות הממששות זרמים שונים. ניתן לחלק את מחלקות אלה לזרמים של תווים (טקסט) ולזרמים של בתים (כל שאר סוגי הנתונים).
עבור תווים, כל זרמי הקריאה יורשים מהמחלקה Reader וכל זרמי הכתיבה יורשים מהמחלקה Writer.
דוגמאות: FileWriter, FileReader, StringWriter, StringReader, BufferedWriter, BufferedReader.
עבור בים, כל זרמי הקריאה יורשים מהמחלקה InputStream וכל זרמי הכתיבה יורשים מהמחלקה OutputStream.
דוגמאות: FileInputStream, FileOutputStream ועוד.

<u>דוגמא</u>

התוכנית הבאה מדגימה העתקה של קובץ:

```
import java.io.*;

public class FileCopy
{
      public static void main(String[] args)
      {
            File inputFile = new File("InputFile.txt");
            File outputFile = new File("OutputFile.txt");
```

```
            FileReader in = new FileReader(inputFile);
            FileWriter in = new FileWriter(outputFile);

            int c;
            while ( (c = in.read()) != -1) out.write(c);

            in.close();
            out.close();
      }

}
```

כתיבת אובייקטים לזרם

הדוגמא הבאה תדגים כיצד אנו יכולים לכתוב אובייקטים אל זרם:

```
FileOutputStream out = new FileOutputStream("theTime");
ObjectOutputStream s = new ObjectOutputStream(out);
s.writeObject("Today");
s.writeObject(new Date());
s.flush();
```

כדי לקרוא את הנתונים שכתבנו, נכתוב את הקוד הבא:

```
FileInputStream in = new FileInputStream("theTime");
ObjectInputStream s = new ObjectInputStream(in);
String today = (String)s.readObject();
Date date = (Date)s.readObject();
```

כדי שנוכל לשמור גם מחלקות אותן אנחנו יוצרים, נדרוש שהיא תגזר מהממשק Serializable, כלומר:

```
public class MySerializableClass implements Serializable
{
    ...
}
```

3. SWING

הקדמה

SWING זהו שם לאוסף מחלקות ב-Java המספקות כלים ליצירת ממשק משתמש (GUI).
SWING הינה חלק מאוסף מחלקות רחב יותר, המכונה Java Foundation Classes ñ JFC העוסקים כולם בגרפיקה והצגת המידע אל המשתמש.

SWING מספק **רכיבי ממשק משתמש** (components) רבים כגון כפתורים, רשימות, תפריטים ועוד.
בנוסף הוא מספק **מכולות** (containers) כגון חלונות וסרגלי כלים.

ניתן לחלק את המחלקות של SWING לשתי קבוצות:

- מחלקות המייצגות מרכיבי ממשק משתמש. מחלקות אלה יורשות מהמחלקה JComponent, ושמן מתחיל באות J: JButton, JList, JTextField וכו'.
- מחלקות המספקות שירותים למחלקות ה-GUI.

מכולות

מכולות הם מחלקות הנגזרות מהמחלקה java.awt.Container. מכולות הם רכיבים המסוגלים להכיל רכיבים אחרים. המכולות משתמשות ב-layout manager כדי למקם את הרכיבים השונים עליהן.
רכיבים חדשים מוספים על ידי השיטה add של המכולה. ישנן מספר שיטות add חופפות. הבחירה באיזו שיטה להשתמש נעשית בהתאם ל-layout manager בו אנו משתמשים.
כיוון ש-container נחשב גם ל-component ניתן להוסיף על container נתון containers אחרים בתור components, ועליהם להוסיף containers או components אחרים וכן הלאה.

Top-Level Containers

לכל תוכנית אשר משתמשת בממשק משתמש SWING יש לפחות רכיב top-level container אחד.
מכולת top-level מספקת את התמיכה שרכיבי SWING צריכים על מנת לבצע את הציור שלהם ואת ניהול האירועים שלהם.
SWING מספק שלוש מכולות top-level:

- JFrame – המשמש ליצירת חלון התוכנית המרכזי.
- JDialog – המשמש ליצירת חלונות נוספים לתוכנית.
- JApplet – המאפשר ליצור תצוגות תחת חלון דפדפן.

כדי שרכיב יופיע על המסך הוא חייב להיות חלק מהירככית הכלה שבראש top-level container.
ב-top level container קיימות מספר שכבות. אחת השכבות נקראת בשם content pane.
שכבה זו מכילה את הרכיבים אשר יהיו נראים על ממשק המשתמש. מסיבה זו הפעלת המתודה add כדי להוסיף GUI components תיעשה על האובייקט שמייצג את שכבת ה-content של ה-top level container.
כדי להוסיף ל-top level container רכיב GUI כגון JButton או JPanel למשל יש להשיג reference לאובייקט שמייצג את שכבת ה-content. השיטה getContentPane() השייכת ל-container תחזיר לנו reference מטיפוס Container לאובייקט שמייצג את שכבת ה-content.

דוגמא:

```
JFrame f = new JFrame();
Container contentPane = f.getContentPane();
contentPane.add(new JButton());
```

JFrame

חלון הממומש כמופע של המחלקה JFrame הוא חלון עם קווים כמסגרת, כותרת וכפתורים לסגירה ומזעור של החלון.

אפליקציות המממשות ממשק משתמש מכילות לרוב לפחות חלון אחד מסוג זה.

