תכנות מתקדם בשפת C#

סדנה בתכנות מונחה עצמים, 2023א׳

אביב נעמן, אורן דה-לם

AvivNaaman/OpenU-OOP

EVENT ORIENTED PROGRAMMING

- פרדיגמת תכנות שמתבססת על אירועים שונים – שלרוב מגיעים מהמשתמש או מהסביבה, בהם הקוד מטפל והם קובעים את זרימת התוכנית.
- כאשר מתרחש אירוע אנו נרצה לקרוא לפונקציה מטפלת – היא מכונה - **Callback**.
- אנו נרצה להעביר לפונקציה מידע אודות מקור האירוע ומידע נוסף אודות האירוע עצמו (Event Args), כדי שהפונקציה המטפלת תקבל הקשר על התרחשות האירוע.

SOLUTION 1: EVENT HANDLER INTERFACE

- לשם מימוש הפרדיגמה, הפתרון הקלאסי הוא שימוש בממשק של Event Handler:

1. הממשק מגדיר בתוכו את פונקציית ה-Callback אשר מטפלת באירוע.
2. יוצרים מחלקה שמממשת את הממשק, ומעבירים מופע שלה ללולאת האירועים.
3. ברגע שהאירוע מתרחש, קטע הקוד שבו הונע האירוע משתמש במופע שהועבר לו לפני כן, ומפעיל את הפונקציה – שקיומה מובטח בזכות מימוש הממשק.

C# SOLUTION: EVENTS & DELEGATES

- אופן הטיפול באירועים שהוזכר לעיל הוא סטנדרטי בשפות תכנות רבות, ביניהן בשפת Java, ולמרות שהוא גמיש ונכון, אופן הבנייה שלו מסובך למדי ודורש כמות גדולה של קוד.
- על כן, שפת C# מספקת מנגנון נוח לטיפול באירועים – Delegates.
- Delegate הוא אופן הגדרה ששקול להגדרת טיפוס של מצביע לפונקציה. בעזרתו מגדירים את החתימה של ה-Callback בשורה אחת. מחליף את הגדרת הממשק.
- לאחר מכן, ניתן להגדיר event מטיפוס ה-Delegate שיצרנו, והוא אוסף מצביעים לפונקציות, שניתן להפעיל ברגע שמתרחש אירוע כלשהו. מחליף את המופע של הממשק.

LAMBDA EXPRESSIONS

- למבדות או פונקציות אנונימיות הן תחביר ב#C שמאפשר הגדרה קצרה ופשוטה לפונקציה, לרוב כדי להעבירה כארגומנט לפונקציה אחרת.
- כדאי שהלמדה תהיה קצרה ומובנת, אחרת עדיף ליצור פונקציה רגילה.
- ללמדות יש גישה לפרמטרים חיצוניים, כלומר, אם אנו מגדירים למדא במקום מסוים, היא תוכל לגשת לכל המשתנים הזמינים בהקשר של יצירתה.
- סינטקס:

```
(arg0, arg1, …) => single_expression
(arg0, arg1, …) => {expression1; expression2;…}
```

FILE SYSTEM WATCHER

- המחלקה FileSystemWatcher (מתוך System.IO) מספקת ממשק נוח אשר "מפקח" ומתריע על שינויים במערכת הקבצים (תיקיות וקבצים) במערכת.
- המחלקה יכולה להאזין לסוגי שינויים רבים לבחירת המשתמש, כגון: זמני יצירה וגישה, גודל, הרשאות, מאפיינים שונים של הקובץ, שמות ומיקומים: NotifyFilter
- המחלקה מאפשרת סינון עפ"י שמות: Filter
- המחלקה מאפשרת פיקוח רקורסיבי בעזרת הפעלת IncludeSubdirectories
- ניתן להוסיף Callbacks מהסוגים: Changed, Created, Renamed, Error, Deleted

OBSERVABLE COLLECTION

- האובייקט ObservableCollection מאפשר מעקב אחרי כל שינוי שמבוצע אליו בעזרת פונקציות.
- זאת אומרת בכל שינוי על ה collection תהיה פונקציה שנקראת עם המידע על השינוי.
- שימושי לדוגמא כאשר יש צורך להעביר רשימה לפונקציה שלקוח כתב ויש צורך להיות מודע ולבצע שינויים בזמן אמת תוך כדי שהלקוח משנה את הרשימה.