הערה: קווי המסגרת וכפתורי הסגירה/מזעור של החלון ייראו שונים בין מערכת הפעלה אחת לשניה, ויתאימו אוטומטית לסגנון המקובל באותה מערכת.

דוגמא:

```
import javax.swing.*;

public class HelloWorldSwing
{
	public static void main(String[] args)
	{
		JFrame frame = new JFrame("Hello World Window Title");
		final JLabel label = new JLabel("Hello World");
		frame.getContentPane().add(label);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.pack();
		frame.setVisible(true);
	}
}
```

נקודות לגבי הדוגמא:

- JLabel זהו רכיב המייצג תווית – טקסט סטטי המוצג על המסך שאותו המשתמש איננו יכול לשנות.
- השיטה pack() גורמת לחלון לשנות את גודלו ככה שיתאים בצורה מירבית לרכיבים הנמצאים בו.

את אותה הדוגמא היה ניתן לממש גם בצורה הבאה:

```
import javax.swing.*;

public class HelloWorldFrame extends JFrame
{
	public HelloWorldFrame()
	{
		super("Hello World Title");
		final JLabel label = new JLabel("Hello World");
		getContentPane().add(label);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		pack();
		setVisible(true);
	}


	public static void main(String[] args)
	{
		HelloWorldFrame frame = new HelloWorldFrame();
	}
}
```

במקרה זה אנחנו יורשים מהמחלקה JFrame ומרחיבים אותה, ויוצרים מחלקת חלון חדשה הכוללת את רכיבי התוכנית שלנו.

JDialog

מחלקה זו משמשת ליצירת חלונות משניים עבור התוכנית שלנו.
לאובייקט מטיפוס JDialog אין קיום עצמאי כמו שיש לאובייקט מטיפוס JFrame. אובייקט מטיפוס JDialog חייב להיות קשור לאובייקט קיים מטיפוס JFrame או JDialog (dialog יכול להתווסף ל-dialog אחר).
התלות של אוביקט מסוג JDialog באובייקט אליו הוא קשור מתבטאת במספר תחומים:

- אם החלון אליו הדיאלוג קשור נסגר, הדיאלוג נסגר אף הוא.
- אם החלון אליו הדיאלוג קשור ממוזער, גם הדיאלוג ממוזער.
- אם החלון אליו הדיאלוג קשור מוצג על המסך (אחרי מזעור) גם הדיאלוג יופיע שוב.

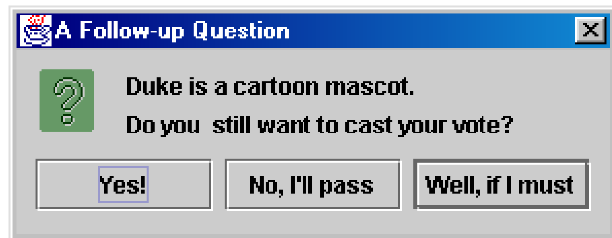
אובייקט חלון מסוגל להיות מוגדר כ-**modal**. חלון המוגדר להיות modal הוא חלון שכאשר הוא פעיל, לא ניתן לגשת לאף חלון אחר באותה הררכית הכלה עד שחלון ה-modal נסגר.
שימושים לחלון מסוג כזה יכולים להיות למשל הצגת מידע דחוף למשתמש עם מספר אפשרויות, שהמשתמש חייב לבחור אחת מהן לפני שיוכל להמשיך להשתמש בתוכנית.

כדי ליצור חלונות מותאמים אישית לצריכנו נשתמש ישירות במחלקה JDialog, בדומה לדוגמאות שראינו בעמודים הקודמים לגבי JFrame.
כמו כן, SWING מכילה מספר חלונות סטדנטריים בהם אנו יכולים להשתמש, לדוגמא: JProgressBar, JFileChooser, JColorChooser ועוד.

המחלקה JOptionPane משמשת ליצירה מהירה של דיאלוגים פשוטים. דוגמא לשימוש בה:

```
      Object[] options = {"Yes!", "No, I'll pass", "Well, if I must"};
      int n = JOptionPane.showOptionDialog(frame,
      "Duke is a cartoon mascot. \n" + "Do you still want to cast your
vote?",
            "A Follow-up Question",
      JOptionPane.YES_NO_CANCEL_OPTION,
      JOptionPane.QUESTION_MESSAGE,
      null,
      options,
      options[2]);
```

והתוצאה:



JComponent

JComponent משמש כמחלקת בסיס לכל האובייקטים הגרפיים מלבד ה-top level containers.
כדי להשתמש ברכיב הנגזר מ-JComponent יש להכניס אותו בהיררכית הוראה שבבסיסה top-level container.
JComponent מספק תמיכה בנושאים שונים, ביניהם:

- אפשרות להתאמה אישית של מראה הבקרים.
- תמיכה במקלדת.
- Tooltip support
- תמיכה באנשים בעלי מוגבליות
- תמיכה במסגרות (borders).
- ועוד..

מודל אירועים

בכל פעם שהמשתמש מקיש על אות, לוחץ על העכבר וכו', קורה אירוע.
כל אובייקט בתוכנית יכול לקבל מידע על כך שהאירוע קרא על ידי רישום עצמו למקור האירוע המתאים.
אובייקטים רבים יכולים להקשיב לאותו אירוע.

דוגמא להוספת פונקציה המקשיבה לאירוע "לחיצת עכבר על כפתור":

```
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    numClicks++;
    label.setText(labelPrefix + numClicks);
  }});
```

EOF