LINQ

- **L**anguage **IN**tegrated **Q**ueries
- זהו פיצ׳ר מובנה בתוך שפת C# שמאפשר לבצע פעולות על אוספי נתונים בצורה פשוטה וקלה, ובתחביר שדומה לשפת שאילתות (כמו SQL)
- בעזרת LINQ קל לבצע פעולות מורכבות על אוספים בצורה פשוטה ומובנת
- לLINQ ישנם שני סוגים של תחבירים:

1. תחביר שאילתות
2. תחביר פונקציונלי

LINQ SYNTAX - BASICS

- from *var* in *collection*
- where *condition*
- select *expression*

- מעבר על אוסף
- סינון והתנייה
- אחזור

LINQ SYNTAX - ADVANCED

- קיבוץ
- group *value* by *key* into *var*
- מיון
- orderby *expression*
- איחוד (Join)
- join *variable* in *collection* on *boolean_expression*

QUERYABLE METHODS

- כחלק מ-LINQ, ישנן אוסף נרחב של פונקציות שניתן לבצע על אובייקטים מסוג Queryable.
- הפונקציות הללו כוללות בתוכן: First, Last, FirstOrDefault, Sum, Max, Min, Any, All, Where, Sort, Join, Aggregate, Last, Single, Skip, Zip, Count, Reverse,…
- הפונקציות פועלות על אובייקטים מסוג Queryable – ורבות מהן מקבלות כקלט פונקציות למדא שמבצעות פעולה או מחזירות ערך כנדרש.

XML

- XML הינו פורמט לשמירת מידע.
- הפורמט מדגיש פשטות.
- ניתן להבנה גם אנושית וגם של מחשב.
- תפקידו העיקרי הינו סריאליזציה של מידע, זאת אומרת כפורמט להעברת מידע בין שני מערכות.

LINQ TO XML

- לינק ל-XML זה ספריה של NET. שמאפשרת ניתוח וגישה נוחה לקבצי XML מתוך NET. בעזרת פיצ׳ר הLINQ.
- עיקר הגישה פועל בצורה של שאילתות (דומה לשפת SQL).
- הספרייה היא Lazy Evaluation, משמעו שאין יצירה של המידע עד שמשתמשים בו.

APP CONFIG

- כלי עזר בNET. שמסייע בשמירה של הגדרות (קונפיגורציה) של האפליקציה בצורה פשוטה.
- ניתן להתקנה בגרסאות מודרניות של NET. בעזרת החבילה `System.Configuration.ConfigurationManager`
- המידע נשמר בקובץ `app.config` שמהווה למעשה XML.
- מספק תצורת Key-Value בסיסית בעזרת `appSettings/connectionStrings`.

APP CONFIG: CUSTOM SECTIONS

- מאפשר יצירת סכימות מותאמות אישית, והמרה אוטומטית לאובייקטים.
- בשלב ראשון כותבים מחלקה היורשת מConfiguraitonSection.
- עבור כל מאפיין שרוצים בקונפיגורציה, מוסיפים property עם getter.
- בXML מוסיפים configSection.
- בXML מוסיפים elements או atrributes עם השמות המתאימים לפי ההגדרה במחלקה ובconfigSection.

THREADING

- Thread הינו מעבד וירטואלי. זאת אומרת יחידת עיבוד שמערכת ההפעלה מספקת שמתפקד כמו מעבד.
- מחשבים מודרניים בעלי מספר ליבות יוכלו להריץ כמה Thread-ים באותו הזמן מה שיכול לגרום לשיפור בביצועים.
- כל תהליך במערכת כולל לפחות Thread אחד.

TASKS

- טסקים הינם מנגנון לניהול threadים.
- כאשר רוצים להפעיל רשימה של פעולות באופן מקבילי מריצים כל פעולה על טסק אחד, מערכת dotnet תשלוט בכמות ה threads בפועל ותנסה להריץ בופן אופטימלי את כל ה טסקים.
- למעשה טסקים מאחורי הקלעים מחזיקים thread pool שגדל או קטן לפי הצורך.

ASYNC - AWAIT

- Async – Await הוא מנגנון למניעת המתנה על IO.
- ניתן להסביר בצורה הבאה: כאשר רוצים להכין ארוחת בוקר עם קפה וחביתה אפשר להכין את הקפה ואחר כך את החביטה (פעולות ישירות) או תוך כדי שממתינים שהמים ירתחו בשביל הקפה להתחיל כבר להכין את החביתה.
- תכנון אסינקי ימנע מצב בו ממתינים לפעולה שלא דורשת את המעבד בעוד יש פעולה שכן דורשת את המעבד.
- **הערה: אין מקביליות ב async-await, יש ניצול יותר יעיל של thread בודד.**

EXTENSION METHODS

- דרך פשוטה של שפת C# להוסיף שיטות עזר על אובייקטים, מבלי לרשת את המחלקה.
- על מנת להשתמש במנגנון זה, מוסיפים את המילה השמורה this על הארגומנט הראשון של פונקציה מסוימת:

```
public static bool compatible(this ICar car) {...}
```

- ומיד לאחר מכן ניתן להשתמש בפונקציה כמו כל פונקציה במחלקה של האובייקט, למרות שזו לא פונקציה שמוגדרת בתוך המחלקה:

```
ICar c = Car();
var is_good = c.compatible();
```

REFLECTION

- רפלקציה מאפשר להשיג מידע על אסמבלים (קבצים מקומפלים), מודולים וטיפוסים. אפשר להשתמש גם כדי לייצר טיפוסים חדשים בזמן ריצה.
- אחד השימושים העיקריים ב-Reflection הוא כדי לטעון קוד חיצוני לתוך תוכנה שרצה ולנתח מה מבצע הקוד שנכנס, הטיפוסים שלו, וכו.
- Reflection יהיה שימושי לדוגמא במערכת אינטרנטית שמנהלת סוכנים אוטומתיים, כאשר משתמש (בן אדם) מתחבר לשרת ה #C ורוצה לטעון את הסוכן שלו, הוא פשוט יעביר את האסמבלי של הסוכן שלו שיטען בזמן ריצה בצד השרת.

ATTRIBUTES

- Attributes הינם דרך להוסיף מידע\יכולות לקטע קוד (מחלקה, פונקציה, אסמבלי).
- Attributes יקבלו את הארגומנטים שלהם בזמן קומפילציה, כמובן אפשר להשתמש במשתנים כמו זמן וכו שלא יהיו קבועים בזמן קומפילציה בתוך הקוד שממש את ה attribute.

SERIALIZATION

- סריאליזציה היא דרך לשמור את הנתונים על אובייקט כמו כן מאפשר יצירה מחדש של אותו אובייקט.
- שימושי כדי להעיר מידע דרך האינטרנט או בין תוכנות.
- ישנם מספר סוגים עיקריים של סריאליזציה: JSON, XML, Binary.

JSON

- JavaScript Object Notation
- בדומה ל XML גם Json הינו פורמט לשמירת מידע שמבוסס על התחביר של שפת JavaScript.
- קריא גם לבני אדם וגם למכונות.
- משמש בעיקר כדי להעביר מידע מתוכנה אחת לאחרת בעזרת APIs (לעיתים קרובות דרך האינטרנט).

THANKS!

SOURCE CODE:

AvivNaaman/OpenU-OOP